Biophysics of Human Neutrophil Haptokinesis

Steven Henry

University of Pennsylvania, sjhenry@ptd.net

Follow this and additional works at: http://repository.upenn.edu/edissertations

Part of the Allergy and Immunology Commons, Biomedical Commons, Biophysics Commons, Immunology and Infectious Disease Commons, and the Medical Immunology Commons

Recommended Citation

http://repository.upenn.edu/edissertations/1061

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/edissertations/1061
For more information, please contact libraryrepository@pobox.upenn.edu.
Biophysics of Human Neutrophil Haptokinesis

Abstract
Neutrophils are a type of white blood cell and first responders to tissue trauma and infection. This thesis explores the role of extracellular adhesivity in dictating neutrophil phenotype with respect to cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The principle tool employed is microcontact printing, a powerful method to spatially organize a cell's adhesive environment. We demonstrate the capacity of neutrophils to sense adhesive density on stiff substrates and differentially respond to surfaces with low and high fibronectin content. On low and moderately adhesive surfaces neutrophils assume a highly spread, uropod-absent phenotype reminiscent of keratocytes. On highly adhesive surfaces neutrophils assume the classic amoeboid morphology with an elongated cell body, narrow lamellipodium, and knob-like trailing uropod. Our work reconciles conflicting observations of these two phenotypes previously attributed solely to the underlying stiffness of substrate. The spreading and motility quantified are haptokinetic, induced through the quiescent cell's interaction with immobilized adhesive ligand alone. Function blocking antibody studies implicated the promiscuous Mac-1 integrin receptor in supporting haptokinetic migration. We elucidate the density sensing length scale by presenting high and low adhesive cues to the cells simultaneously. Through rational design of the adhesive domains we conclude that neutrophils sense density at the whole cell length scale, integrating adhesive stimuli over their entire contact interface. Adhesion density sensitivity in stiff microenvironments has applicability to the study of cancer metastasis and particularly the epithelial-to-mesenchymal transition model. We also employ the microfabricated-Post-Array-Detectors (mPADs) traction platform to measure the forces associated with neutrophil spreading. We resolve with high spatial and temporal resolution a highly coordinated protrusive wave front of pN magnitude that propagates radially outwards from the cell center. Small molecule inhibitor studies establish that spreading was not analogous to lamellipodium formation but was sensitive to perturbations of actin cortical stiffness. Lastly, we apply the principles uncovered in neutrophils to the patterning of surface-active microfluidic vesicles by tuning vesicle-substrate adhesion and repulsion at the contact interface. The generation of ordered arrays of micron scale vesicles was a first of its kind.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Bioengineering

First Advisor
Daniel A. Hammer

Keywords
adhesion, microcontact printing, motility, neutrophil, outside-in integrin activation, traction force

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/1061
Acknowledgements

I thank Dan Hammer for inviting me to join his research group and giving me free reign to be curious and explore my interests. Dan has been a patient and enthusiastic advisor. In reflecting upon my graduate training, I recognize that Dan has placed an immense trust in me, and all of his graduate students, by allowing open and frequent collaboration with peers and other research laboratories. I thank my committee members for their guidance and time in serving as advisors to my thesis. Beyond his role as committee member, John Crocker has been an invaluable mentor. I am immensely thankful for John’s energy and willingness to provide thoughtful reflection at every stage of my growth as a scientist. Numerous lab mates past and present were instrumental in my graduate training. Of truly exceptional note were my senior lab mates and collaborators Neha Kamat and Brendon Ricart. Neha was and remains an inspirational scientific role model. Brendon has been an appreciated voice of perspective and encouragement as I conclude my graduate work. Outside the lab, Mike Norton has been a steadfast friend and I will miss our lively and meandering conversations. Most importantly I thank my wife Kate who has equally shouldered the burden of this challenging journey. This body of work is as much a product of her enduring support as my own diligence.
ABSTRACT

BIOPHYSICS OF HUMAN NEUTROPHIL HAPTOKINESIS

Steven J. Henry

Professor Daniel A. Hammer

Neutrophils are a type of white blood cell and first responders to tissue trauma and infection. This thesis explores the role of extracellular adhesivity in dictating neutrophil phenotype with respect to cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The principle tool employed is microcontact printing, a powerful method to spatially organize a cell’s adhesive environment. We demonstrate the capacity of neutrophils to sense adhesive density on stiff substrates and differentially respond to surfaces with low and high fibronectin content. On low and moderately adhesive surfaces neutrophils assume a highly spread, uropod-absent phenotype reminiscent of keratocytes. On highly adhesive surfaces neutrophils assume the classic amoeboid morphology with an elongated cell body, narrow lamellipodium, and knob-like trailing uropod. Our work reconciles conflicting observations of these two phenotypes previously attributed solely to the underlying stiffness of substrate. The spreading and motility quantified are haptokinetic, induced through the quiescent cell’s interaction with immobilized adhesive ligand alone. Function blocking antibody studies implicated the promiscuous Mac-1 integrin receptor in supporting haptokinetic migration. We elucidate the density sensing length scale by presenting high and low adhesive cues to the cells simultaneously. Through rational design of the adhesive domains we conclude that neutrophils sense density at the whole cell length scale, integrating adhesive stimuli over their entire contact interface. Adhesion density sensitivity in stiff microenvironments has applicability to the study of cancer metastasis and particularly the epithelial-to-mesenchymal transition model. We also employ the microfabricated-Post-Array-Detectors (mPADs) traction platform to measure the forces associated with neutrophil spreading. We resolve with high spatial and temporal resolution a highly coordinated protrusive wave front of pN magnitude that propagates radially outwards from the cell center. Small molecule inhibitor studies establish that spreading was not analogous to lamellipodium formation but was sensitive to perturbations of actin cortical stiffness. Lastly, we apply the principles uncovered in neutrophils to the patterning of surface-active microfluidic vesicles by tuning vesicle-substrate adhesion and repulsion at the contact interface. The generation of ordered arrays of micron scale vesicles was a first of its kind.
# Table of Contents

Acknowledgements ........................................................................................................ iii

Abstract ........................................................................................................................ iv

List of Figures .............................................................................................................. ix

Chapter 1: Overview of the Thesis .............................................................................. 1
  Organization ..................................................................................................... 1
  Specific Aims ................................................................................................... 2
    Aim 1: Quantify effect of adhesion density on neutrophil shape
    and motility .......................................................................................... 3
    Aim 2: Elucidate the length scale of neutrophil density sensing ......... 4
    Aim 3: Measure the forces associated with adhesion-driven
    spreading of neutrophils ........................................................................ 4
    Aim 4: Spatially organize acellular microfluidic vesicles into arrays
    using surface adhesion ......................................................................... 5

Chapter 2: Background ................................................................................................ 7
  The Human Neutrophil .................................................................................... 7
  Microcontact Printing ...................................................................................... 11
  microfabricated-Post-Array-Detectors ............................................................. 13
  Models of Cell Spreading ................................................................................ 16
  References ........................................................................................................ 21

Chapter 3: Ligand Density Elicits a Phenotypic Switch in Human Neutrophils ......... 27
  Preface .............................................................................................................. 27
  Abstract ............................................................................................................ 27
  Introduction ...................................................................................................... 28
  Materials and Methods ..................................................................................... 29
    Media ................................................................................................... 29
    Substrates ............................................................................................. 30
    Protein Deposition and Blocking ......................................................... 31
    Neutrophil Isolation ............................................................................. 32
    Cell Motility Experiments .................................................................... 32
    Integrin Blocking ................................................................................. 34
  Results and Discussion .................................................................................... 34
    Observation of Neutrophil Phenotypes on Two Different Substrates ........................................................................ 34
    Quantifying Motility of Amoeboid and Keratocyte-Like Phenotypes 44
    Flow Cytometry to Assess Activation State ....................................... 46
    Effect of Chemoattractant on Keratocyte-Like Motility ................... 48
    Identifying Integrin Chains Responsible for Adhesion ....................... 53
  Conclusions ...................................................................................................... 57
  Acknowledgements .......................................................................................... 57
Chapter 4: Human Neutrophil Adhesion Density Sensing at the Whole Cell Length Scale

Preface
Abstract
Introduction
Materials and Methods
Media and Reagents
Substrate Production
Neutrophil Isolation
Quantitative Fluorescence Microscopy
Cell Motility Experiments and Data Analysis
Results
Engineering Substrates to Present Neutrophils with Two Adhesive Length Scales
Neutrophils Integrate Adhesive Stimulation Across Entire Contact Interface
Comparable Neutrophil Motility on Discrete Island and Continuous Fields
Discussion
Acknowledgements
References

Chapter 5: Protrusive and Contractile Forces of Spreading Human Neutrophils

Preface
Abstract
Introduction
Materials and Methods
Media and Reagents
microfabricated-Post-Array-Detectors (mPADs) and Microcontact Printing
Calculation of Post Spring Constant
Neutrophil Isolation
Spreading Experiments
Antibody Blocking and Cytoskeletal Inhibitor Studies
Cell Profile Imaging
Data Analysis
Identification of Cell-Engaged Posts
Cell Reference Frame Coordinate System
Computing a Spreading Velocity
Results and Discussion
Neutrophil Spreading on mPADs
Metrics of Spreading and Contractility
Biochemical Perturbations of the Cell Cytoskeleton
References
Chapter 6: Single Vesicle Patterning of Uniform, Giant Polymersomes Into Microarrays .......................................................... 132
Preface .............................................................................................................. 132
Abstract .......................................................................................................... 132
Introduction ..................................................................................................... 133
Materials and Methods .................................................................................. 135
  Reagents ........................................................................................................ 135
  Polymersome Preparation ........................................................................ 136
  Polymersome Functionalization ................................................................ 136
  Substrate Fabrication .................................................................................... 137
  Vesicle Patterning .......................................................................................... 137
Results and Discussion .................................................................................... 138
  Producing Monodisperse Populations of Micron-Scale Microfluidic Vesicles ........................................................................ 138
  Controlling Microfluidic Payload Encapsulation ...................................... 141
  Surface Functionalizing Vesicles for Controlled Adhesion ....................... 141
  Spatially Patterning Arrays of Functionalized Vesicles and an Application .......................................................... 142
Conclusions ...................................................................................................... 153
Acknowledgements .......................................................................................... 153
References ........................................................................................................ 154

Chapter 7: Future Directions ........................................................................................ 156
Visualizing the Cytoskeleton ........................................................................... 156
  Fixation and Vinculin Staining Method Notes ......................................... 161
Neutrophil Motility on mPADs ....................................................................... 162
  Additional Small Molecule Inhibitor Work .................................................. 170
Vesicle Haptokinesis ........................................................................................ 171
References ........................................................................................................ 175

Appendix A: Custom MATLAB Code for Analysis of Neutrophil Motility ....... 177
  Introduction .................................................................................................... 177
  Methodology .................................................................................................. 177
  Code ............................................................................................................... 184

Appendix B: Custom MATLAB Code for Analysis of Neutrophil Spreading ....... 307
  Introduction .................................................................................................... 307
Methodology ........................................................................................................ 309
Areas for Code Improvement ........................................................................ 314
Code ............................................................................................................... 314
References .................................................................................................... 386
List of Figures

Figure 3.1  Two neutrophil morphologies.................................................................35
Figure 3.2  Phenotype does not follow method of protein deposition .................37
Figure 3.3  Keratocyte-like phenotype recapitulated on Pluronic-blocked glass................................................................................................38
Figure 3.4  Exquisite cell-ligand specificity on Pluronic-blocked substrates .......39
Figure 3.5  Microcontact printing overview and sub-saturating density quantification ..................................................................................40
Figure 3.6  Neutrophil phenotype on increasing densities of fibronectin (FN) ...42
Figure 3.7  Mean squared displacements (MSDs) of two motility modes............45
Figure 3.8  Quantification of L-selectin expression levels *via* flow cytometry ...47
Figure 3.9  Quantification of neutrophil haptokinesis and chemokinesis of keratocyte-like phenotype.................................................................50
Figure 3.10 Sample sizes per condition and complete results of model-independent significance testing..........................................................51
Figure 3.11 MSDs of all independent observations of FN/fMLF experimental conditions tested........................................................................................52
Figure 3.12 Integrin blocking on FN........................................................................54
Figure 3.13 Neutrophil adhesion to BSA ................................................................56
Figure 4.1  Stamp-off method of microcontact printing to generate island arrays................................................................................................69
Figure 4.2  Stamp-off method of microcontact printing to generate island arrays: no contrast enhancement...........................................................70
Figure 4.3  Measurement of mean printed island diameter and pitch ..................71
Figure 4.4  Engineering islands with two adhesive length scales .........................73
Figure 4.5  Neutrophils sense density at whole cell length scale .........................75
Figure 4.6  Observation of each phenotype per experimental condition..............76
Figure 4.7  Neutrophil motility on islands is comparable to low density continuous fields ..................................................................................78
Figure 4.8  MSD analysis of neutrophil motility on islands and fields..............79
Figure 5.1  Human neutrophil spreading on fibronectin printed mPADs ..........97
Figure 5.2  Identification of cell-engaged posts by variance analysis...............98
Figure 5.3  Cell reference frame coordinate transformation schematic ..........100
Figure 5.4  Spatial dichotomization of force trajectories..............................101
Figure 5.5  Characterizing protrusion and contraction using the ensemble of
neutrophil spreading events .....................................................................104
Figure 5.6  Per-cell spreading velocity analysis.............................................105
Figure 5.7  Cytoskeletal perturbation via small molecule inhibitors ............109
Figure 5.8  XZ kymograph of neutrophil spreading on stiff mPADs.............110
Figure 5.9  Vertical profiles of neutrophils imaged via confocal microscopy ....113
Figure 5.10 Metrics of XZ and XY cell profiles .............................................114
Figure 5.11 Confocal images of 0.49 μm diameter green fluorescent beads
using the same magnification and acquisition settings as post
arrays ........................................................................................................118
Figure 5.12 Confocal measurements of FN-AlexaFluor488 printed post arrays
in a solution of 90% glycerol ...................................................................119
Figure 5.13 Post invagination as origin of protrusive signature......................122
Figure 6.1  Controlling vesicle size and loading.............................................139
Figure 6.2  Controlling vesicle diameter by adjusting the continuous phase
flow rate ......................................................................................................140
Figure 6.3  Functionalizing polymersomes via biotinylation..........................143
Figure 6.4  NeutrAvidin (NAv) functionalization of polymersomes made
from biotin-modified polymer ..................................................................144
Figure 6.5  Patterning single polymersomes ..................................................146
Figure 6.6  Polymersome capture on NAv-printed surfaces .........................147
Figure 6.7  Polymersomes patterned on NAv islands of 50 μm diameter
and 100 μm pitch ......................................................................................148
Figure 6.8  Effect of NAv surface area on non-specific binding ....................149
Figure 6.9 Polymersomes patterned on smaller NAv islands of 10 μm diameter and 50 μm pitch ..........................................................151

Figure 6.10 Creating sensor arrays ........................................................................152

Figure 7.1 Loss of phenotype after paraformaldehyde fixation .......................157

Figure 7.2 Improved phenotype stability and vinculin plaque detection with microtubule stabilizing buffer fixation ..............................................159

Figure 7.3 Summary of mPADs specifications ...................................................163

Figure 7.4 Neutrophil motility on post arrays is stiffness dependent ...............165

Figure 7.5 Neutrophil traction generation is stiffness dependent......................168

Figure 7.6 Attempted vesicle haptokinesis experiment .....................................172

Figure A.1 Neutrophil motility data analysis workflow .................................178

Figure B.1 Resting lattice bitmap reconstruction .............................................308
Chapter 1

Overview of the Thesis

Organization

Broadly speaking, the unifying theme of this thesis work was biological adhesion. The majority of investigations contained herein explored how a particular human immune cell called the neutrophil responded to environmental adhesive cues in terms of motility, mechanical force generation, and the molecular constituents involved in those two processes. Additionally, studies on the application of biological adhesion and repulsion to acellular vesicle patterning were also pursued. In all studies, the unifying methodology employed was microcontact printing. This is a powerful process for spatially organizing the adhesive environment of cellular and acellular systems. Using microcontact printing we engineered environments to study the response of neutrophils and surface active microfluidic vesicles to extracellular adhesive cues.

The thesis is organized into seven chapters and two appendices. Chapter 2 presents the reader with background sufficient to contextualize the experimental results. It includes a general overview of the biological role of neutrophils in the body and the evolution of microcontact printing as a technological platform. Chapter 3 explores adhesion density as a controller of neutrophil shape and migratory phenotype. Elucidating the receptor responsible for mediating this density-sensitive adhesion is also presented. The custom MATLAB codes developed to analyze the motility data are provided in Appendix A. Chapter 4 addresses an outstanding question that arises from the prior
chapter, namely what is the length scale of the neutrophil’s sensitivity to adhesive density? By rational design of hybrid surfaces in which high and low adhesive stimulation is presented to the cells simultaneously we address this question. In Chapter 5 we consider the mechanics associated with adhesion-induced spreading prior to the onset of the motility observed in Chapters 3 and 4. Here we employed a traction platform to measure the protrusive and contractile forces associated with spreading. A substantial portion of this chapter is also devoted to elucidating the cytoskeletal components involved in the protrusive and contractile phases of spreading. The custom MATLAB codes developed to analyze the mechanical spreading data are provided in Appendix B. Chapter 6 transitions to acellular microfluidic vesicles which have been rendered surface active through biotinylation. The role of interface adhesion and repulsion is explored in patterning these vesicles into regular arrays. Chapter 7 explores future directions for further inquiry. Importantly, preliminary and pilot experimental observations accompany these recommendations.

The work contained herein is the product of years of collaborative effort with researchers inside and outside the Hammer laboratory. In that spirit, a preface to each experimental chapter is included which lists the co-authors involved in that chapter’s studies as well as their individual contributions. The preface also details if the content of that chapter has been published in a peer-reviewed journal or is under review.

**Specific Aims**

The following specific aims are provided to help organize the thesis content and articulate the motivations for pursuing each aim as well as explicitly state the hypotheses
being tested. Because in multiple cases the initial hypotheses were rejected, the principal experimental outcomes of the aims are also summarized.

*Aim 1: Quantify effect of adhesion density on neutrophil shape and motility*

**Motivation:** Two characteristic migratory phenotypes have been reported of neutrophils on planar (unconstrained two dimensional) surfaces: amoeboid and keratocyte-like. Observations of keratocyte-like phenotype were previously ascribed uniquely to the underlying stiffness of the migratory surface. However, scattered observations of both phenotypes were present on substrates of equivalent stiffness throughout the literature. The goal of this aim was to reconcile these disparate observations by considering the effect of surface adhesivity on neutrophil shape and mode of migration.

**Hypotheses:** Neutrophil shape and motile phenotype are strongly controlled by the underlying adhesivity of the extracellular environment. Integrin receptors will mediate this adhesion.

**Outcomes:** Neutrophil shape and motile phenotype were dictated by adhesion density on equivalently stiff substrates. The differences were qualitatively and quantitatively distinct. On highly adhesive surfaces, neutrophils assumed the classic amoeboid phenotype having a narrow elongated cell body, moving quickly, and performing frequent directional changes. On low and moderately adhesive surfaces neutrophils assumed a phenotype reminiscent of fish epithelial keratocyte cells featuring a highly spread lamellipodium, moving slowly, but in a directionally persistent manner. Adhesion was found to be mediated by the promiscuous MAC-1 ($\alpha_M\beta_2$) receptor. We demonstrated this promiscuity by recapitulating the findings on a second adhesive ligand.
Aim 2: Elucidate the length scale of neutrophil density sensing

Motivation: In the previous aim we found that neutrophils assumed the amoeboid phenotype on highly adhesive surfaces and the keratocyte-like phenotype on low and moderately adhesive surfaces. The goal of this aim was to consider the length scale of this density sensing and present neutrophils with hybrid environments in which high and low density cues were presented to the cells simultaneously.

Hypothesis: If neutrophils sense density on the submicron length scale they will assume the amoeboid phenotype on discrete islands of high density protein, despite the total protein content across the cell-substrate interface being low. Conversely, if neutrophils integrate adhesive stimulation across their cell bodies they will assume the keratocyte-like phenotype on the same surfaces despite the submicron, on-island protein density being high.

Outcomes: Neutrophils assume the keratocyte-like phenotype on discrete islands in which the on-island (local) protein density was high but the area average (global) protein density was low. By careful design and validation of these hybrid surfaces we were able to conclude that neutrophils integrate adhesive stimulation over their entire cell-substrate contact interface and respond to discrete islands as if they were a continuous field of low density protein.

Aim 3: Measure the forces associated with adhesion-driven spreading of neutrophils

Motivation: In the previous two aims the focus was on quantifying long time (i.e. minutes and hours) motility elicited by surface adhesivity. However neutrophil spreading was a prerequisite to the onset of motility and was known be a temporally fast (i.e.
seconds) phenomenon. Here the goal was to quantify the forces associated with neutrophil spreading and identify the molecular components involved.

**Hypotheses:** Neutrophil spreading will be an active process analogous to lamellipodium formation in which actin polymerization and branching protrude the cell membrane outwards.

**Outcomes:** Adhesion driven spreading was sufficiently forceful to generate detectable deflections on the order of pN. However, using inhibitors of various cytoskeletal components, we demonstrated that this protrusion was not the result of lamellipodium formation. Rather we showed that adhesion-driven spreading was a competition between surface energy at the cell-substrate interface and resistance to shape change imparted by the cell cortical tension. Protrusion was observed because a small degree of adhesive protein on the sidewall of pillar tips induces the cell to spread through a finite volume of pillar tips.

_Aim 4: Spatially organize acellular microfluidic vesicles into arrays using surface adhesion_

**Motivation:** Regular arrays of pay-load capable micron scale acellular vesicles were lacking as an experimental platform for the study of vesicle sensing and inter-vesicle communication. Our goal was to realize such an array by merging the technological platforms of microcontact printing and microfluidic vesicle generation.

**Hypotheses:** Adhesive islands via microcontact printing will stabilize micron-scale microfluidic vesicles rendered surface active through biotinylation. Interstitial pluronic will prevent non-specific vesicle adhesion.
Outcomes: We demonstrated a first of its kind organization of micron scale microfluidic vesicles into regular arrays by adhesion stabilization. Although we initially hypothesized that the role of between-island PEGylation was to inhibit non-specific binding we actually found that the PEG brush in these interstitial spaces interacted with the PEG brush on the vesicle membrane and induced vesicle mobility through steric repulsion. Once on an adhesive island, biotin-avidin ligation stabilized the vesicle against further transit provided the adhesive plaque was sufficiently large.
Chapter 2

Background

The Human Neutrophil

The complex and distributed organ that is the mammalian immune system is often dichotomized in terms the innate and adaptive branches for the sake of pedagogy. The innate branch consists of physical barriers to pathogen challenges (e.g. epidermis and mucosal films) as well as the family of terminally differentiated granulocytes (e.g. neutrophils and eosinophils) that are capable of executing their response to infection and trauma on the timescale of seconds and minutes. While fast, the response is not highly specific. Conversely the adaptive branch is comprised of the family of T-cells and B-cells that mount a highly specific pathogen response and confer long term immunological memory to the host organism. However, this exquisitely specific response requires a latency period of hours and days to fully develop. Thus, the two branches are complementary and collectively ensure the temporal continuity of the host organism’s immunity. In reality the distinction is entirely pedagogical as the constituent elements are intimately and continuously engaged in a dynamic cross talk (1).

Neutrophils, as do all blood cells, originate in the bone marrow from hematopoietic precursor cells. They represent the largest fraction of blood cells continuously generated at ~ $10^{11}$ neutrophils per day in healthy adults. Once matured these cells are equipped with a variety of terminal functions including pathogen engulfment (phagocytosis), secretion of soluble chemical cues (cytokines), production of
reactive oxygen intermediates (ROIs), and the controlled release of nuclear DNA to form nuclear-extracellular-traps (NETs) (2).

A prerequisite to the execution of any of these terminal functions is the cell’s arrival at the locus of trauma (3) or infection (4) via vascular rolling, extravasation, and extravascular migration (5). This spatially and temporally controlled sequence of events is collectively known as the leukocyte adhesion cascade. Our detailed molecular understanding of this sequence is the result of decades of empirical observations in vivo and in vitro.

In the vasculature, quiescent neutrophils transiently roll along the endothelial cell wall in a selectin-mediated capacity and can be induced to arrest after encountering endothelial-immobilized chemokines and chemoattractants (6). Arrest is achieved through integrins, a family of heterodimeric receptors expressed on the cell surface, which ligate a variety of extracellular adhesive ligands and enable cell anchorage. The two integrins of predominate importance in the leukocyte adhesion cascade are LFA-1 (αLβ2) and MAC-1 (αMβ2), both of which ligate the ICAM-1 adhesive ligand which is also expressed on the endothelial cell surface (7). LFA-1 and MAC-1 work cooperatively to enable neutrophil firm arrest with the later being particularly sensitive to chemoattractant stimulation (8). In a process known as inside-out integrin activation, chemokine ligation by cell-surface G-protein coupled receptors induces increased MAC-1 affinity and, consequently, cell adhesiveness (9). Following firm arrest, neutrophils exit the vasculature between or through endothelial cells and migrate to the tissue wound site (5).
In addition to soluble and immobilized chemical cues that direct immune cell response and function (10-15), cells encounter numerous physical cues in the body (e.g. stiffness, dimensionality, adhesivity, and roughness) that are strong determinants of shape, force generation, and gene expression (16-17). Blood cell response to physical cues such as substrate rigidity (15, 18-21), confinement (22-23), shear force (24), and adhesion density (25-26) have been areas of on-going investigation. The focus of this thesis is the role of adhesive ligand density on directing neutrophil phenotype in terms of cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The motivation for considering this particular environmental factor stemmed from the desire to reconcile two conflicting morphological observations of neutrophil shape and motility in the literature.

On a majority of two-dimensional in vitro substrates, neutrophils are reported to exhibit an amoeboid morphology (27-33). The distinguishing features of this phenotype are a narrow, elongated cell body with a frontward ruffled-lamellipodium and rearward knob-like uropod (27). Detailed images of this morphology have been captured with high resolution scanning electron microscopy (SEM) (34-35). However, there have also been observations of neutrophils assuming a very different, well-spread, uropod-absent, phenotype on two-dimensional substrates (15, 18-19). In those instances the alternative phenotype was attributed exclusively to the underlying stiffness of the material, as neutrophil spread area was shown to increase with increasing substrate rigidity. Yet, the amoeboid phenotype was also observed on stiff substrates such as those in the previously mentioned SEM studies. This suggested to us that substrate stiffness was not a unique controller of neutrophil morphology. Hypothesizing that another factor was involved in
modulating these two phenotypes, our work focused on the role of adhesion ligand density. This hypothesis was motivated by observed adhesion density sensitivity in fish keratocytes (36) and computational predictions of adhesion density effects on cell motility (37).

It is important to comment on the nature of the neutrophil-surface interaction we are exploring in this thesis as it differs in considerable ways from the adhesive interactions just discussed in the context of the leukocyte adhesion cascade. The basis of the empirical work presented in subsequent chapters is neutrophil adhesion, motility, and force generation induced by haptokinetic interaction of the cell with fibronectin under static (no flow) conditions. Fibronectin (FN) is a large glycosylated adhesive ligand present in blood and extracellular matrices (38). Through function blocking antibody studies we ultimately attribute our observed neutrophil adhesion and motility to MAC-1 (Fig. 3.12). We suggest that the promiscuity of this receptor for multiple adhesive ligands demonstrated by us (Fig. 3.13) and others (39-40) confers the neutrophil with the ability to mount an immunologic response even the absence of canonical selectin-mediated rolling and chemoattractant-induced firm arrest.

To validate that our experimental platform was a valid model of haptokinetic stimulation, we used L-selectin as a marker of neutrophil quiescence. L-selectin expression levels are a sensitive indicator of a neutrophil’s transition from quiescence (high expression) to a phenotype primed for integrin-mediated binding (low expression) (41). Our series of flow cytometry control experiments revealed that while neutrophils were capable of chemoattractant-stimulated (inside-out) integrin activation, they were not primed for binding prior to FN exposure (Fig. 3.8). As such our model of neutrophil
haptokinesis is itself a model of the alternative integrin activation pathway called outside-in stimulation. In contrast to inside-out activation, the outside-in pathway is whereby low affinity integrin-ligand interactions induce a high affinity conformational change in the integrin receptor (42).

**Microcontact Printing**

To probe neutrophil response to adhesive ligand density and organization we employed microcontact printing (43). The ability to pattern molecules on the micron scale was pioneered by the Whitesides group and initially demonstrated by organizing self-assembled monolayers of alkanethiols on gold substrates using elastomeric stamps to achieve transfer (44). Elastomeric stamps of poly(dimethylsiloxane) (PDMS), sold under the trade name Sylgard 184 by Dow Corning, are themselves patterned by casting against silicon wafers etched by standard photolithographic techniques (45). After the PDMS stamps are cured, the peeled stamp has the complimentary topography of the silicon master against which the stamp was initially cast.

This engineering approach to spatially organizing a cell’s adhesive environment has been widely used to probe integrin clustering (46-47), effect of cell shape on viability (48) and focal adhesion architecture (49), and the role of extracellular matrix distribution on cell spreading (50) in the context of mesenchymal cells. In this thesis we apply the same principles to elucidate the effect of adhesive ligand density distribution on human neutrophils. Only recently has microcontact printing been applied in studies of hematopoietic-derived cells (12, 14, 51-52).

The form of microcontact printing employed in this thesis is the direct patterning of protein molecules onto PDMS surfaces. This is facilitated by the fact that PDMS,
natively hydrophobic, can be rendered hydrophilic by plasma oxidation (53-54). The differential hydrophobicity of the stamp and substrate are critical in mediating protein transfer at the interface (55). In our experiments a thin layer of PDMS, spun on glass coverslips and cured, is rendered hydrophilic by treatment in ultraviolet ozone (UVO) (47). When a protein-inked PDMS stamp (hydrophobic) contacts a UVO-exposed PDMS coverslip (hydrophilic), a preferential transfer of protein from the hydrophobic stamp to the hydrophilic coverslip occurs. In the case of a flat stamp a continuous field of protein is transferred to the substrate. In the case of a stamp with topographical features a discretized pattern of protein is transferred. A schematic of this process is provided in Figure 3.5.

A variation of microcontact printing we also employ to organize the spatial adhesive environment of neutrophils is referred to as “stamp-off”. Stamp-off was developed by Desai and coworkers and exploits the tunable hydrophobicity of PDMS by inserting an additional step in the normal microcontact printing work flow (47). Prior to contacting a flat, uniformly inked PDMS stamp (hydrophobic) to a plasma-treated PDMS coverslip (hydrophilic), the original stamp is itself contacted by a plasma-treated PDMS stamp (hydrophilic) with topographical features. The result is selective removal (i.e. “stamp-off”) of protein from the uniformly inked stamp. A schematic of this process is provided in Figure 4.1. Successive iterations of stamp-off and re-inking can result in complex patterns of multiple adhesive ligands on a flat stamp (47). The advantage stamp-off affords is the elimination of failure modes like feature collapse and ceiling sag that direct stamp-on methods can succumb (43).
The final step in microcontact printing on PDMS substrates is passivation or blocking of the bare regions not occupied by protein. This is accomplished by incubation of the printed substrate in a dilute solution of the non-ionic triblock copolymer Pluronic F-127 (poly(ethylene oxide)-poly(propylene oxide)-poly(ethylene oxide)). The PPO core is hydrophobic and organizes along the PDMS interface while the PEO termini are hydrophilic and project into the aqueous environment. The result is a passivated surface resistant to neutrophil binding (Fig. 3.4). Pluronic mediated resistance is critical as conventional passivation strategies (e.g. albumin incubation) induce non-specific neutrophil adhesion. In this thesis the neutrophil adhesion and motility are entirely attributable to the printed adhesive ligand with no off-ligand effects. Pluronic passivation is stable for days (56) and possible on a variety of tissue culture amenable substrates including glass and tissue culture plastic (57). Lastly, Pluronic is amenable to fluorescence tagging which can facilitate quantitative measurements of the extent of surface loading (58).

**microfabricated-Post-Array-Detectors**

A number of traction platforms have been developed to measure a cell’s mechanical interaction with its surrounding microenvironment. These include wrinkling silicone films (59), two-dimensional (60-61) and three-dimensional (62) bead-in-gel systems, fluorescent markers in PDMS sheets (63), micromachined cantilevers (64), as well as moderately dense (65) and ultra-dense (66) arrays of vertical polymeric posts (microfabricated-Post-Array-Detectors or “mPADs”). In this thesis we employ mPADs to quantify the forces associated with neutrophil spreading as the system represents a natural extension of the PDMS-based work done to assess adhesive ligand density sensitivity in
the same cells. A note on nomenclature, the descriptors “mPADs”, “post arrays”, and “pillar arrays” are used interchangeably throughout this thesis.

The same basic fabrication scheme previously described to achieve topographical stamps for microcontact-printing (i.e. photolithographic etching of a silicon master) is utilized to generate mPADs. However, in the case of mPADs, cells are in a sense induced to spread on the topographical “stamp” itself. This is a subtle over-simplification as the topographical stamp actually serves as a negative relief of the silicon master, used to cast a replica of the silicon master features in PDMS prepolymer onto thin glass coverslips. Extensively annotated protocols for the fabrication of mPADs, their functionalization, and the associated analyses of cell deflections have been made publically available by the Chen group (67-68).

There are a variety of stiffness definitions to describe the discretized environment a cell experiences on mPADs. Figure 7.3 summarizes the specifications of each post array discussed in this thesis as well as a number of stiffness metrics. On the simplest level, each pillar can be modeled as a cantilever subjected to a load at its unconstrained terminus (see derivation of Eq. 5.4) (69) in which case the material spring constant \( k_{\text{spring}} \) is a natural description of pillar stiffness. Pillar stiffness is altered by changing the diameter, length, or modulus of the polymer used for casting. In this thesis stiffness is tuned by varying diameter and length parameters. Important theoretical work done by Schoen and coworkers established a series of corrections to these spring constants as a function of post aspect ratio. The corrections account for the contribution of pillar tilting and base warping to the measured free terminus deflection (70). The magnitude of the correction decreases nonlinearly with increasing post aspect ratio (i.e. the ratio of the post
diameter to its width). While the spring constant is a natural description of single pillar stiffness, it is ambiguous with respect to the macroscopic or ensemble (multi-post) stiffness perceived by a cell spanning many such posts. In a macroscopic context it is more natural to describe the arrays in terms of a Young’s modulus (E) or shear modulus (G) (71). Alternatively, Ladoux and coworkers developed a theoretical description of effective array stiffness by solution of the Green’s function for a discretized substrate (under certain governing assumptions) (72). The Ladoux model estimates the Young’s moduli of post arrays as being substantially softer than anticipated by a local pure shear model. While different definitions of stiffness yield different stiffness values, relative differences within a given experiment can yield insights into the effect of microenvironment stiffness on biological function.

The nature of the cell-pillar interaction has been an active area of study. Through confocal microscopy and reconstruction of the vertical profile of cell-engaged pillars, Lemmon and coworkers established that shear is a greater contribution to post deflection than torque (73). A criticism sometimes leveled against mPADs as a traction platform is the discretized nature of the cell-substrate interaction. Work by Lenhert and coworkers in mesenchymal cells (50) as well as our own findings in neutrophils (Chapter 4) demonstrate that this discretization is not prohibitive and does not produce anomalous biological phenotypes. In fact the integrated response of local adhesive stimuli over the cell body to induce a coordinated whole cell response, yields important insight into the nature of the intracellular signaling at play during cell spreading and traction generation.
Models of Cell Spreading

In Chapter 5 we report the first measurements of neutrophil protrusive force during adhesion ligand induced spreading. The treatment of cell spreading as a physical phenomenon, a thermodynamic competition between the energy of the adhesive environment driving the cell to spread and the cell’s cohesive forces resisting shape change, has a long history. S. B. Carter first observed a preference in mouse fibroblasts for adhesion to high density palladium-shaded glass over low density (74). Carter dubbed fibroblast motility up a gradient of adhesivity “haptotaxis.” Importantly he drew a physical analogy with the wetting of a liquid droplet on a surface. The equilibrium shape of such a droplet in an aqueous medium is described by Young’s equation relating the angle of the droplet-substrate interface to the substrate-medium, droplet-medium, and substrate-droplet interfacial energies. In the same manuscript as Carter’s haptotaxis observations, J. L. Moilliet modeled the fibroblast as a liquid droplet subject to Young’s equation and then extended the model to include a third liquid interface (a thin shell surrounding the liquid droplet) in an attempt to recapitulate the effect of the cell membrane enclosing the cell cytoplasm (74). The incorporation of this additional boundary interface had the effect of increasing the observed contact angle. The conclusion was that the cell membrane imparts the cell with a greater resistance to spreading than a purely liquid model alone would predict.

Single cell micropipette aspiration of quiescent (i.e. not spread) neutrophils has been reliably used to measure neutrophil cortical tension and cytoplasmic viscosity. In nearly all cases, passive neutrophils are modeled as simple viscous fluid drops with constant surface tension, a model also referred to as the Newtonian liquid drop model.
(75-79). While the Waugh group has demonstrated some non-Newtonian characteristic in the neutrophil cytoplasm, namely a power law relation between viscosity and shear rate (80), the prevalent treatment is of the resting neutrophil as a passive viscous liquid drop with apparent surface tension.

Actin in a quiescent neutrophil is confined to a thin cortical shell proximal to the cytoplasmic membrane (81). Disruption of actin polymerization kinetics with the small molecule inhibitors jasplakinolide and cytochalasin B has been found to alter the measured surface tension of resting neutrophils subjected to micropipette aspiration. Jasplakinolide induces actin polymerization and stabilizes filamentous actin (82). In neutrophils, pretreatment with jasplakinolide has been shown to increase the rigidity of the cortex as measured by micropipette aspiration (83). Conversely, cytochalasin B is known to dramatically reduce the rate of actin polymerization and simultaneously interfere with filament-filament interactions that stabilize the actin network (84). In contrast to jasplakinolide stiffening in neutrophils, pretreatment with cytochalasin B has been shown to decrease cortical rigidity as measured by micropipette aspiration (78). In this thesis we extend these observations made in passive, non-adherent neutrophils to neutrophils undergoing adhesion-induced spreading. We find that spreading is completely abrogated following cortical stiffening (via jasplakinolide) while the velocity of spreading is dramatically reduced following cortical softening (via cytochalasin B).

More recent work by Cuvelier and coworkers considers the dynamics of cell spreading (in terms of contact area radius as a function of time) when modeled as a viscous shell that encloses a liquid droplet (85). McGrath provides a concise summary of Cuvelier’s model in reference (86). The model predicts two spreading regimes. At short
times contact radius evolves as $R \sim t^{0.5}$. At long times the adhesive patch is comparable to the size of the cell and contact radius evolves as $R \sim t^{0.25}$. The Cuvelier group considers this model of a viscous cortex encasing a liquid droplet to be broadly applicable as they empirically demonstrate its relevance to the spreading of mesenchymal carcinoma cells (HeLa and S180) and biotinylated red blood cells.

However, the appropriateness of the Cuvelier model in recapitulating neutrophil spreading dynamics is debatable. Recently, Waugh and coworkers observed that neutrophil spreading on fields of the chemoattractant IL-8 exhibit a linear increase in contact radius with time and a logarithmic deceleration in spreading velocity (87). We estimate from their published data that contact radius evolves as roughly $R \sim t^{0.8}$, a spreading rate significantly faster than the short time ($R \sim t^{0.5}$) and long time ($R \sim t^{0.25}$) regimes predicted by the viscous shell surrounding a liquid drop model. Additionally, Waugh and coworkers observe that the neutrophil contact area grows so quickly its diameter exceeds the equatorial diameter of the quiescent neutrophil for the majority of the experimental observation. This implies that if a comparison is to be made it must be made within the long time viscous dissipation regime of Cuvelier’s model. A regime where the discrepancy between Waugh’s empirical $R \sim t^{0.8}$ and Cuvelier’s predicted $R \sim t^{0.25}$ is even more conspicuous.

However, in this thesis we report neutrophil spreading velocities of adhesion-driven spreading on fibronectin printed mPdAdS, a model of haptokinetic spreading. While we have limited resolution of the evolution of the spreading neutrophil’s contact interface with time, because we are tracking fluorescent post tips and not the neutrophil membrane itself, we can still approximate the spreading velocity in terms of the
propagation rate of the radial protrusive force (Fig. 5.1 and Fig. 5.5 C). We estimate that
our neutrophil contact interface grows as \( R \sim t^{0.4} \) which is reasonably close to the short
time \( R \sim t^{0.5} \) dependency predicted by Cuvelier in the viscous shell surrounding a liquid
drop model. It is interesting to note that Waugh and coworkers observed a decrease in
spreading velocity when the IL-8 receptor (CXCR1) was blocked coupled with the fact
that we observe a slower spreading velocity in neutrophils on the adhesive ligand
fibronectin as compared to the chemoattractant IL-8. While intracellular signaling is
important in both cases (blocking IL-8 and FN receptors reduces or eliminates spreading,
see Fig. 5.9 A, ii) it does appear that neutrophil spreading on FN induces a mechanically
distinct response as compared to IL-8. The increased spreading velocity on immobilized
chemoattractant may have biologically relevant implications as chemoattractant
immobilization and expression on endothelial cells is a critical homing cue, inducing
neutrophils to execute the leukocyte adhesion cascade and exit the vasculature at the
locus of trauma or infection (6).

Our observation of neutrophil spreading on FN post arrays is also consistent with
previous Hammer laboratory measurements of neutrophil spreading rates on FN as
measured by RICM (88) where the total cell contact radius grew as approximately \( R \sim t^{0.45} \). There are significant points of departure from the Cuvelier RICM validation
experiments in which the region of intimate cell-substrate contact in mesenchymal cells
grew as a radially symmetric disk. In neutrophils this symmetry was not observed, as the
regions of intimate cell-substrate contact decorated the periphery with virtually no
intimate contact at the core. An additional discrepancy is the observation we make in this
thesis that cytochalasin B softening of the cortical shell decreases spreading velocity (Fig.
5.7 B) whereas cytochalasin D treatment in HeLA cells was found to increase spreading 
velocity in the Cuvelier studies.

Beyond single mesenchymal cell spreading, the Cuvelier viscous shell 
surrounding a liquid droplet model has been successfully applied to recapitulate the 
hydrodynamics of multi-cellular aggregate spreading with relevance to the fields of tissue 
morphogenesis and cancer metastasis (89-90). In the context of the data presented in this 
thesis, previous Hammer lab measurements, and recent Waugh lab observations it 
appears the Cuvelier model may be a reasonable course-grained representation of 
macroscopic neutrophil spreading on adhesive fields of FN but not immobilized fields of 
chemoattractant. The “course-grained” qualifier is with respect to the significant points of 
departure previously enumerated. In summary, our observations suggest that physical 
wetting is a useful toy model to help conceptualize the competition of interface energies 
at play during neutrophil spreading, but it alone is not sufficient to reconcile the 
cumulative body of empirical observations.
References


Chapter 3

Ligand Density Elicits a Phenotypic Switch in Human Neutrophils

Preface

The content of this chapter has been adapted from its published version in the journal *Integrative Biology* (2014, Vol. 6:348-356, DOI: 10.1039/C3IB40225H) by permission of The Royal Society of Chemistry. The published manuscript was coauthored by Steven J. Henry, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: SJH designed and executed experiments, analyzed data, and wrote the manuscript; JCC consulted on design of analysis routines, data interpretation, and edited the manuscript. DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose can be retrieved from the published version online.

Abstract

Neutrophils are mediators of innate immunity and motility is critical to their function. We used microcontact printing to investigate the relationship between density of adhesive ligands and the dynamics of neutrophil motility. We show that neutrophils adopt a well-spread morphology without a uropod on moderate densities of adhesion ligand. As density is increased, the morphology switches to a classic amoeboid shape. In
addition to the morphological differences, the dynamics of motility were quantitatively
distinct. Well-spread cells without uropods glide slowly with high persistence while
amoeboid cells made frequent directional changes, migrating quickly with low
persistence. Using an antibody panel against various integrin chains, we show that
adhesion and motility on fibronectin were mediated by MAC-1 (α5β2). The phenotypic
switch could be generalized to other surface ligands, such as bovine serum albumin, to
which the promiscuous MAC-1 also binds. These results suggest that neutrophils are
capable of displaying multiple modes of motility as dictated by their adhesive
environment.

**Introduction**

Leukocytes are important mediators of immunity, and motility is critical to their
function. Neutrophils in particular act as first responders to pathogen challenges (1) as
well as sterile trauma (2) resulting in inflammation. The role of soluble chemoattractants
in stimulating and directing neutrophil motility has long been of interest (1) and has been
explored in various engineered *in vitro* systems (3-4). Recently, attention has shifted to
environment dimensionality in dictating the mode of leukocyte migration (5-7). As the
empirical body of leukocyte observations has grown, it is now appreciated that these cells
can employ an assortment of migratory mechanisms.

On a majority of two-dimensional *in vitro* substrates, neutrophils exhibit an
amoeboid morphology (3-4, 8-12). The distinguishing features of this phenotype are an
elongated cell body with a frontward ruffled-lamellipodium, a midregion that contains the
nucleus, and rearward knob-like uropod (8). Detailed images of this morphology have
been captured with high resolution scanning electron microscopy (SEM) (13-14).
However, there have also been observations of neutrophils assuming a very different, well-spread phenotype without uropods on two dimensional substrates (15-17). In those instances the alternative phenotype was attributed to the underlying stiffness of the material, as neutrophils on softer substrates were shown to re-assume an amoeboid phenotype. Yet, neutrophils also display the amoeboid phenotype on stiff substrates, such as those in the previously cited SEM studies, suggesting substrate stiffness is not a unique controller of cell morphology (18-19). Hypothesizing that another factor was involved in modulating these two phenotypes, our study focused on the role of ligand density.

In this paper, we investigated neutrophil morphology and motility on increasing densities of the extracellular matrix protein fibronectin (FN). We observed that neutrophils exhibited a well-spread, uropod-absent phenotype on sub-saturating, intermediate densities of FN. On high densities of ligand this phenotype was replaced with the amoeboid phenotype. The modes of motility associated with these two morphologies were quantifiably distinct as shown by comparison of their mean squared displacement with time. Finally, we determined that the FN adhesion and motility were mediated by MAC-1 ($\alpha_M\beta_2$). The phenotypic switch could be generalized to other surface ligands, such as bovine serum albumin, to which the promiscuous MAC-1 also binds.

**Materials and Methods**

**Media**

Rinsing buffer was Hanks’ Balanced Salt Solution (Invitrogen, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Invitrogen) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca$^{2+}$ and 2 mM Mg$^{2+}$. 
Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Low-endotoxin bovine serum albumin (BSA) (Sigma) was prepared at 2 % and 0.2 % w/v in PBS without calcium and magnesium (PBS(-)). Labeling of proteins via Alexa Fluor carboxylic acid, succinimidyl ester (Invitrogen) was performed in accordance with the manufacturer’s recommended protocol. Stock N-formylmethionyl-leucyl-phenylalanine (fMLF) (Sigma, Saint Louis, MO) was reconstituted in glacial acetic acid before dilution. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2 % w/v in PBS(-). All solutions were sterile filtered or prepared sterile. Bicinchoninic acid protein assays (Pierce Biotechnology, Rockford, IL) were performed on stock solutions of proteins to measure concentration.

**Substrates**

Poly(dimethylsiloxane) (PDMS) (Sylgard 184 Silicone Elastomer, Dow Corning, Midland, MI) coated coverslips were prepared from number one thickness glass coverslips (Fisher Scientific, Hampton, NH) of 25 mm diameter spun with degassed PDMS (10:1 base:cure by weight). Spinning at 4000 rpm for 1 min, leveling at RT, and baking at 65 °C overnight resulted in a 12.5 ± 0.4 μm layer of PDMS. Bare glass coverslips were cleaned via piranha wash (2:1 by volume H₂SO₄:H₂O₂) and thoroughly rinsed in diH₂O. Coverslips were dried completely in a 90 °C oven. Coverslips, bare and PDMS-coated, were affixed to the bottom of six-well tissue culture plates which had either been hot-punched or laser-cut to generate a 22 mm diameter opening in the bottom of the wells. Coverslip bonding was performed using a continuous bead of Norland Optical Adhesive 68 (Thorlabs, Newton, NJ), cured for 20 min under a long wavelength ultraviolet lamp.
**Protein Deposition and Blocking**

Stamps for printing were prepared from PDMS, mixed at 10:1 base:cure by weight, degassed, and poured over a silicon wafer. The polymer was cured by baking for 2 hr or longer at 90 °C. Trimmed stamps were sonicated in 200 proof ethanol for 10 min, rinsed twice in diH2O and dried in a gentle stream of filtered N2(g). The face of the PDMS stamp previously cast against the silicon wafer was covered with a sessile drop of protein solution. After incubation, stamps were rinsed twice in a submerging quantity (~ 50 mL) of diH2O and dried in a gentle stream of filtered N2(g). For motility studies stamps were 1 cm², inked with 200 μL of protein solution for 2 hrs at RT. For all other experiments, stamps were 0.36 cm², inked with 50 μL of protein solution for 1 hr at RT. After stamp inking and drying, mounted PDMS-coated coverslips were treated for 7 min with ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) to render the surface hydrophilic (20). Stamps were placed in conformal contact with the activated substrate for approximately 30 s.

For physisorption experiments, sterile flexiPERM (Sigma) silicone gaskets were affixed to the substrates to hold an aliquot of protein at a concentration and volume that preserved the number of protein molecules per unit area of exposed surface for comparison with the printed conditions.

Blocking printed or adsorbed surfaces by submersion in 0.2 % w/v solutions of Pluronic F-127 or BSA (0.2 % or 2 %) was performed for 1 hr. Native glass is not amenable to Pluronic blocking until silanized by immersion in 5 % dimethyl dichlorosilane (Sigma) in dichlorobenzene (Sigma) (21). After blocking, each well was rinsed four or five times with 2 mL PBS(-) without dewetting the functionalized surface.
to prevent Pluronic sloughing. If substrates were not used the day of fabrication, they were stored overnight at 4 °C under PBS(-). Prior to cell plating, storage PBS(-) was exchanged for running buffer, without dewetting, and equilibrated to 37 °C at 5 % CO₂ in a cabinet incubator.

**Neutrophil Isolation**

Whole blood was obtained from human donors via venipuncture and collection in heparin vials. Samples were collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24-48 hrs prior to donation. Blood samples were allowed to cool to RT (15-30 min) and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45 min at 500 x g and 21 °C. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 5 min at 350 x g and 21 °C. Red Blood Cells (RBC) were eliminated from the resulting cell pellet via hypotonic lysis. After lysis, vials were centrifuged for 5 min at 350 x g and 21 °C and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at 5 x 10⁵ - 1 x 10⁶ cells/mL on a tube rotisserie at 4 °C until time of plating.

**Cell Motility Experiments**

For a given experimental condition, 7.5 x 10⁴ neutrophils were seeded on a pre-equilibrated substrate under 1.5 mL of running buffer. Visual confirmation was made that cells had a rounded (i.e. not polarized) morphology at time of plating. Substrates and cells were incubated 10 min at 37 °C and 5 % CO₂ to allow settling and gently rinsed
twice with 1 mL of fresh running buffer to remove non-adherent cells. Prior to rinsing, visual observation of the cells confirmed a transition from rounded to a well-spread morphology. Cell density was minimized to prevent cell-cell collisions but sufficiently dense to acquire reasonable sample sizes for statistical testing. Adherent neutrophils at multiple locations on the same substrate were imaged by time-lapse videomicroscopy for 30 min or longer at 30-90 s intervals in a temperature controlled chamber.

Phase-contrast image stacks corresponding to each imaging location of a particular experimental condition were processed via a custom MATLAB (The MathWorks, Natick, MA) script that identified cell boundaries, computed geometric centroids, and connected centroids in consecutive frames to form trajectories. Portions of trajectories were only retained for cells prior to cell-cell collisions and for cells that did not undergo apoptosis. To improve statistical power, multiple locations were imaged per condition. Summing across all field of views (FOVs) acquired we observed a total of 2688 neutrophils, 60 % of which (n = 1606) where tracked and their trajectories utilized in MSD construction and curve fitting. Within this group of observed and tracked cells 75 % (n = 1204) were tracked for the entire duration of the 30 min observation window and used in model-independent analyses. The remaining cells were tracked for only a portion of that observation window as they subsequently underwent cell-cell collisions. Of those cells that were observed but not tracked (n = 1082), 88 % were excluded on the basis of cell-cell contact, residing at the edge of the FOV, or exiting the FOV. The remaining 12 % were excluded on the basis of having an anomalous phenotype (e.g. appearing apoptotic). Cell tracking, mean-squared displacement computation, and error analysis
were based upon the multiple particle tracking method reviewed by Crocker and Hoffman (22). Annotated code used in the analysis of this chapter is reported in Appendix A.

**Integrin Blocking**

The following panel of function-blocking antibodies against various integrin chains was assembled and used at final concentrations of 50 μg/mL: anti-β₁ clone MAb13 (BD Biosciences), anti-β₂ clone L130 (BD Biosciences), anti-α₅ clone ICRF44 (eBioscience), and anti-α₅ clone SAM1 (eBioscience). Isotype controls to IgG1 and IgG2a were purchased from eBioscience. 5 x 10⁵ neutrophils in 200 μL running buffer were incubated for 10 min with antibodies at RT with periodic mixing before exposure to the FN substrate. Substrates and cells were incubated 10 min at 37 °C and 5 % CO₂ and immediately fixed in a solution of 4 % formaldehyde (Fisher) or 10 % neutral buffered formalin (Sigma) for 30 min at RT with periodic mixing. After fixation substrates were rinsed thoroughly with PBS to remove nonadherent cells.

**Results and Discussion**

**Observation of Neutrophil Phenotypes on Two Different Substrates**

A common method of preparing two-dimensional surfaces for motility studies is to adsorb an adhesive ligand onto glass or polystyrene and subsequently wash with a solution of bovine serum albumin (BSA). The BSA wash is intended to mask bare regions of the substrate, unoccupied by protein, and impede non-specific cell-substrate interactions. When we plated human neutrophils on such a surface (fibronectin (FN)-adsorbed and BSA-blocked), the cells assumed an amoeboid phenotype having elongated cell bodies, trailing uropods, and narrow lamellipodia (Fig. 3.1 A). The associated motility was undular, with cells undergoing frequent directional changes (Movie S1). The
Figure 3.1 Two neutrophil morphologies. (A) FN-adsorbed glass, blocked with BSA. Scalebar is 50 μm. Region (i) is enlarged 3X. (B) FN-printed PDMS, blocked with Pluronic. Scalebar is 50 μm. Region (ii) is enlarged 3X. (C) Same preparation as (B) but higher magnification image. Scalebar is 10 μm.
amoeboid phenotype has been reported elsewhere (3-4, 8-14) for neutrophils on various two-dimensional surfaces. By contrast, when we printed a poly(dimethylsiloxane) (PDMS) surface with FN and blocked with Pluronic, we elicited a very different phenotype. In the latter case the neutrophils were highly spread and no trailing uropods were discernible (Fig. 3.1 B). With this phenotype, the cells appeared to glide and were highly persistent in their direction (Movie S2). Our impression is that this latter phenotype was more qualitatively reminiscent of fish-keratocytes (18) than amoeboid cells.

Complementary controls of neutrophils on FN-printed glass and FN-adsorbed PDMS demonstrated that the phenotypic differences depended on the blocking agent, not the method of protein deposition (Fig. 3.2 A-D). Quantitative fluorescence measurements of fluorophore-labeled FN confirmed that total FN loading of glass and PDMS surfaces were comparable (Fig. 3.2 E). When we silanized glass and then blocked surfaces with Pluronic, we found the well-spread, uropod-absent phenotype could be elicited on FN functionalized glass (Fig. 3.3 A). Our interpretation is that when blocking with Pluronic, cell binding was solely due to the underlying FN, and the blocking agent did not contribute to the adhesion (Fig. 3.4 C-E).

We hypothesized that the amoeboid phenotype is a result of adhesion to high densities of surface ligand, and that blocking with BSA served to increase the total ligand content. To test this hypothesis, we used microcontact printing to systematically control the density and type of surface ligand (Fig. 3.5 A). Microcontact printing is a tool to spatially pattern cellular adhesive ligands (20, 23) and to print the tips of polymeric posts for force measurements (24). While microcontact printing has been extensively used to
Figure 3.2 Phenotype does not follow method of protein deposition. To determine if method of protein deposition dictated the two cell phenotypes we compared the following surface preparation strategies: (A) FN-adsorbed glass, BSA blocked (reproduced from Fig. 3.1 A), (B) FN-printed glass, BSA blocked, (C) FN-adsorbed PDMS, Pluronic blocked, and (D) FN-printed PDMS, Pluronic blocked (reproduced from Fig. 3.1 B). Scalebars = 50 μm. Phenotype followed the method of blocking not the method of FN deposition. (E) Mean intensity of FN594 (FN conjugated to Alexa Fluor 594 dye) adsorbed onto glass and printed onto PDMS. Images were acquired under identical settings and the mean pixel intensity computed. For each preparation, the mean pixel intensity of the corresponding negative control was subtracted to produce the “zeroed mean intensity”. Error bars are ± standard error of the mean (n = 2 independent experiments). Amount of deposited FN on both surfaces is comparable. This figure was presented in the supplementary text of the original manuscript.
To determine if substrate type (i.e. glass vs. PDMS) dictated the two cell phenotypes we performed the following controls: (A) FN-printed silanized glass, Pluronic blocked (B) FN-printed PDMS, Pluronic blocked. Surfaces functionalized at 40 % FN surface saturation. Scalebars = 50 µm. This figure was presented in the supplementary text of the original manuscript.
Figure 3.4 Exquisite cell-ligand specificity on Pluronic-blocked substrates. Pluronic F-127 blocking of PDMS substrates allows complete inhibition of non-specific binding in human neutrophils. (A) FN conjugated to Alexa Fluor 647 (FN647) after adsorption to a piranha cleaned coverslip, blocked with 0.2 % BSA in PBS (w/v). The distinct edge shown was achieved by affixing a single-well flexiPERM gasket to the coverslip which was removed prior to blocking and cell plating. (B) DIC image of fixed human neutrophils in same location as (A). Observe that cell adhesion is seen in regions of the substrate not functionalized with FN. (C) FN647 after printing on a PDMS spin-coated coverslip, blocked with 0.2 % Pluronic F-127 in PBS (w/v). (D) DIC image of fixed human neutrophils on microcontact printed substrate in same location as (C). No adhesion outside of the functionalized area is observed. (E) Phase contrast image of fixed cells at a different edge location on same substrate (C-D). All scale bars are 40 μm. Note: non-uniform image acquisition parameters preclude comparison of fluorescent signal intensities between the glass and PDMS conditions (A, C). Surfaces functionalized at 40 % FN surface saturation. This figure was presented in the supplementary text of the original manuscript.
Figure 3.5 Microcontact printing overview and sub-saturating density quantification. (A) PDMS is cast against a silicon wafer to generate a smooth inking face. Stamps are trimmed and a sessile drop of protein solution at known concentration is used to coat the smooth stamping face. Stamps are rinsed and dried gently in a stream of nitrogen. Separately PDMS-spun coverslips are rendered hydrophilic by exposure to UV ozone for 7 min. When the inked stamps are brought into contact with the spun coverslip there is preferential transfer of the protein from the natively hydrophobic stamp to the hydrophilic coverslip. Finally the substrate is passivated by submersion in a nonionic triblock copolymer sold under the tradename Pluronic F-127. Bare regions of the PDMS not occupied by adhesive ligand are rendered stealth to neutrophils by Pluronic coating. (B) Quantitative fluorescence microscopy to determine the relative density of protein on printed substrates by titrating inking concentration. The saturating condition was considered to be 100 μg/mL. Error bars are standard error of the mean (n = 7-9 independent experiments). This figure was presented in the supplementary text of the original manuscript.
study the behavior of mesenchymal (20, 23-27) cells, it has only recently been applied in studies of hematopoietic-derived cells (28-31). In our study, microcontact printing was used to immobilize different densities of FN on PDMS. By titrating the inking concentration of the protein solution used to prepare the stamps, we could reproducibly achieve sub-saturating densities of deposited FN (Fig. 3.5 B). After fabricating a series of PDMS surfaces with systematically varied densities of FN, all blocked with Pluronic F-127, we scored the resulting neutrophil phenotypes observed (Fig. 3.6).

On surfaces printed with little or no FN and blocked with Pluronic, cells failed to polarize or spread and remained spherical, presenting as bright white circles under phase contrast imaging (Fig. 3.6 i). On intermediate densities of printed-FN, blocked with Pluronic, the well-spread, uropod-absent phenotype was observed (Fig. 3.6 ii). Frequency of the keratocyte-like phenotype peaked at 40 % surface saturation. As density of FN increased, the well-spread phenotype was observed less frequently. Once surface density reached 83 % saturation the amoeboid phenotype was predominate (Fig. 3.6 iv).

Others have observed this well-spread, uropod-absent phenotype in neutrophils on FN-conjugated polyacrylamide gels (15-17). In those instances the morphology was attributed to the underlying stiffness of the material, as neutrophils on softer gels were more amoeboid. Indeed, our relatively thick PDMS layers (~ 12 μm) and the use of a 10:1 formulation (base:cure, w/w) means the substrates were quite stiff, with Young’s moduli on the order of megapascals (32-33). However, here we demonstrated that the well-spread phenotype on stiff surfaces is only inducible for sub-saturating densities of ligand. This observation contributes to the growing empirical body of evidence showing neutrophils and other leukocytes can adopt a variety of motile mechanisms to achieve
Figure 3.6 Neutrophil phenotype on increasing densities of fibronectin (FN). Adherent neutrophils as percentage of total plated cells per sub-saturating densities of FN. Representative images from a single experiment on (i) 0.7 % (ii) 11 % (iii) 66 % and (iv) 100 % saturated FN substrates. Scalebars are 50 μm. Error bars are ± standard error of the mean. Substrate density was measured via quantitative fluorescence microscopy (Fig. 3.5 B). All substrates were FN-printed and Pluronic-blocked PDMS.
translocation and helps reconcile the occurrence of both phenotypes elsewhere in the literature of neutrophil motility on stiff substrates.

Ziebert and Aranson have constructed a biophysical model of cell motility that demonstrates phenotypic transitions in the mode of migration as a function of underlying substrate stiffness and surface adhesivity (19). On stiff substrates their model predicts a transition from stick-slip to gliding motion as surface ligand density is increased. While we have not observed stick-slip motion at low adhesivity we have found an intermediate ligand density window in which neutrophils display a highly persistent gliding phenotype. It will be interesting to see if the incorporation of intracellular viscoelasticity into their future models can recapitulate our transition from gliding motion to amoeboid motion at saturating conditions of adhesive ligand. A transition from gliding to more erratic motion has also been reported of fish keratocytes on stiff substrates as surface adhesivity increases (18).

Our study of how neutrophil phenotype depends on adhesion draws an interesting qualitative comparison with recent work on the capacity of physical confinement to dictate migratory cell phenotype. Migratory cells in physically confined channels or on narrow one-dimensional tracks of ligand have been shown to lose characteristics of conventional two-dimensional migration (7). Hung and co-workers have also found that the mechanism of propulsion differs as a function of substrate dimensionality (34). In the future, immunocytochemical staining and small molecule inhibitor studies of our amoeboid versus keratocyte-like morphologies may reveal similar discrepancies driven by ligand density.
Quantifying Motility of Amoeboid and Keratocyte-Like Phenotypes

The dynamics of amoeboid and keratocyte-like motility were distinct, as revealed by comparing their mean squared displacements (MSD) as a function of time (Fig. 3.7). On log-log axes, the slopes of MSD vs. time for the two populations were different. Neutrophils undergoing amoeboid migration accumulated squared displacement diffusively (slope ~ 1) while neutrophils undergoing keratocyte-like migration accumulated squared displacement superdiffusively (slope > 1). Fitting the curves for MSD vs. time with the persistent random walk model of cell kinesis (35-36) allowed us to quantify neutrophil motility in terms of the best-fit parameters speed (S) and persistence (P). Doing so confirmed our qualitative assessment that amoeboid motility was faster and less persistent (S_{amoeboid} = 6 \mu m/min, P_{amoeboid} = 0.5 min) than keratocyte-like motility (S_{keratocyte-like} = 3 \mu m/min, P_{keratocyte-like} = 15 min). Comparing the cytoskeletal architecture of these two dramatically different phenotypes remains to be done. It will be interesting to learn how stress fibers are organized in the keratocyte-like cell, compared to the amoeboid cell.

To this point neutrophils were induced to adhere and be motile on FN substrates without prior or concurrent stimulation by soluble chemoattractant. Therefore, the resulting motility was haptokinetic, driven by FN stimulation at the cell-substrate interface. A control study quantifying selectin-expression (37) via flow cytometry confirmed neutrophils were not primed for integrin-based adhesion to FN surfaces by virtue of isolation or storage stresses (Fig. 3.8).
Figure 3.7 Mean squared displacements (MSDs) of two motility modes. Time and ensemble averaged MSDs of neutrophils undergoing amoeboid motility or keratocyte-like motility. Amoeboid cells acquire displacement diffusively, slope ~ 1. Keratocyte-like cells acquire displacement superdiffusively, slope > 1. Dotted line is fit of empirical data with persistent random walk (PRW) model of cell motility. Error bars are ± standard error of the mean.
Flow Cytometry to Assess Activation State

The prose in this section was presented in the supplementary text of the original manuscript. Because neutrophils were robustly haptokinetic on FN alone without the addition of chemoattractant, we verified that cells were not primed for binding to the adhesion ligand as a result of stresses experienced prior to FN exposure. We used L-selectin as the marker of cell activation state. Kishimoto and coworkers demonstrated that L-selectin is a sensitive marker of a neutrophil’s transition from quiescence to a phenotype primed for integrin-mediated binding (37), a transition denoted by rapid L-selectin shedding.

Staining of all treatment conditions was for 45 min on ice in the dark immediately followed by fixation in 2% formaldehyde for 20 min. After fixation, vials were spun to pellet cells (350 x g, 5 min, 4 ºC) and resuspended in HBSS without calcium or magnesium. This rinsing sequence was repeated three times. After the final resuspension, cells were stored overnight on ice in the dark until flow cytometry measurements the following day. Antibodies were mouse-anti-human CD62L-PE-Cy5 (eBioscience) and mouse IgG1k-PE-Cy5 isotype control (eBioscience).

Immediately after isolation, neutrophils were stained for L-selectin (Fig. 3.8 A). Positive (i.e. activated) controls were generated by exposing isolated neutrophils to the chemoattractants TNFα and fMLF immediately following isolation (Fig. 3.8 B-C). A decrease in L-selectin expression by cells exposed to chemoattractant, relative to the post-isolation control, demonstrated the isolated neutrophils had the capacity to be activated. To mimic the conditions cells would experience prior to plating on a FN-printed PDMS substrate, a separate aliquot of cells was subjected to storage, buffer
Figure 3.8 Quantification of L-selectin expression levels via flow cytometry. Expression levels were assayed under the following conditions: (A) immediately after isolation from whole blood, (B) immediately after isolation including 100 U/mL TNFα or (C) 100 nM fMLF as positive activation controls, and (D) prior to FN exposure mimicking the storage, buffer exchange, and re-warming steps experienced by plated cells. Scalebar is 400 counts. Mean relative median fluorescence intensity (Relative M.F.I) was computed for each experimental condition (E). Errorbars are standard error of the mean (n = 2 donors). Asterisk denotes significant difference and n.s. denotes a difference not statistically significant as computed by post-hoc SNK Multiple Comparisons Method (p < 0.05). fMLF was excluded from significance testing (n.t.). This figure was presented in the supplementary text of the original manuscript.
exchange, and re-warming consistent with the plating protocol used in our motility studies. Flow cytometry on these pre-FN mimics (Fig. 3.8 D) showed a slight increase in L-selectin expression relative to the post-isolation control.

To quantify the extent of these shifts, the relative median fluorescence intensity (Relative M.F.I. = (M.F.I\_Sample - M.F.I\_Isotype)/M.F.I\_Isotype) of each condition was computed (Fig. 3.8 E). A statistically significant decrease in L-selectin expression as a function of TNF\(\alpha\) was observed relative to the post-isolation control and pre-FN mimic. No statistically significant difference was found between the post-isolation control and pre-FN condition. Thus while the isolated neutrophils were capable of activation, they were not primed for integrin-mediated binding by virtue of isolation or storage stresses prior to FN exposure. This finding, coupled with high cell-FN specificity on Pluronic blocked PDMS substrates, leads us to attribute the post-plating adhesion and haptokinesis solely to the deposited FN.

**Effect of Chemoattractant on Keratocyte-Like Motility**

We explored the capacity of the potent neutrophil chemoattractant formyl-Met-Leu-Phe (fMLF) (38) to modulate the motility of neutrophils undergoing keratocyte-like migration. On 44 % saturated FN surfaces, the addition of 10 nM fMLF to haptokinetic neutrophils had the effect of increasing the total dispersion of the cell system (Fig. 3.9 A-B). To quantify the extent of motility in a model-independent fashion we extracted the maximum displacements for cells tracked over 30 min. Cell trajectories shorter than 30 min were excluded in this analysis to avoid inadvertently biasing the data. The mean of the maximum displacements (\(<\text{max}(|\Delta r|)>\)) was computed for each combination of FN adhesiveness and fMLF concentration (Fig. 3.9 C). Introducing fMLF, after onset of FN-
induced haptokinesis, potentiated motility in a dosage-dependent manner at an intermediate ligand density of 44% saturation. However, at a higher surface saturation of 73%, fMLF was no longer capable of increasing the basal motility induced by FN stimulation. The number of independent observations for each condition and a comprehensive description of mean maximum displacement data are reported in Fig. 3.10.

Computation of the MSD provides dynamic information on the dispersion of cells and allows the incorporation of cell trajectories shorter than the total experimental acquisition time. Time and ensemble-averaged MSDs for each independent observation were computed from all available cell trajectories through 30 min. The MSDs corresponding to Fig. 3.9 A-B data are reported in Fig. 3.9 D. In general, on log-log axes, the slope of the MSD curves are relatively constant and greater than unity. This denotes superdiffusive motility in which cells accumulate squared displacement faster than expected by pure diffusion. Considering the best-fit parameters speed and persistence, systematic variation in the dose of fMLF alters cell speed at intermediate density FN (Fig. 3.9 E), but not the persistence time for any of the FN-fMLF conditions tested (Fig. 3.9 F). All MSDs-vs.-time contributing to construction of Fig. 3.9 D are compiled in Fig. 3.11 along with complete results of multiple comparisons testing on mean speed data.

In both analyses the capacity of chemoattractant to augment haptokinetic motility in the keratocyte-like phenotype was found to be a function of the underlying adhesiveness. This emphasizes the importance of considering the role of substrate adhesiveness in controlling the cell response to the milieu of soluble chemoattractants and cytokines known to orchestrate directional motility during inflammation.
Figure 3.9 Quantification of neutrophil haptokinesis and chemokinesis of keratocyte-like phenotype. Human neutrophil trajectories through 30 min of motility on 44 % saturated FN surface in (A) the absence of fMLF and (B) the presence of 10 nM fMLF. Scalebar is 50 μm. Solid red circle is the mean maximum displacement (\(\langle\max(|\Delta r|)\rangle\)) of 30 min neutrophil trajectories for (A) \(\langle\max(|\Delta r|)\rangle\sim 24 \mu\text{m}\) and (B) \(\langle\max(|\Delta r|)\rangle\sim 51 \mu\text{m}\). (C) Mean of the set of mean maximum displacements for all independent observations of a particular FN density and fMLF combination tested (\(\langle\langle\max(|\Delta r|)\rangle\rangle\)). (D) MSD(\(\tau\)) corresponding to a single donor’s neutrophils migrating on 44 % saturated FN surface in the presence or absence of fMLF. Dotted red line is fit of persistent random walk model (PRW) to empirical data. Model fit parameters (E) speed and (F) persistence. Error bars are ± standard error of the mean. Asterisk denotes significant difference relative to No fMLF condition as computed by post-hoc SNK Multiple Comparisons Method (\(p < 0.05\)).
Figure 3.10 Sample sizes per condition and complete results of model-independent significance testing. (A) Table summarizing sample sizes for each experimental condition (FN/fMLF combination). N_{indep} (column 3) is the number of independent observations where an independent observation is a unique donor/donation combination. N_{cells,tot} (column 4) is the number of total cell trajectories acquired across all independent observations. \langle n_{cells}\rangle (column 5) is the average number of cells contributed by each independent observation without weighting. Because each independent observation of a condition contributed a different number of cells, weighting is required. Weighting mean values by the number of cells used in the computation of the mean results in an effective number of independent observations on the mean given by N_{indep,eff} (column 7) and a corresponding effective average number of cells per independent observation \langle n_{cells,eff}\rangle (column 6). These later two values can be thought of as a hypothetical number of independent observations (N_{indep,eff}) of equal statistical power, each experiment contributing the same number of cells (\langle n_{cells,eff}\rangle). (B) Complete results of significance testing corresponding to the mean maximum displacement metric of Fig. 3.9 C. A star denotes a significance difference as computed by post-hoc SNK Multiple Comparisons Method (p < 0.05). This figure was presented in the supplementary text of the original manuscript.
Figure 3.11 MSDs of all independent observations of FN/fMLF experimental conditions tested. (A) For a given elapsed time interval (τ), MSD(τ) is the variance of the population of displacements within and across all cells (i.e. time and ensemble averaged). τ_{min} is the experimental frame rate and τ_{max} is 30 min. This study utilized six donors (closed symbols), four of which donated on a separate experimental day (open symbols). Variability within a given donor on different experimental days for the same experimental condition led us to treat each donor/donation as an independent observation. Plots are organized by adhesiveness (columns) and concentration of fMLF (rows). All plots are scaled identically. Error bars are ± standard error of the variance (i.e. of the MSD(τ)). Eye guides of slope (“m”) 1 and 2 are provided for reference. (B) Complete results of significance testing corresponding to the speed parameter “S” from the persistent random walk fit to the empirical MSDs. A star denotes a significance difference as computed by post-hoc SNK Multiple Comparisons Method (p < 0.05). No statistically significant differences were found among persistence values of Fig. 3.9 E. This figure was presented in the supplementary text of the original manuscript.
Identifying Integrin Chains Responsible for Adhesion

To identify the integrin chains responsible for neutrophil binding to FN, function-blocking antibodies with previously demonstrated efficacy in leukocytes were employed (39-40). Functional blocking of \( \beta_2 \) integrins (Fig. 3.12 D) resulted in a substantial decrease in cell adhesion on FN relative to the positive control without antibody present (Fig. 3.12 A). Targeting the \( \alpha_M \) integrin, which coordinates with \( \beta_2 \) integrin to form the MAC-1 heterodimer, was also found to disrupt cell binding on FN significantly (Fig. 3.12 F). In neither case did blocking \( \beta_1 \) (Fig. 3.12 C) nor \( \alpha_5 \) (Fig. 3.12 E) integrin chains disrupt binding. These results led us to attribute the observed FN-induced adhesion and subsequent haptokinesis to the \( \beta_2 \) and \( \alpha_M \) integrin subunits, or the MAC-1 receptor.

In neutrophils there is known cross talk between \( \beta_1 \) and \( \beta_2 \) integrins when ligating extracellular matrix proteins such as FN (39, 41). Our finding that neutrophils utilize MAC-1 (\( \alpha_M \beta_2 \)) on FN is consistent with other empirical observations. In particular van den Berg and coworkers demonstrated that stimulation of \( \beta_1 \) integrins yields \( \beta_2 \)-mediated adhesion in neutrophils on FN that can be mitigated by function-blocking antibodies against MAC-1 (39). Our blocking study is a probe on the long time-limit (i.e. minute length scale) adhesion of neutrophils to FN. Lishko and co-workers demonstrated that a balance of MAC-1 and VLA-5 (\( \alpha_5 \beta_1 \)) is required for neutrophil translocation on FN attributing MAC-1 to adhesion and VLA-5 to migration (41). Our work reveals that the adhesive contribution of MAC-1 is the dominant ligated integrin and may explain the reduced speed of the keratocyte-like phenotype.
**Figure 3.12 Integrin blocking on FN.** Integrin blocking of neutrophils pre-incubated with antibodies against various integrin chains before exposure to 44% saturated FN surface. *(A)* Positive binding control, no antibodies. *(B)* Negative binding control, no FN, just Pluronic blocking. *(C)* anti-β₁ clone MAb13, *(D)* anti-β₂ clone L130, *(E)* anti-α₅ clone SAM1, and *(F)* anti-α₅ clone ICRF44. Scalebars are 50 μm. *(G)* Mean ratio of adherent cells to isotype control. Error bars are ± standard error of the mean (n = 3-4). Asterisk denotes significant difference relative to isotype control as computed by post-hoc Dunnet’s Method (p < 0.05).
MAC-1 also binds to members of the Ig superfamily (42-43), such as ICAM-1, which illustrates the promiscuity of this integrin. We hypothesized that the emergence of the amoeboid phenotype on BSA-blocked surfaces of intermediate density FN was due to simultaneous binding of MAC-1 to BSA and FN. Indeed, we were able to recapitulate the keratocyte-like phenotype on intermediate densities of BSA alone (Fig. 3.13A). The percentage of plated neutrophils exhibiting keratocyte-like phenotype on fields of BSA at sub-saturating density was 63 % (n = 3, SE = 22 %). Furthermore, at saturating densities of BSA alone, neutrophils again switched to the amoeboid phenotype (Fig. 3.13B). The percentage of plated neutrophils exhibiting amoeboid phenotype on fields of BSA at saturating density was 73 % (n = 1, SD = 4 %). When we repeated the function-blocking antibody study on neutrophils exposed to intermediate-density BSA substrates, we again found that MAC-1 was mediating adhesion (Fig. 3.13C).

The finding that neutrophils were employing the promiscuous integrin MAC-1 to mediate adhesion to our experimental surfaces reinforces the necessity of choosing an appropriate blocking reagent against non-specific cell adhesion. BSA, which is often used to block surfaces, actually functions as an adhesive ligand. Coating surfaces with Pluronic is the only method we have found to reliably eliminate all non-specific background adhesion in our in vitro motility assays. This type of exquisite discrimination of the roles of different ligands is only possible with improved surface techniques, such as microcontact printing (20).

Aside from the obvious conclusion that care must be taken to block non-specific binding with appropriately neutral ligands, future work will address how the organization and density of adhesion ligands leads to the morphology of cell response. Now, we can
Figure 3.13 Neutrophil adhesion to BSA. (A) Keratocyte-like phenotype of neutrophils on intermediate density of BSA. (B) Amoeboid phenotype returns on saturating density of BSA. (C) Recapitulation of antibody blocking study of neutrophils on intermediate density BSA surfaces. Mean ratio of adherent cells to isotype control. Error bars are ± standard error of the mean (n = 2-3). Asterisk denotes significant difference relative to isotype control as computed by post-hoc Dunnet’s Method (p < 0.05).
speculate that a high density of adhesion ligands over a large spatial domain promotes uropod formation. If this is the case, distribution of ligands into patches would prevent uropod formation, even if the density in the patches were locally high.

Conclusions

Our work has demonstrated that neutrophils are capable of a phenotypic switch in morphology and associated motility as dictated by adhesion ligand density. The nature of the density sensing remains to be addressed in determining whether neutrophils are sensitive to these changes at the receptor length scale or across their total cell-substrate contact area. We anticipate microcontact printing will be a useful platform in addressing this question. By quantifying the motility associated with the amoeboid and keratocyte-like phenotypes we found the modes of migration to be distinct. The biophysical mechanism that underpins these differences is unclear. We suspect visualizing cytoskeletal architectures will improve our mechanistic insight. Lastly, our finding that the integrin heterodimer MAC-1 was being employed to mediate adhesion to our experimental surfaces reinforces the importance of avoiding BSA as an agent to block non-specific binding in neutrophils.

Acknowledgements

We are grateful to Eric Johnston for technical assistance in the laboratory and Professor Chistopher S. Chen, Michael T. Yang, PhD and Ravi A. Desai, PhD for their time and expertise in teaching us microcontact printing. Funding for this work was provided by a National Science Foundation Graduate Research Fellowship to SJH and a grant from the National Institutes of Health (HL18208) to DAH.


Chapter 4

Human Neutrophil Adhesion Density Sensing at the Whole Cell Length Scale

Preface

The content of this chapter has been adapted from the version in preparation for submission to *Annals of Biomedical Engineering*. The manuscript was coauthored by Steven J. Henry, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: SJH designed and executed experiments, analyzed data, and wrote the manuscript; JCC consulted on design of analysis routines, data interpretation, and edited the manuscript. DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose will be retrievable from the published version online.

Abstract

Neutrophils, highly motile immune cells, are capable of a phenotypic switch with respect to their shape and mode of migration as driven by adhesive ligand density. In this study, we engineered planar adhesive environments to elucidate the length scale of neutrophil adhesion density sensing. The engineered surfaces were hybrid in that they presented neutrophils with high and low density cues simultaneously. By controlling island geometry we achieved arrays in which the local (on-island) adhesion density was
high but the global (multi-island) adhesion density over the entire cell-substrate interface was low. These hybrid surfaces were achieved by the stamp-off method of microcontact printing. Neutrophils in contact with these island arrays assumed a well-spread and directionally-persistent motile phenotype in contrast to their classic amoeboid morphology on continuous fields of high adhesion density. By virtue of our rationally designed substrates, we were able to conclude that neutrophils were sensing density at the whole cell length scale, integrating the stimulation received across their entire contact interface and mounting a whole cell response on the timescale of seconds. This work demonstrates the capacity of adhesive microenvironments to direct neutrophil motile phenotype which has broader implications in physiologic processes such as cancer metastasis.

**Introduction**

Neutrophils are a type of white blood cell (leukocyte) that responds to tissue trauma and infection on the timescale of seconds and minutes. These cells are equipped with a variety of terminal functions including phagocytosis, cytokine secretion, and nuclear-extracellular-trap setting (1). A prerequisite to the execution of any of these terminal functions is the cell’s arrival at the locus of trauma (2) or infection (3) via vascular rolling, extravasation, and extravascular migration (4). In addition to soluble chemical cues that direct immune cell response and function, cells encounter numerous physical cues (e.g. stiffness, dimensionality, adhesivity, and topology) that are strong determinants of shape, force generation, and gene expression (5-6). Leukocyte response to physical cues such as substrate rigidity (7-9), confinement (10), and adhesion density (11) have been areas of on-going investigation.
Previously, others have demonstrated on planar (2D, unconfined) substrates that neutrophil contact area and force generation were stiffness-dependent (7-9). However we showed that stiffness alone was not a unique controller of adherent neutrophil shape or motility as varying the adhesivity of the surface also dictated cell phenotype on equally stiff substrates (11). In that work, using the method of microcontact printing, we quantified neutrophil shape and motility on sub-saturating densities of the extracellular matrix protein fibronectin (FN). On highly adhesive surfaces neutrophils assumed a classic amoeboid phenotype characterized by an elongated cell body, knob-like trailing uropod, and a narrow, ruffled leading edge lamellipodium (12-13). The observed motility was fast and consisted of frequent directional changes. However on low and moderately adhesive surfaces neutrophils assumed a phenotype reminiscent of fish keratocytes (14-15), characterized by the absence of a trailing uropod and a highly spread fan-like lamellipodium. The observed motility was a slow, but directionally persistent gliding motion. The capacity of adhesion density to alter the phenotypic mode of neutrophil migration drew analogy with adhesion sensitivity in fish keratocytes observed by Barnhart and coworkers (15) and computational predictions of the effect of adhesion on stiff substrates in migratory cells made by Ziebert and Aranson (16).

An open question from our prior work was the length scale over which the neutrophil adhesion density sensing was occurring. Were neutrophils responding to local adhesive cues on the length scale of receptor clusters or integrating adhesive stimulation across their entire contact interface? To address this question we employed the stamp-off variation of microcontact printing (17), to generate a hybrid surface in which high and low density adhesive cues were presented to neutrophils simultaneously. This
engineering approach of spatially organizing a cell’s adhesive environment has been widely used to probe integrin clustering (18-19), effect of cell shape on viability (20) and focal adhesion architecture (21), and the role of extracellular matrix distribution on cell spreading (22) in the context of mesenchymal cells. Here we report the effect of adhesive ligand density distribution on neutrophils, a distinct cell of hematopoietic origin.

**Materials and Methods**

**Media and Reagents**

Rinsing buffer was Hanks’ Balanced Salt Solution (Life Technologies, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Life Technologies) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca$^{2+}$ and 2 mM Mg$^{2+}$. Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Labeling of FN via Alexa Fluor carboxylic acid, succinimidyl ester (Life Technologies) was performed in accordance with the manufacturer’s recommended protocol. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2% w/v in PBS without calcium and magnesium (“PBS(-)”). All solutions were sterile filtered or prepared sterile. The bicinchoninic acid protein assay (Pierce Biotechnology, Rockford, IL) was performed on stock FN solutions to measure concentration. Poly(dimethylsiloxane) (PDMS) was Sylgard 184 Silicone Elastomer from Dow Corning (Midland, MI) prepared per the specified weight ratio of base:cure agents, mixed vigorously, and degassed until optically clear.
**Substrate Production**

25:1 base:cure (w/w) PDMS stamps were cast against a silicon wafer to produce an extremely smooth surface. Stamps were trimmed to approximately 25 mm$^2$, sonicated in 200 proof ethanol for 10 min, rinsed twice in diH$_2$O and dried in a gentle stream of filtered N$_2$(g). The surface of the PDMS stamp, previously cast against the silicon wafer, was incubated with a 50 µL aliquot of fluorescently labeled fibronectin (FN-AlexaFluor594) at a known concentration in PBS(-) for 1 hr at room temperature. After incubation stamps were rinsed twice in a submerging quantity (~ 50 mL) of diH$_2$O and dried in a gentle stream of filtered N$_2$(g) (Fig. 4.1 A).

For experiments with islands, these inked stamps were subject to stamp-off. An array of holes was generated by casting 10:1 base:cure (w/w) PDMS reliefs of positive silicon microfabricated-Post-Array-Detectors. Silicon masters were manufactured in Professor Christopher S. Chen’s laboratory in the manner detailed by Yang et al.(23) Cast PDMS hole arrays were rendered hydrophilic by 7 min treatment in ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) (17). The hydrophilic array was inverted, set atop the inked stamp, and peeled to produce two complimentary surfaces (Fig. 4.1 B).

PDMS coated coverslips were prepared from number one thickness glass coverslips (Fisher Scientific, Hampton, NH) of 25 mm diameter spun with degassed PDMS (10:1 base:cure (w/w)). Bare glass coverslips were cleaned via oxygen plasma etching and then spun at 4000 rpm for 1 min under PDMS. Leveling at RT, and baking at 65 °C overnight resulted in an approximately 10 µm thick layer of PDMS. Cured coverslips were affixed to the bottom of six-well tissue culture plates which had either been hot-punched or laser-cut to generate a 22 mm diameter opening in the bottom of the
wells. Coverslips were bonded using Norland Optical Adhesive 68 (Thorlabs, Newton, NJ). Mounted coverslips were rendered hydrophilic by 7 min treatment in ultraviolet ozone and then printed with a continuous field of protein or the stamped-off array of islands (Fig. 4.1 C). Substrates were blocked against non-specific binding by submerging in 0.2 % w/v F-127 in PBS(-) and incubating 30 min at RT (Fig. 4.1 D). After blocking, F-127 was exchanged for PBS(-) by repeated and gentle rinsing with running buffer. Chambers were pre-warmed to 37 °C in a cabinet incubator before cell plating and imaging.

**Neutrophil Isolation**

Whole blood was obtained from human donors *via* venipuncture. Samples were collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24 hrs prior to donation. Blood samples were allowed to cool to RT for 15 min and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45-60 min at 550-650 x g and 21 ºC. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 10 min at 250 x g and 21 ºC. Red Blood Cells (RBC) were eliminated from the resulting cell pellet *via* hypotonic lysis. After lysis, vials were centrifuged for 10 min at 250 x g and 21 ºC and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at 10^6 cells/mL on a tube rotisserie at 4 ºC until time of plating to maintain cells in suspension.
**Quantitative Fluorescence Microscopy**

A non-flickering mercury bulb within the manufacturer-specified bulb lifetime was used to illuminate samples. Adjustments to bulb alignment and focus were made to achieve a uniform field of illumination. Within a given experimental series all acquisition parameters were held constant and images acquired identically. For each condition (i.e. feature and ligand density combination) multiple fields of view (FOV) were acquired across the entire printed domain as well as appropriate measurements of background fluorescence intensity. To mitigate the effects of photobleaching, focus was set in a region adjacent to the FOV actually imaged. To compare results across independent experiments, mean fluorescent intensities were normalized by the mean intensity of the saturating condition within that series after background subtraction.

**Cell Motility Experiments and Data Analysis**

Neutrophils were seeded into pre-warmed culture dishes and allowed to gravity sediment onto the printed arrays. Multiple position time-lapse videomicroscopy was performed to track cell shape and position for at least 30 min with images acquired every 15-60 sec. Motility quantification was performed using a custom suite of MATLAB (The MathWorks, Natick, MA) scripts which identified cell boundaries, computed geometric centroids, and connected centroids in consecutive frames to form trajectories. Cell tracking, mean squared displacement computation, and error analysis were based upon the multiple particle tracking method reviewed by Crocker and Hoffman (24).
Results

*Engineering Substrates to Present Neutrophils with Two Adhesive Length Scales*

By using the stamp-off method of microcontact printing (Fig. 4.1 A-D) we generated hexagonal arrays of submicron diameter islands of the extracellular matrix protein fibronectin (Fig. 4.1 I and J). A spread neutrophil was in contact with many of these islands (~ 100 islands/cell) at once as they were small and tightly spaced relative to the total size of the cell (Fig. 4.5 E). To aid visualization of the islands, contrast was enhanced in fluorescence images of Figure 4.1, however the unenhanced images are provided in Figure 4.2. Printed islands were hexagonally arranged with a measured mean diameter of 0.904 ± 0.010 μm and pitch of 1.932 ± 0.002 μm (Fig. 4.3). Quantities are means ± standard error of the mean for five independent substrates with an average of 1296 printed islands measured per substrate. Individual islands had a surface area of 0.64 μm² whereas the macroscopic surface area (i.e. a region containing many islands) represented a reduced contact area of 20 % compared to a continuous field.

The principle aim of this study was to generate a hybrid surface in which neutrophils were presented simultaneously with two length scales of adhesive stimulation. This required controlling array geometry and protein loading density such that the final printed surface had locally (i.e. on islands) high protein content but globally (i.e. the area equivalent to a cell body) low average protein content. Inking concentration was a more facile variable to manipulate at the wet bench as compared to island geometry. Therefore, we fixed array geometry and performed a sweep of inking concentrations to identify high and low conditions such that stamp-off of a high content
Figure 4.1 Stamp-off method of microcontact printing to generate island arrays. Substrate preparation consisted of: (A) stamp inking, (B) stamp-off, (C) stamp-on, and (D) Pluronic F-127 blocking. (E) Brightfield image of hole array used in stamp-off procedure. (F) Fluorescence image of protein on hole array after stamp-off. (G) Higher magnification image of hole array after stamp-off. (H) Brightfield image of PDMS coverslip after stamp-on. (I) Fluorescence image of protein after stamp-on. (J) Higher magnification image of island array after stamp-on. Fluorescence images were contrast-enhanced to assist island visualization. Unenhanced images are reported in Figure 4.2. Scalebars = 200 μm for E, F, H, and I. Scalebars = 10 μm for G and J.
Figure 4.2 Stamp-off method of microcontact printing to generate island arrays: no contrast enhancement. (A) Brightfield image of hole array used in stamp-off procedure. (B) Fluorescence image of protein on hole array after stamp-off. (C) and (D) are higher magnification images of A and B features respectively. (E) Brightfield image of PDMS coverslip after stamp-on procedure. (F) Fluorescence image of protein on flat PDMS coverslip after stamp-on. (G) and (H) are higher magnification images of E and F features respectively. For a given magnification, fluorescence images were captured identically. Scalebars = 200 µm for A, B, E, and F. Scalebars = 10 µm for C, D, G, and H.
Figure 4.3 Measurement of mean printed island diameter and pitch. Scanning electron micrograph (A) plan view and (B) cross-section view of PDMS hole arrays used in stamp-off procedure. (C) Fluorescence image of printed islands and its (D) corresponding binary bitmap after image processing to measure island positions. (E) histogram of island diameters in D. (F) histogram of nearest neighbor distances (i.e. pitch) in D. The values of diameter and pitch quoted in the main text are the mean and standard error of the mean from five samples, analyzed in the same manner as above.
continuous field produced islands with a global area average equivalent to a low content continuous field.

For our experimental geometry and inking process, we identified that FN-saturated stamps (inking concentration in excess of 30 μg/mL) could be stamped-off to produce islands resulting in a global density equivalent to that of a uniformly inked 2 μg/mL stamp (Fig. 4.4 A, shaded region). It is important to note that this set of conditions straddled the adhesive threshold (44 % relative to saturation, denoted by the dotted line in Fig. 4.4 A) we previously identified (11) below which the keratocyte-like phenotype occurs, but above which the amoeboid phenotype occurs.

Quantitative fluorescence microscopy was used to measure the on-island (Fig. 4.4 C), area average (Fig. 4.4 D), and between-island (Fig. 4.4 E) densities of printed protein. We found that printed islands had on-island densities (Fig. 4.4 B iii) approaching that of continuous fields of high protein content (Fig. 4.4 B i) while the protein content between islands was nearly zero (Fig. 4.4 B v). The area average protein content of the islands (Fig. 4.4 B iv) was equivalent to that of 2 μg/mL continuous fields of protein (Fig. 4.4 B ii). For each condition 10-12 fields of view were acquired from each of four independent substrates.

**Neutrophils Integrate Adhesive Stimulation Across Entire Contact Interface**

We previously published observations of a phenotypic switch in neutrophil shape and motility governed by adhesive density (11). Here, consistent with those findings, we observed the amoeboid phenotype on high density (50 μg/mL) continuous fields of FN (Fig. 4.5 A, D) and the keratocyte-like phenotype on low density (2 μg/mL) continuous fields of FN (Fig. 4.5 B). Both high and low density continuous fields represented
Figure 4.4 Engineering islands with two adhesive length scales. (A) Quantitative fluorescence microscopy of printed continuous fields (red squares) and stamped-off islands (blue circles). Gray shaded region represents domain where stamp-off of high density continuous fields produces islands with an area average equivalent to a low density continuous field. Dotted line denotes adhesive threshold delineating neutrophil phenotypes. Errorbars are ± standard deviation from 2-4 replicates for each concentration within a single experiment. (B) Quantitative fluorescence microscopy of representative substrates used in adhesion and motility assays: (i) high density FN (50 μg/mL) continuous fields, (ii) low density (2 μg/mL) continuous fields. (iii) on islands (see ROI of C), (iv) area average of protein density across islands (see ROI of D), and (v) residual protein density between islands (see ROI of E). Scalebars = 2 μm. Errorbars are ± standard error of the mean from four independent substrate preparations.
surfaces with uniform adhesive stimulation across the cell-substrate interface. By contrast, our hybrid island surfaces presented the cells with two effective adhesive length scales simultaneously. On the scale of single islands the density was high, comparable to that of high protein-content continuous fields. On the scale of multiple islands the density was low, comparable to that of low protein-content continuous fields. We hypothesized that if a neutrophil was sensitive to local density it would assume the amoeboid phenotype whereas if it was sensitive to global density, across its contact interface, it would assume the keratocyte-like phenotype. Consistent with the later hypothesis, neutrophils assumed the keratocyte-like phenotype on engineered islands where the total protein content averaged over the cell contact interface was low (Fig. 4.5 C, E). Both phenotypes were observable in the same field of view when continuous fields were adjacent to discrete islands (Fig. 4.5 G). In Figure 4.6 the phenotype scores for the three experimental conditions across all FOVs acquired are reported. On high density continuous fields the amoeboid phenotype predominates (59 % amoeboid) while on low density continuous fields and islands the keratocyte-like phenotype predominates (70 % and 78% keratocyte-like respectively).

Additionally, we observed that neutrophils could exhibit a rapid change in motile phenotype. Neutrophils migratory in the amoeboid phenotype on high density continuous fields could transform into the keratocyte-like phenotype on the order of seconds when they moved from continuous fields to stamped-off islands. The movie corresponding to Fig. 4.5 G is provided as Electronic Supplementary Material Movie S1. There was a small degree of convective flow in the system that allowed neutrophils to transit across the non-adhesive domain between fields and islands. It is important to note that this non-
Figure 4.5 Neutrophils sense density at whole cell length scale. On continuous fields of FN neutrophils assume an (A) amoeboid phenotype on high density surfaces and a (B) keratocyte-like phenotype on low density surfaces. (C) However, on discrete islands where local density is high and global density is low, neutrophils assume the keratocyte-like phenotype. Higher magnification images of (D) amoeboid phenotype on continuous field and (D) keratocyte-like phenotype on discrete islands where fluorescent signal has been superimposed. (F) Fluorescense image corresponding to G. (G) Phase contrast image of neutrophils exhibiting amoeboid and keratocyte-like phenotypes in the same FOV with no adhesion in stamp-off control domain. Timelapse movie of neutrophil motility in G is supplied as Electronic Supplementary Movie S1. Scalebars = 50 μm for A, B, C, F, and G. Scalebars = 10 μm for D and E.
Figure 4.6 Observation of each phenotype per experimental condition. Observed phenotypes for the three experimental conditions across all FOVs acquired were manually scored. On high density continuous fields the largest fraction was amoeboid (59 %). On low density continuous fields and islands the largest fraction was keratocyte-like (70 % and 78 % respectively). Other denotes cells that were adherent but not spread or had ambiguous morphologies.
adhesive domain was a control, establishing that the residual protein content between islands (Fig. 4.4 B v) was not sufficient to support adhesion. This can be concluded because the large non-adhesive band between the continuous field and discrete islands was generated by stamp-off in a manner identical to that used in the interstitial space between islands.

**Comparable Neutrophil Motility on Discrete Islands and Continuous Fields**

After 30 minutes of motility neutrophils undergoing amoeboid migration on high density continuous fields of FN (Fig. 4.7 A) achieve a greater net dispersal than their keratocyte-like counterparts on low density continuous fields (Fig. 4.7 B) as well as hybrid islands (Fig. 4.7 C). A model-independent metric of dispersal is the mean maximum displacement ($\langle \max(|\Delta r|) \rangle$) of all trajectories followed through 30 min. Cell trajectories followed less than 30 min were excluded in the computation of this analysis to avoid biasing the data. Keratocyte-like motility on low density continuous fields of FN and hybrid islands was statistically indistinguishable (Fig. 4.7 D).

To assess the evolution of the motile cells we also computed mean squared displacements (MSD) as a function of time (Fig. 4.8 A) and fit the curves with the persistent random walk model of cell kinesis ($\langle \Delta r^2(\tau) \rangle = 2S^2P[\tau-P(1-exp(-\tau/P))]$) (25–26) in terms of the best-fit parameters speed (S, Fig. 4.8 B), persistence (P, Fig. 4.8 C), and the random motility coefficient ($S^2P/2$, Fig. 4.8 D). This analysis made clear that the origin of the increased dispersion seen in the model-independent analysis (Fig. 4.7 A) was the result of amoeboid neutrophils moving at least twice as fast as keratocyte-like neutrophils (Fig. 4.8 B). Although there was no statistically significant difference in the directional persistence of the two phenotypes we did see an increase in the distribution of
Figure 4.7 Neutrophil motility on islands is comparable to low density continuous fields. Cell trajectories through 30 min of motility from single representative experiments of (A) amoeboid motility on high density continuous fields, (B) keratocyte-like motility on low density continuous fields, and (C) keratocyte-like motility on hybrid islands. Scalebar = 50 µm. Dotted red circle is mean maximum displacement ($\langle\text{max}(|\Delta r|)\rangle$) of set of 30 min trajectories. (D) Mean of the set of mean maximum displacements ($\langle\langle\text{max}(|\Delta r|)\rangle\rangle$) across all independent observations. Error bars are ± standard error of the mean (N$_{\text{experiments/condition}}$ = 6-7, n$_{\text{cell/experiment}}$ = 17-27). Asterisk denotes significant difference between populations as computed by post-hoc Dunn-Sidak multiple comparisons method (p < 0.05).
Figure 4.8 MSD analysis of neutrophil motility on islands and fields. (A) Mean squared displacements from single representative experiments for each condition. Dotted red line is fit of persist random walk model (PRW) to empirical data. Mean of the set of model fit parameters (B) speed, (C) persistence, and (D) the random motility coefficient for all independent observations. Error bars are ± standard error of the mean (N_experiments/condition = 6-7, n_cell/experiment = 36-42). Asterisk denotes significant difference between populations as computed by post-hoc Dunn-Sidak multiple comparisons method (p < 0.05).
persistence values of keratocyte-like neutrophils on islands (Fig. 4.8 C). Finally, we observe a statistically significant increase in amoeboid migration relative to keratocyte-like migration in terms of the random motility coefficient as this metric is dominated by cell speed (Fig. 4.8 D).

Discussion

We previously reported the ability of adhesion ligand density to dictate the shape and mode of neutrophil migration on equally stiff substrates (11). A question that arose from that work was the length scale of the ligand density sensitivity. To address this question we employed the stamp-off method of microcontact printing (17) and engineered adhesive environments to present neutrophils with two adhesive length scales simultaneously. By careful control of protein loading for a given adhesive geometry we were able to achieve a three condition experimental state space that straddled the critical adhesive threshold (44 % saturation) which delineated the keratocyte-like phenotype from the classical amoeboid phenotype. The three conditions explored were as follows: a continuous field of FN at high density (> 44 % saturation) known to elicit the amoeboid phenotype, a continuous field of FN at a low density (< 44 % saturation) known to elicit the keratocyte-like phenotype, and a hybrid island array where the on-island density was high but the area average density was low. On these hybrid adhesive surfaces neutrophils robustly assumed the keratocyte-like phenotype, integrating the adhesive stimuli over their entire contact interface and responding as if the set of discrete islands were a continuous field.

The integration of distributed adhesive contact into a global cell response has been observed in a variety of mesenchymal cells (20, 22). Lehnert and coworkers
explored a large state space of island size and spacing in fibroblasts and melanoma cells and found that these cells spread on discrete islands of $< 1 \, \mu m^2$ with pitch $< 5 \, \mu m$ as if they were continuous fields of protein. Likewise we observe that neutrophils spread and are motile on discrete islands of $0.64 \, \mu m^2$ and $1.9 \, \mu m$ pitch as if they were continuous fields of low density protein. However, the dramatic reduction in contact area that occurs when neutrophils assume the highly motile amoeboid phenotype is quite distinct from the behavior of mesenchymal cells in the presence of high density adhesive stimulation. It is interesting to note that the reduced contact area of amoeboid neutrophils ($\sim 100 \, \mu m^2$, although admittedly difficult to measure in brightfield) is within the same order of magnitude as the actual adhesive contact of keratocyte-like neutrophils on islands ($\sim 100$ islands/cell X $0.64 \, \mu m^2$/island). Thus the phenotypic switch in neutrophils could be driven by the cell’s attempt to maintain a constant level of adhesive stimulation across its contact interface.

While the islands we employed in this study are submicron they are large compared to the lateral distance between adhesive ligand binding sites of 58-73 nm necessary to support integrin receptor clustering (18). Clustering of $\beta_2$ (CD11b) integrins and the downstream cytoskeletal rearrangement that results is critical to the neutrophil’s execution of terminal effector functions like reactive oxygen intermediate generation and proteolytic enzyme secretion (27). We previously showed, using function blocking antibodies, that neutrophils utilize the promiscuous integrin receptor MAC-1 ($\alpha_M \beta_2$) to support haptokinetic migration and density sensitivity (11). Therefore the keratocyte-like phenotype on islands suggests neutrophils do not respond to adhesive ligand density on
the receptor cluster length scale but rather integrate the total adhesive stimulus across all clusters.

The pursuit of constant adhesive stimulation across the contact interface may itself by the consequence of the cell attempting to maintain tensional homeostasis. Our lab has previously demonstrated that neutrophil traction stresses are highest in the rear uropod of motile amoeboid neutrophils (28). This asymmetric rearward contractility is understood to be the mechanism by which the cytoplasm is propelled forward generating a protrusive force despite the contact footprint of the cell being quite small. This is in contrast to the behavior of less polarized mesenchymal cells which show a linear increase in traction generation as contact footprint increases (29). Whereas the keratocyte-like phenotype presumably represents a state of high traction generation doing work against the substrate, the amoeboid phenotype represents a state of high traction generation doing work to deform the cell body itself. In neutrophils these distinct states are archived on equally stiff substrates but elicited by the extent of adhesive stimulation imparted to the cell.

Our findings may have applicability to the study of cancer metastasis and specifically the epithelial to mesenchymal transition model (30-31). It has been established that tumor stiffening drives integrin clustering which supports the malignant cell phenotype (32). Perhaps a concurrent increase in extracellular adhesivity of the stiff tumor microenvironment could subsequently induce a highly motile amoeboid-like transition in previously stationary malignancies.
Acknowledgements

We are grateful to Eric Johnston for laboratory assistance and Christopher S. Chen, PhD and Ravi A. Desai, PhD for their time and expertise in teaching us the stamp-off method of microcontact printing. Funding for this work was provided by a National Science Foundation Graduate Research Fellowship to SJH and a grant from the National Institutes of Health (HL18208) to DAH.
References


Chapter 5

Protrusive and Contractile Forces of Spreading Human Neutrophils

Preface

The content of this chapter has been adapted from the version under revision at *Biophysical Journal*. The manuscript was coauthored by Steven J. Henry, Christopher S. Chen, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: SJH designed and executed experiments, analyzed data, and wrote the manuscript; CSC provided mPADs masters and edited the manuscript. JCC consulted on design of analysis routines, data interpretation, and edited the manuscript. DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose will be retrievable from the published version online.

Abstract

Human neutrophils are mediators of innate immunity and undergo dramatic shape changes at all stages of their functional life cycle. In this work we quantified the forces associated with a neutrophil’s morphological transition from a non-adherent, quiescent sphere to its adherent and spread state. We did this by tracking, with high spatial and temporal resolution, the cell’s mechanical behavior during spreading on microfabricated-post-array-detectors (mPADs) printed with the extracellular matrix protein fibronectin.
Two dominant mechanical regimes were observed: transient protrusion and steady state contraction. During spreading, a wave of protrusive force (75 ± 8 pN/post) propagates radially outwards from the cell center (at a speed of 206 ± 28 nm/s). Once completed, the cells enter a sustained contractile state. While post engagement during contraction was continuously varying, posts within the core of the contact zone were less contractile (-20 ± 10 pN/post) than those residing at the geometric perimeter (-106 ± 10 pN/post). The magnitude of the protrusive force was found to be unchanged in response to cytoskeletal inhibitors of lamellipodium formation and myosin II mediated contractility. However, cytochalasin B, known to reduce cortical tension in neutrophils, slowed spreading velocity (61 ± 37 nm/s) without significantly reducing protrusive force. Relaxation of the actin cortical shell was a prerequisite for spreading on post arrays as demonstrated by stiffening in response to jasplakinolide and the abrogation of spreading. ROCK and myosin II inhibition reduced long term-contractivity. Function blocking antibody studies revealed haptokinetic spreading was induced by \( \beta_2 \) integrin ligation. Neutrophils were found to moderately invaginate the post arrays to a depth of approximately 1 \( \mu \)m as measured from spinning disk confocal microscopy. Our work suggests a competition of adhesion energy, cortical tension, and the relaxation of cortical tension is at play at the onset of neutrophil spreading.

**Introduction**

Neutrophils are white blood cells of the innate immune system. They act as first responders to tissue trauma (1) and pathogen challenges (2), initiating the body’s inflammatory response on the timescale of seconds to minutes. Central to neutrophil
function is spreading in which the cell begins as a quiescent sphere and becomes well-spread and migratory (3). There are numerous observations of the dynamics of neutrophil spreading in vitro. Lomakina et al. (4) measured neutrophil spreading as haptokinetically-stimulated by immobilized fields of the chemokine interleukin 8. Sengupta et al. (5) measured neutrophil spreading on continuous fields of fibronectin, induced by soluble formylated chemotactic peptide. Using reflection interference contrast microscopy, it was observed that regions of closest membrane contact to the substrate were present at the periphery of the spreading cell. It was hypothesized that these regions would ultimately correspond to domains of high force generation. In neither study were the tractions associated with neutrophil spreading directly measured.

Our goal in this work was to measure the forces of neutrophil spreading on microfabricated-post-array-detectors (mPADs). While mPADs have long been used to measure forces in mesenchymal cells (6-10) they have only recently been employed to study immune cell function. Ricart et al. (11) used mPADs to measure the traction stresses of dendritic cells undergoing chemotaxis and established that these cells migrate by a frontward pulling mechanism. Bashour et al. (12) explored the mechanics of T-lymphocyte activation and spreading on mPADs functionalized by antibodies to the activation receptors CD28 and CD3. While the mechanodynamics of T-lymphocyte spreading were measured, the role of the cell cytoskeleton was not investigated.

Here, we report the protrusive and contractile behavior of spreading neutrophils with high spatial and temporal resolution on fibronectin printed mPADs. Spreading was a fast, radially symmetric wave sufficiently forceful to generate outward deflections of the underlying posts. After protrusion, cells contracted with posts on the perimeter of the
contact zone exhibiting higher contractility than those in the core. Small molecule inhibitor perturbations of the cellular cytoskeleton revealed that cortical actin relaxation was critical upstream of protrusion but protrusion itself was not myosin II dependent. Conversely, long-time sustained contractility was dependent on ROCK and myosin II. Function blocking antibody studies revealed that haptokinetic spreading on fibronectin was β2 integrin induced. Confocal z-stacks uncovered moderate post invagination into the cell body which was ultimately fortuitous in reporting the energy associated with the quiescent-to-spread shape change.

Materials and Methods

Media and Reagents

Rinsing buffer was Hanks’ Balanced Salt Solution (Life Technologies, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Life Technologies) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca2+ and 2 mM Mg2+. Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Labeling of FN via Alexa Fluor carboxylic acid, succinimidyl ester (Life Technologies) was performed in accordance with the manufacturer’s recommended protocol. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2% w/v in PBS without calcium and magnesium (“PBS(-)”). Stock delta9-DiI lipophilic membrane dye (Life Technologies) was prepared in 200 proof ethanol at 50 ng/mL. All solutions were sterile filtered or prepared sterile. The bicinechonic acid protein assay (Pierce Biotechnology, Rockford, IL) was performed on stock FN solutions to measure concentration. Poly(dimethylsiloxane) (PDMS) was Sylgard 184 Silicone Elastomer from
Dow Corning (Midland, MI) prepared per the specified weight ratio of base:cure agents, mixed vigorously, and degassed until optically clear. Silane was Trichloro(1H, 1H, 2H, 2H-perfluorooctyl)silane from Sigma (Saint Louis, MO).

**microfabricated-Post-Array-Detectors (mPADs) and Microcontact Printing**

mPADs were fabricated and printed as detailed by Yang et al. (13). Scanning electron microscopy of our cast arrays (Fig. 5.1A) allowed us to characterize the post geometry (diameter = 604 ± 31 nm, length = 5.576 ± 0.286 μm, m ± sd) and compute an associated spring constant, $k_{spring} = 0.28 \pm 0.09$ pN/μm. For these posts, with length tenfold longer than width, the Schoen et al. (14) substrate-warping correction to $k_{spring}$ of seven percent was less than the measurement error.

The positive silicon masters were manufactured in and provided directly by the Chen laboratory. From these positive masters, negative PDMS reliefs were cast then silanized by vapor deposition. Silanized molds were coated with a small amount of 10:1 base:cure (w/w) PDMS and degassed. The 10:1 PDMS for positive casting was carefully weighed using a calibrated analytical balance and thoroughly mixed and degassed before coating the molds. PDMS coated molds were pressed against the oxygen plasma cleaned glass coverslips and leveled in a 110 °C for 20 hr. After curing, molds were released in a shallow dish of 200 proof ethanol and sonicated for 2 min. Posts were recovered in a Samdri-PVT-3D critical point drier (Tousimis, Rockville, MD) and stored in a dessicator jar until use.

25:1 base:cure (w/w) PDMS stamps were cast against a silicon wafer to produce an extremely smooth surface. Stamps were trimmed to approximately 25 mm$^2$, sonicated in 200 proof ethanol for 10 min, rinsed twice in diH$_2$O and dried in a gentle stream of
filtered N\textsubscript{2(g)}. The surface of the PDMS stamp, previously cast against the silicon wafer, was incubated with 50 μL of 100 μg/mL FN-AF488 in PBS(-) for 1 hr at room temperature (RT). After incubation stamps were rinsed twice in a submerging quantity (~50 mL) of diH\textsubscript{2}O and dried in a gentle stream of filtered N\textsubscript{2(g)}. Cast mPADs on coverslips were loaded in autoclaved Attofluor chambers (Life Technologies) and rendered hydrophilic by 7 min treatment in ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) (15). Inked stamps were inverted and set atop the UVO treated post tips. After contact the chamber was flooded with 200 proof ethanol and the stamp removed in one quick motion. Posts were observed under fluorescence microscopy to verify printing fidelity and post viability. Subsequently, posts were stained with DiI to facilitate long-duration tracking by incubation for 15 min at RT in the dark. After staining, ethanol was exchanged for PBS(-) via repeated and gentle rinsing. Substrates were blocked against non-specific binding by submerging in 0.2% w/v F-127 in PBS(-) and incubating 30 min at RT. After blocking, F-127 was exchanged for PBS(-) by repeated and gentle rinsing with running buffer. Chambers were pre-warmed to 37 °C in a cabinet incubator before cell plating and imaging.

**Calculation of Post Spring Constant**

For a cantilever beam of uniform cross section carrying a load at its unconstrained terminus, the equation of the beam’s elastic deformation curve is given by (Eq. 5.1):

\[ F = \left( \frac{3EI}{L^3} \right) \delta \]  

(Eq. 5.1)

where F, E, I, L and \( \delta \) are the force exerted at the unconstrained terminus, Young’s modulus of the beam material, moment of inertia of the beam cross section, and the
resulting deflection respectively (16). The assumption being made is that the beam is subjected to small deflections which do not cause plastic deformation. The terms within the parenthesis of Eq. 5.1 are collectively referred to as the post spring constant ($k_{spring}$, Eq. 5.2):

$$k_{spring} = \frac{3EI}{L^3}$$  \hspace{1cm}  (Eq. 5.2)

Since our fabrication protocol is identical to Yang et al. (13) we employ their measured Young’s modulus for PDMS cured 20 hr at 110 °C of $E = 2.5 \pm 0.5$ MPa. The moment of inertia $I$ for a circular cross section is (Eq. 5.3):

$$I = \frac{\pi d^4}{64}$$  \hspace{1cm}  (Eq. 5.3)

Where $d$ is the diameter of the post. Substituting (Eq. 5.3) into (Eq. 5.2) yields (Eq. 5.4):

$$k_{spring} = \frac{3E\pi d^4}{64L^3}$$  \hspace{1cm}  (Eq. 5.4)

Using scanning electron microscopy we captured a series of micrographs and measured the post diameter, $d = 604 \pm 31$ nm (m ± sd), and length, $L = 5.576 \pm 0.286$ μm (m ± sd). Using error propagation (Eq. 5.5) we computed the mean and standard deviation ($\sigma_{k_{spring}}$, Eq. 5.5) of our empirical spring constant as $k_{spring} = 0.28 \pm 0.09$ pN/nm (m ± sd).

$$\sigma_{k_{spring}} = \sqrt{\left(\frac{\partial k_{spring}}{\partial E}\right)^2 \sigma_{E}^2 + \left(\frac{\partial k_{spring}}{\partial d}\right)^2 \sigma_{d}^2 + \left(\frac{\partial k_{spring}}{\partial L}\right)^2 \sigma_{L}^2}$$  \hspace{1cm}  (Eq. 5.5)

Work by Schoen et al. (14) demonstrated that in low aspect ratio posts (i.e. posts short compared to their width) substrate warping at the base was a substantial contribution to the observed deflection. The authors constructed a table of correction...
factors to reduce the apparent spring constant as a function of post aspect ratio. For our posts with \( L/d = 9.2 \) the interpolated Schoen et al. reducing factor (assuming Poisson ratio of 0.5) is 7.8%. This correction is less than the propagated error in our empirical spring constant calculation and so we anticipate substrate warping is not a major contribution to the observed deflections in this study.

**Neutrophil Isolation**

Blood was collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Cells were isolated as previously described (17). Whole blood was obtained from human donors via venipuncture and collected in sodium heparin Vacutainers (BD Biosciences). Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24 hrs prior to donation. Blood samples were allowed to cool to RT for 15 min and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45-60 min at 550-650 \( \times \) \( g \) and 21 °C. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 10 min at 250 \( \times \) \( g \) and 21 °C. Red Blood Cells (RBC) were eliminated from the resulting cell pellet via hypotonic lysis. After lysis, vials were centrifuged for 10 min at 250 \( \times \) \( g \) and 21 °C and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at \( 10^6 \) cells/mL on a tube rotisserie at 4 °C until time of plating.

**Spreading Experiments**

Brightfield and fluorescence microscopy were performed using a spinning disk confocal. Prior to cell plating, the experimental chamber was mounted on a 37 °C
temperature controlled stage. Images were acquired with a 60X water-immersion lens at a frame rate of 1 frame/sec. Acquisition began prior to cell plating. A small volume of suspended neutrophils were introduced into the experimental chamber and allowed to gravity-sediment onto the FN printed mPADs.

**Antibody Blocking and Cytoskeletal Inhibitor Studies**

To assess the role of $\beta_2$ integrins in neutrophil spreading and adhesion on post arrays, quiescent neutrophils were incubated for 30 min at 4 °C on a tube inverter with anti-$\beta_2$ clone L130 (BD Biosciences) at 50 $\mu$g/mL. This clone and concentration were previously shown by us (17) and others (18) to be a function blocking antibody of neutrophil adhesion on FN. To assess the roles of various cytoskeletal components during spreading, quiescent neutrophils were incubated for 30 min at 4 °C on a tube inverter with the small molecule inhibitor at the stated final concentration. The corresponding experimental chamber was pretreated at 37 °C for 30 min with the same inhibitor concentration. The small molecule inhibitors explored, having previously been demonstrated to alter hematopoietic cell mechanics, were: 5 $\mu$M blebbistatin (Sigma) (19), 1 $\mu$M CK666 (Sigma, Lot: 043M4606V) (20), 3 $\mu$M cytochalasin B (Sigma) (21), 1 $\mu$M jasplakinolide (Life Technologies) (19, 22), and 1 $\mu$M Y27632 (EMD Millipore, Billerica, MA) (23).

**Cell Profile Imaging**

To map the neutrophil vertical profile during spreading, cell membranes were stained with the lipophilic dye delta9-DiI (DiI) at a final concentration of 50 ng/mL for 15 min on a tube inverter at 4 °C. Cells were rinsed twice with fresh storage buffer by gentle centrifugation at 200 x g for 5 min. Z-slices were acquired at 0.25 $\mu$m intervals.
For membrane staining experiments, posts were labeled with AlexaFluor-488 conjugated FN (FN-AF488) only, not DiI.

**Data Analysis**

Fluorescent image stacks focused on the plane of post tips were processed via a series of custom MATLAB (The MathWorks, Natick, MA) scripts. These scripts identified fluorescently labeled post centroids, connected centroids in consecutive frames to form trajectories, dederifted the trajectories, and positioned them relative to their undeflected resting lattice locations. Aspects of our scripts were adapted from the publicly available MATLAB routines (24) of Pelletier et al. (25) which were based upon Crocker and Grier’s original particle tracking code (26). Annotated code used in the analysis of this chapter is reported in Appendix B.

**Identification of Cell-Engaged Posts**

After trajectory dederifting, constructing a scatter plot of the variances in the tangential and radial directions reveals two populations of trajectories. A compact cloud of data with low variance corresponds to the posts in the field of view outside of the cell-substrate contact zone. The remaining posts correspond to those within the cell-substrate contact zone and are considered “cell-engaged”.

**Cell Reference Frame Coordinate System**

The strong directional bias of peripheral posts (Fig. 5.1 D) towards the cell centroid motivated us to translate post trajectories from a laboratory reference frame \((x(t), y(t))\) into a cell reference frame \((r_{\parallel}(t), r_{\perp}(t))\). For each post a vector connecting the geometric centroid and the resting lattice position of that post was constructed. This
vector was used to position the orthogonal \( (r_\parallel(t), r_\perp(t)) \) pair such that the radial axis was parallel with the connecting vector.

**Computing a Spreading Velocity**

For each cell we constructed a scatter plot of the time at which a post’s maximum protrusive force was observed (relative to the first protrusive event which denoted the onset of spreading) as a function of the radial distance of the post from the cell centroid. A best fit linear equation was computed, subject to the constraint \( t_{F_{\text{max}}}(r) = 0 \). The inverse of the slope of the curve was the propagation velocity. The mean velocity quoted in Fig. 5.5 C) is the mean and standard error of the ensemble of 14 spreading velocities acquired in this manner.

**Results and Discussion**

**Neutrophil Spreading on mPADs**

Quiescent neutrophils were capable of spreading atop a plane of FN printed post tips. The onset of spreading was concomitant with strong outward deflections observed at a few posts in the center of the final contact zone and propagated in a radially symmetric wave until the cell’s final and maximum spread area was reached (Fig. 5.1 B). This transient protrusive signature was replaced by a sustained contractile phase a few minutes after spreading ceased. The complete spreading sequence with superimposed deflection vectors of Fig. 5.1 B is provided in Movie S1. Post positions were tracked in the fluorescence channel as cell lensing obscured tip detection under brightfield microscopy. Cell-engaged posts experienced significant deflections compared to their non-engaged counterparts (Fig. 5.1 C). This fact was exploited to filter cell-engaged from non-engaged posts in the field of view by considering the variance of the trajectories (Fig. 5.2). The
Figure 5.1 Human neutrophil spreading on fibronectin printed mPADs. (A) SEM image of mPADs used in this study. (B, top row) Brightfield frames from time-lapse sequence of a single neutrophil rapidly spreading across an array of posts. (B, bottom row) Corresponding frames in fluorescence channel of post tips with superimposed deflection vectors (enlarged 5X to aid visualization). Frames were taken from the full time-lapse sequence provided in Movie S1. (C) Trajectories of each post in (B) as recorded for 25 min. Red crosshairs denote the resting lattice position of the undeflected posts. The dotted circle is the enlarged post trajectory of panel (D), where the red asterisk marks the time average position of the trajectory.
**Figure 5.2 Identification of cell-engaged posts by variance analysis.** Scatter plot of post trajectory variances in the tangential (⏞) and radial (⏞) directions for data corresponding to Fig. 5.1 and Fig. 5.4. The compact cloud of posts with low variance corresponds to posts outside the cell-substrate contact zone. The remaining diffuse cloud is declared “cell-engaged” and used in data analysis. 1 pixel = 0.192 μm.
enlargement of a single perimeter post (Fig. 5.1 D) reveals a strong radial bias in the post’s motion away from and towards the center of the cell’s final contact zone.

From post deflections, we quantified force trajectories in the cell reference frame in the radial and tangential directions (Fig. 5.3). For each post, a force trajectory was constructed by multiplying the deflection from resting lattice position with the known spring constant of the posts (k_{spring} = 0.28 ± 0.09 pN/nm). The force detection floor for our system was 9 ± 2 pN as determined by calculation of the mean displacement of posts not contacted by the cell and subsequent application of the spring constant. At maximum cell-generated protrusion and contraction this detection threshold resulted in signal-to-noise ratios of 8:1 and 12:1 respectively.

When we compared an ensemble plot of the radial force of each post between the periphery and the core as a function of time (gray lines, Fig. 5.4 B), a clear stratification of the data occurred. By mapping the deflection trajectories of posts within the top (low contractility) and bottom (high contractility) bands to the spatial position of the posts in the contact zone two groups of posts emerged. Perimeter posts were generally strongly contractile at long times as compared to core posts. However, both sets exhibited a strong transient protrusive spike. The ensemble averages of Fig. 5.4 B show two major mechanical regimes: initial transient protrusion and long-time sustained contraction. While tangential deflections were present throughout the experiment, no net asymmetry in the form of cell rotation or twist was observed in either perimeter or core posts (Fig. 5.4 C).
Figure 5.3 Cell reference frame coordinate transformation schematic. In the lab reference frame post trajectories are positioned relative to the field-of-view origin. A cell reference frame is more intuitive and constructed by translating post trajectories in terms of the orthogonal axes ($r_\parallel(t)$, $r_\perp(t)$) such that the radial axis is parallel to a vector connecting the geometric centroid of the cell and the resting lattice position of the post.
Figure 5.4 Spatial dichotomization of force trajectories. (A) Brightfield and fluorescence channel frames at $t = 300$ s from Fig 5.1 B. Green circles and red diamonds denote the subset of posts residing in the core and at the perimeter of the contact zone respectively. (B) Radial force trajectories over time. (C) Tangential force trajectories over time. In (C) and (D) individual gray lines correspond to individual cell-engaged posts. Ensemble averages of the subset of perimeter (red diamonds) and core (green circles) posts are superimposed.
**Metrics of Spreading and Contractility**

The behavior of the single spreading neutrophil illustrated in Fig. 5.1, Fig. 5.4, and Movie S1 is representative of our entire set of observations of spreading under control conditions \(n = 14\) cells, 4 different donors, 386 post trajectories) as shown in Fig. 5.5A. Whereas in Fig. 5.4B the mean curves were of the ensemble of posts beneath a single cell, in Fig. 5.5A the mean curves are of the ensemble of all mean trajectories for our entire set of 14 spreading cells. To achieve this mean-of-means, the independent mean radial trajectories were aligned on their respective protrusive maxima and assigned the elapsed event time \(\tau = 0\).

The qualitative and quantitative similarity of the protrusive event for core and perimeter posts is evident in the expanded view of Fig. 5.5B in which the forcefulness and duration of the protrusive events are similar. The protrusive event was immediately followed by a contractile rebound. Outwardly deflected posts did not settle back to their resting lattice position but were summarily deflected inwards. In the core of the cell, the rebound resulted in a transient contractile maximum that relaxed to a less contractile steady state. However, in the perimeter, the posts continuously deflected to a steady state contractile maximum.

To better capture the wave-like propagation of the protrusive front during spreading we plotted the time at which protrusive force was a maximum as a function of the radial distance of the protrusive event from the cell centroid for each cell and fit the data with a linear equation. The inverse of the best-fit slope was the cell’s spreading velocity. Fig. 5.5C shows the ensemble best-fit equation for all spreading events (all per-
cell fits are reported in Fig. 5.6). Using this analysis we computed a mean neutrophil spreading velocity of 206 ± 28 nm/s (m ± sem).

We considered a variety of metrics to characterize the radial forces during the transient protrusive (Fig. 5.5 D) and steady state contractile (Fig. 5.5 E) regimes. Consistent with our qualitative observations, the protrusive signatures of core and perimeter posts were not significantly different with respect to the maximum force generated (≈ 75 pN) (Fig. 5.5 D i), duration of the protrusive deflection (FWHM ≈ 17 s) (Fig. 5.5 D ii) or the variance in the ensemble of maximum forces (≈ 24 pN²) (Fig. 5.5 D iii). We did however find a significant decrease in the fraction of perimeter posts (perim: 0.67 ± 0.05) that exhibited a protrusive spike as compared to the fraction of core posts (core: 0.83 ± 0.05) (Fig. 5.5 D iv). Thus, during spreading, when a post was protrusively engaged by the cell, the basic dynamic form of the deflection did not depend on whether the post was at the core or the periphery. However, as distance from the cell centroid increased the occurrence of protrusion decreased.

Within the steady state contractile regime we found significant differences in core and perimeter posts with respect to the sustained contractile force (core: -20 ± 10 vs. perim: -106 ± 10 pN/post) and its variance (core: 16 ± 4 vs. perim: 46 ± 4 pN²/post). Perimeter posts were five times more contractile (Fig. 5.5 E i) and had three times greater variability (i.e. larger distributions in force) in their sustained contractility (Fig. 5.5 E ii) compared to their core counterparts. Our observation that spread neutrophils were most contractile at their periphery compliments the RICM measurements of spreading neutrophils on FN by Sengupta et al. (5). In that prior work, the region of intimate membrane-substrate contact was located at the periphery of the spreading neutrophil. It
Figure 5.5 Characterizing protrusion and contraction using the ensemble of neutrophil spreading events. (A) Mean radial force trajectories of core (green) and perimeter (red). The transient protrusive and steady state contractile regimes are denoted by the cyan and lavender shaded regions respectively. (B) An expanded temporal resolution of the protrusive regime in A. (C) The time at which protrusive force is maximal as a function of radial distance from the cell centroid. Per cell fits are shown in Fig. 5.6. (D) Mean metrics of transient protrusion: (i) force maximum, (ii) spreading duration via full width at half force maximum, (iii) variance in the ensemble of force maxima, and (iv) the fraction of posts in each geometric group that exhibited a protrusive spike (i.e. the participation ratio). (E) Mean metrics of steady state contraction: (i) force, (ii) variance in the ensemble of mean steady state force. All error bars are ± standard error of the mean (n = 14 cells). Asterisk denotes significant difference between populations as computed by post-hoc Tukey least significant difference method (p < 0.05).
Figure 5.6 Per-cell spreading velocity analysis. Scatter plots for 14 spreading cells of the time at which a post’s maximum protrusive force occurred as a function of the post’s radial distance from the cell centroid.
was hypothesized there, and experimentally demonstrated here, that those regions of intimate membrane-substrate contact are concurrently regions of greatest force generation.

Contrasting our work with Bashour et al. (12), we see greater protrusive and contractile behavior of spreading neutrophils as compared to T-lymphocytes. Spreading neutrophils were approximately six fold more protrusive and two fold more contractile than activated T-lymphocytes. Bashour and coworkers describe a transient regime between spreading and steady state contraction in their data in which T lymphocyte tractions were highly uncoordinated. In our data, we do not see a latent period of uncoordinated traction. Rather, we observe outward protrusion immediately followed by an inward contractile rebound. At the perimeter, this rebound evolves into a highly contractile steady state. It is important to note that the Bashour et al. work was considering the mechanics associated with T-lymphocyte activation through the CD3 T-cell receptor (TCR) and the CD28 coreceptor. Ligation of these receptors induces cytoskeletal rearrangement but is upstream of integrin activation, representing an inside-out pathway. While the Bashour et al. inside-out activation route shares certain scaffolding proteins (e.g. SLP-76) with the outside-in activation route we are engaging in neutrophils, the pathways are not identical (27).

**Biochemical Perturbations of the Cell Cytoskeleton**

To study the role of the cytoskeleton during neutrophil spreading on post arrays we pretreated quiescent cells with small molecule inhibitors targeting various cytoskeletal components. Actin in a quiescent neutrophil is confined to a thin cortical shell proximal to the cytoplasmic membrane (28). It has been demonstrated that this actin shell gives
rise to cortical tension (21-22). We began by considering the effect of jasplakinolide on neutrophil spreading. Jasplakinolide is a cyclic depsipeptide capable of polymerizing and stabilizing filamentous actin (29). In neutrophils, pretreatment with jasplakinolide has been shown to increase the rigidity of the cortex as measured by micropipette aspiration (22). When we treated quiescent neutrophils with jasplakinolide, the ability of the cells to spread was completely eliminated (Movie S2). Interestingly, the cells were still sensing the presence of the FN as detected by the formation of small processes uniformly decorating the cell body seen with brightfield imaging. These processes were never observed in untreated control cells. It is unclear whether the effect of jasplakinolide in our cells was to stabilize existing F-actin structure or deplete a pool of free actin by polymerizing excess F-actin.

Unlike jasplakinolide, cytochalasin B has been shown to decrease cortical rigidity in neutrophils as measured by micropipette aspiration (21). Cytochalasin B is known to dramatically reduce the rate of actin polymerization and simultaneously interfere with filament-filament interactions that stabilize the actin network (30). When treated with cytochalasin B, our neutrophils were still able to spread but with a substantially reduced velocity of $61 \pm 37 \text{ nm/s}$ (Fig. 5.7 B). During spreading, the mean protrusive force exerted per post was not significantly different than observed with untreated cells. However, the duration of the protrusive event was longer as seen by a significant increase in the full width at half max force metric (Fig. 5.7 C, $<\text{FWHM}>$). Inhibition of actin polymerization and filament-filament interaction by cytochalasin B had long time effects as well, significantly decreasing the achieved steady state contractile force of perimeter posts (Fig. 5.7 C, $<\text{Fss}>$) and eliminating the contractile rebound of core posts (Fig. 5.7
A, Cytochalasin B). Considered in the context of the results with jasplakinolide, spreading requires relaxation of the actin cortical shell.

We next considered whether spreading was conceptually analogous to lamellipodium formation by inhibiting Arp2/3, the actin binding protein necessary for filament branching (31). CK666 inhibits Arp2/3 mediated branching by stabilizing the inactive conformation of the seven subunit complex (32). CK666 had no effect on the protrusive capacity of the spreading cells. These cells were not significantly different in the forcefulness or duration of protrusion than their untreated counterparts. That CK666 did not abrogate protrusion suggests the shape change associated with spreading was not analogous to lamellipodium formation, in which Arp2/3 is known to play a critical role (31). We did observe a significant increase in the variance of the forces exerted on core posts during steady state contractility (Fig. 5.7 C, <VAR(Fss)>). This result suggests that a competent actin network may normally dampen post contractility in the core.

Lastly, we hypothesized that steady state contractility would be ROCK and myosin II mediated (19) and tested this by treating neutrophils with Y27632 and blebbistatin (33), respectively. In both cases these inhibitors significantly reduced steady state contractility (Fig. 5.7 C, <Fss>) of perimeter posts but did not eliminate the contractile rebound following protrusion (Fig. 5.7 A, Y27632 and Blebbistatin). In untreated neutrophils, this contractile rebound was only observed in the ensemble of core posts. Treating with Y27632 and blebbistatin revealed that the transient rebound was also occurring in the perimeter posts but was obscured when ROCK and myosin II mediated contractility commenced. Thus the transient contractile rebound is a feature of both core and perimeter posts but masked by long time engagement of the actomyosin-mediated
Figure 5.7 Cytoskeletal perturbation via small molecule inhibitors. (A) Mean radial force trajectories of the ensemble of individual cell spreading events observed after 30 min pretreatment with the stated inhibitor. Trajectories were plotted at 150 pN intervals. (B) Effect of inhibitors on spreading velocity. (C) Effect of inhibitors on metrics of protrusion (cyan shading) and contraction (lavender shading). Asterisk denotes significant difference relative to control computed by post-hoc Tukey-Kramer multiple comparisons method (p < 0.05). Direction of arrow indicates the direction in which the inhibitor shifted the metric relative to the control, if a significant difference was found.
Figure 5.8 XZ kymograph of neutrophil spreading on stiff mPADs. Two representative spreading neutrophils on stiff FN-printed mPADs. Each frame is a vertical (XZ) contour of a spreading neutrophil labelled with the membrane dye DiI. Frames were captured every 15 sec. Double-headed arrows denote observation of necking region.
contractile apparatus at the cell periphery. The implication of this result is that the short
time transient rebound is not actomyosin-dependent.

**Spreading is Haptokinetically Induced**

Neutrophil spreading is induced by haptokinetic interaction with the printed FN. On the soft post arrays (G ~ 5 kPa) used in our traction measurements, cells assumed a sessile drop morphology (Fig. 5.9 *A iii*) as captured by spinning disk confocal microscopy z-stacks. The presence of the FN was critical in supporting the transition from a quiescent to spread phenotype. When posts are blocked with Pluronic but not printed (Fig. 5.9 *A i*), the cells remained spherical and there was no non-specific adhesion. Additionally, integrin ligation by FN was required upstream of spreading, since pre-treating quiescent neutrophils with an antibody against β2 impeded spreading (Fig. 5.9 *A ii*). Haptokinetically-induced neutrophil spreading via β2 integrins is consistent with our published observation that a portion of quiescent neutrophils could be induced to migrate on continuous fields of FN without concurrent or prior stimulation by chemoattractant or selectin-ligation and that this adhesion was mediated by the promiscuous integrin MAC-1 (αMβ2) (17).

We hypothesized that the vertical profile of neutrophils on post-arrays had a stiffness dependence and considered the cell shape when spreading on stiff arrays (G ~ 42 kPa) and extremely stiff, flat PDMS (G ~ 833 kPa). On stiff posts the height (i.e. z-extent) of the cell was reduced (Fig. 5.9 *A iv*) compared to that observed on flat PDMS printed with continuous fields of FN (Fig. 5.9 *A v*). Using Fiji (34), we fit ellipses to the vertical profiles and computed the aspect ratio (i.e. ratio of the major axis length to minor axis length). A clear monotonic trend was observed where aspect ratio of the cell
increased as stiffness increased (Fig. 5.9 B). The dependency of spread area and aspect ratio on discrete post arrays of increasing stiffness is analogous to that observed of neutrophils on continuous polyacrylamide gels of increasing stiffness (35-37). Thus, as established traction methodologies, PDMS post arrays and polyacrylamide gel systems are complementary tools in probing immune cell mechanobiology.

The FN-null and anti-β2 controls had similar aspect ratios close to unity (unity denotes a perfect circle). Monotonic trends were also revealed in circularity, roundedness, and XY cell-substrate contact area as well (Fig. 5.10). In addition to XZ profile aspect ratio in Fig. 5.9 B, we used Fiji (34) to compute XZ circularity (4π*area/perimeter², Fig. 5.10 A), roundedness (1/aspect ratio, Fig. 5.10 B), and XY contact area of the cell-substrate interface (Fig. 5.10 C). As substrate stiffness increases circularity and roundedness monotonically decreased indicating an increasing deviation from a perfect circle. Conversely as substrate stiffness increases the XY contact area monotonically increases. In all metrics FN-null and anti-β2 conditions were indistinguishable.

These results demonstrate that in our system the FN is required for neutrophils to break quiescence and spread in a β2 integrin dependent manner and that the extent of spreading increases as a function of underlying stiffness. We explored a larger range of stiffness than Bashour et al. which may explain why XY spread area increases as a function of stiffness in neutrophils but not in T lymphocytes. As a note, for posts we quote approximate shear moduli (G) computed under the assumption that the mode of cell deformation of the post is shear and exerted over its cross sectional area. This assumption is motivated by the empirical work of Lemmon et al. (7) which demonstrated that shear is a larger contribution to post deflection than torque. Alternatively, Ghibaudo
Figure 5.9 Vertical profiles of neutrophils imaged via confocal microscopy. (A) (i) Quiescent neutrophil on an array of posts blocked with Pluronic F-127, but not printed with FN. (ii) A neutrophil on soft FN posts, pre-treated with anti-β₂ integrin antibody. (iii) Spread neutrophil on soft FN posts used in traction mapping. (iv) Spread neutrophil on stiffer FN post arrays. (v) Highly spread neutrophil on extremely stiff, flat FN fields. (B) Aspect ratio of best-fit ellipses to neutrophil profiles. Error bars are ± standard deviation (n = 8-15 cells per condition). Additional metrics showing similar monotonic trends are reported in Fig. 5.10.
Figure 5.10 Metrics of XZ and XY cell profiles. (A) XZ circularity, (B) XZ roundedness, and (C) XY contact area. Error bars are ± standard deviation (n = 8-15 cells per condition).
and coworkers developed a theoretical description of effective array stiffness by solution of the Green’s function for a discretized substrate (38). The Ghibaudo model estimates the Young’s moduli of post arrays as being substantially softer than anticipated by a local pure shear model (Fig. 5.9 B “$E_{eff}$”).

**Estimating Extent of Post Sidewall Printing**

From our confocal z stacks we observe that neutrophils invaginate FN printed post arrays to the limit of the post sidewall printing (Fig. 5.13 A). The following series of calculations were used to estimate the extent of this sidewall printing and thus depth of invagination. An apparent image ($I$) is the convolution of the object’s intensity profile ($F$) with the optical system’s airy disc ($G$) (Eq. 5.6) (39).

\[
I = F \otimes G 
\]  
(Eq. 5.6)

Assuming the object’s intensity profile and the optical airy disc are reasonably approximated as Gaussian distributions, the convolution of two Gaussians produces a variance ($\sigma^2$) that is the sum of the variances (Eq. 5.7).

\[
\sigma^2_I = \sigma^2_F + \sigma^2_G 
\]  
(Eq. 5.7)

Using green fluorescent beads (Molecular Probe FluoSpheres, Catalog: F8813, Lot: 1600255) of known size (diameter $= 0.49 \pm 0.015 \, \mu m$) we acquired XZ intensity profiles on the spinning disc confocal in the same channel and at the same magnification as our post measurements (Fig. 5.11 B). We normalized each bead intensity profile so the area under the intensity curve equaled unity and the peak of the intensity curve was translated to reside at $x = 0$. Next, each normalized bead intensity curve was fit with a two-parameter Gaussian distribution where mean ($\mu$) and standard deviation ($\sigma$) were
free parameters (Fig. 5.11 D). The mean standard deviation of ten beads was 0.94 ± 0.09 μm (m ± sd).

To estimate the variance of the Gaussian-approximated optical airy disc we must make an assumption about the unconvolved intensity profile of the fluorescent bead. Let the unconvolved intensity profile of the fluorescent bead have a full width at half maximum (FWHM) equal to the known bead diameter. For a Gaussian distribution, FWHM is related to the standard deviation (σ) (Eq. 5.8) via:

\[ FWHM_{\text{bead}} = 2\sqrt{2\ln(2)} \sigma_{\text{bead}} \]  

(Eq. 5.8)

Rearranging for \( \sigma_{\text{bead}} \) and substituting \( FWHM_{\text{bead}} = 0.49 \) μm results in (Eq. 5.9):

\[ \sigma_{\text{bead}} = \frac{0.49}{2\sqrt{2\ln(2)}} \approx 0.21 \mu m \]  

(Eq. 5.9)

Solving Eq. 5.7 for the Gaussian-approximation to the confocal’s optical airy disc yields (Eq. 5.10):

\[ \sigma_{\text{optics}}^2 = \sigma_{\text{image}}^2 - \sigma_{\text{bead}}^2 \]

\[ \sigma_{\text{optics}}^2 = (0.94 \mu m)^2 - (0.21 \mu m)^2 \]  

(Eq. 5.10)

\[ \sigma_{\text{optics}}^2 \approx 0.8403 \mu m^2 \]

Having approximated the contribution of the optical airy disc to the blur in the XZ intensity profile of fluorescent beads of known size, we can now quantify the apparent intensity profile of the printed post arrays and calculate an estimate of the actual extent of sidewall printing. A set of post arrays, printed with FN-AlexaFluor488 in a manner identical to those used in cell spreading experiments, was imaged in an aqueous solution of 90% glycerol. Glycerol was employed to bring the aqueous refractive index closer to that of cured PDMS (Fig. 5.12 A). Next, each normalized post intensity curve was fit with
a two-parameter Gaussian distribution where mean (μ) and standard deviation (σ) were free parameters (Fig. 5.12 C). The mean standard deviation of ten printed posts was 1.00 ± 0.10 μm (m ± sd). Solving Eq. 5.7 for the variance of the Gaussian-approximation to the true intensity profile of printed posts yields (Eq. 5.11):

\[
\sigma_{\text{post}}^2 = \sigma_{\text{image}}^2 - \sigma_{\text{optics}}^2
\]

\[
\sigma_{\text{post}}^2 = (1.00 \mu m)^2 - 0.8403 \mu m^2
\]

\[
\sigma_{\text{post}}^2 \sim 0.1597 \mu m^2
\]  

(Eq. 5.11)

Lastly, we define the extent of sidewall printing as the FWHM of the unconvolved z-intensity profile (Eq. 12):

\[
FWHM_{\text{post}} = 2\sqrt{2\ln(2)}\sigma_{\text{post}}
\]

\[
FWHM_{\text{post}} = 2\sqrt{2\ln(2)}*0.1597\mu m^2
\]

\[
FWHM_{\text{post}} = 0.9410 \mu m
\]  

(Eq. 5.12)

Thus, we conservatively estimate the extent of sidewall printing to be on the order of 1 μm.

**Estimating Energy of Neutrophil-FN Interaction**

If cell wetting (i.e. FN ligation of cell surface receptors) alone drives the spherical-to-sessile drop shape change then the energy of this interaction must be sufficient to deform the known cortical tension of quiescent neutrophils. The following is an order of magnitude analysis to estimate the available binding energy of human neutrophils. The total MAC-1 availability of activated human neutrophils is on the order of ~ 10^5 receptors (40). Our antibody blocking experiments demonstrated that β_2 integrins were a major mediator of neutrophil-FN binding. From kinetic studies of β_2 integrin ligation, the energy liberated upon binding is known to be on the order of ~ -10 k_B T (41).
Figure 5.11 Confocal images of 0.49 μm diameter green fluorescent beads using the same magnification and acquisition settings as post arrays. (A) XY plan view of green fluorescent beads. (B) XZ profile view of green fluorescent beads denoted by yellow dotted line in A. (C) Raw bead intensity along yellow dotted line in B. (D) Normalized bead intensity so area beneath intensity curve equals unity and peak intensity occurs at x = 0. Red-dotted line is best-fit two parameter Gaussian curve.
Figure 5.12 Confocal measurements of FN-AlexaFluor488 printed post arrays in a solution of 90% glycerol. (A) XZ profile view of printed posts. (B) Raw post intensity along yellow dotted line in A. (C) Normalized post intensity so area beneath intensity curve equals unity and peak intensity occurs at $x = 0$. Red-dotted line is best-fit two parameter Gaussian curve.
Assuming all MAC-1 is available for binding and FN binding sites are in excess of MAC-1 than an upper estimate on the liberated binding energy ($\gamma_{\text{cell-FN}}$) is on the order of $\sim -10^6 \text{k}_B\text{T}$. Assuming surface energy alone dictates cell shape we can apply Young’s equation to relate the observed contact angle of the cell profile to the energy of cell-substrate interaction (Eq. 5.13):

$$0 = \gamma_{\text{cell-FN}} + \gamma_{\text{cell-PBS}} \cos \theta \quad \text{(Eq. 5.13)}$$

Note in Eq. 5.13 we implicitly assumed that the energy of substrate-aqueous (i.e. FN-PBS) interaction is insignificant ($\gamma_{\text{FN-PBS}} \sim 0$). Rearranging Eq. 5.13 and solving for the surface energy of the quiescent neutrophil ($\gamma_{\text{cell-PBS}}$) yields (Eq. 5.14):

$$\gamma_{\text{cell-PBS}} = -\frac{\gamma_{\text{cell-FN}}}{\cos \theta} \quad \text{(Eq. 5.14)}$$

From z-stacks of fluorescently labeled neutrophils on FN we can measure the contact angle that the cell forms with the substrate. Contact angles from neutrophils on flat PDMS, microcontact printed with large continuous fields of FN, were used as this case represents the maximum binding energy available to the cell. For flat PDMS, $\theta = 15 \pm 2 ^\circ$ (m ± sd, n = 6 cells). Substituting Eq. 5.14 for $\gamma_{\text{cell-FN}} \sim -10^6 \text{k}_B\text{T}$ and $\theta = 15 ^\circ$ yields $\gamma_{\text{cell-PBS}} > 10^6 \text{k}_B\text{T}$. The surface energy of the quiescent neutrophil is the cortical tension ($T_{\text{cort}}$) multiplied by the surface area $SA$ (Eq. 5.15):

$$\gamma_{\text{cell-PBS}} = T_{\text{cort}} \times SA \quad \text{(Eq. 5.15)}$$

Modeling the spread neutrophil as a hemispherical cap and computing the lateral surface area yields $SA = 446 \pm 56 \mu m^2$. Substituting $\gamma_{\text{cell-PBS}} \sim 10^3 \text{pN-}\mu \text{m}$ (1 $\text{k}_B\text{T} \sim 0.004114 \text{pN-}\mu \text{m}$) and $SA = 446 \mu m^2$ into Eq. 5.15 yields $T_{\text{cort}} \sim 10 \text{pN/}\mu \text{m}$ which is within one order of magnitude of the measured cortical tension of quiescent neutrophils.
Our rough analysis suggests that the upper bound of available energy of the cell-FN interaction on flat PDMS is on the order of the surface energy associated with the resting neutrophil’s cortical tension. However, the actual binding energy is likely lower on the discretized adhesive environment of the printed post arrays.

Furthermore, if the energy of cell-substrate binding alone were sufficient to explain the deformation we would have expected that reducing cortical tension and decreasing viscosity via cytochalasin B treatment would have increased the spreading velocity of neutrophils as was observed of cytochalasin D treated HeLa cells by Cuvelier et al (42). However, in the cytochalasin B case neutrophils spread slower than untreated control cells.

**Origin of the Protrusive Signal**

Simultaneous acquisition of the cell profile and plane of FN printed post tips revealed that neutrophils moderately invaginate the post arrays to a depth of approximately 1 μm (Fig. 5.13 A). Our prior experience with neutrophils on continuous fields of FN on PDMS blocked with Pluronic F-127 (17) and the absence of spreading in the present FN-null experiments suggests that invagination was a consequence of printing adhesive ligand on the post sidewalls. Sidewall printing may have resulted from using soft stamps to print the post arrays coupled with the fact that the post tips themselves were rounded.

During spreading, posts beneath the propagating cell front reported the forces associated with the cell’s shape change from quiescence (spherical) to spread (sessile drop). This was facilitated by the fact that the cell was not spreading exclusively across the top of the plane of post tips but rather through a volume of finite thickness dictated by
Figure 5.13 Post invagination as origin of protrusive signature. (A) Confocal XZ profiles of neutrophils, cytoplasmic-stained by Dil on FN-AF488 post arrays. Each field of view is a different neutrophil. (B) Schematic of (i) cell spreading through a finite volume of posts as driven by sidewall printing and (ii) a conjecture that the transient contractile rebound is driven by local membrane curvature where \( \mathbf{n} \) is a unit normal vector. Schematic is to scale. Extent of sidewall printing was estimated at 1 \( \mu \text{m} \).
the extent of sidewall printing. The posts reported the force of shape change because they physically resided within the cell’s spreading path. (Fig. 5.13 B i). Our inhibitor studies showed that ROCK and myosin II mediated contractility was not fully matured until approximately 500 s after peak protrusive force was generated. We know that FN was required for spreading as FN-null experiments did not induce shape change. Thus to claim that protrusion was the result of cell spreading across the plane of post tips but not through a finite volume suggests that integrin ligation of FN was responsible for the ~ 75 pN/post protrusive force at short times without mature connection to the actomyosin substructure which requires minutes to develop. If sidewall printing were not present we would have been unable to quantify the force associated with this transformation as connection of the mature actomyosin substructure to the integrin adhesive contacts at the cell-post interface requires minutes to develop.

The energy of the MAC-1/FN interaction was estimated to be within an order of magnitude of the energy necessary to achieve the spherical-to-sessile drop transformation resisted by the cortical tension of quiescent neutrophils. That adhesion energy alone was not in excess of the required deformation energy to achieve spreading suggests an additional mechanism was at play. Our jasplakinolide and cytochalasin B inhibitor studies point to cortical tension release as a possible biophysical mechanism neutrophils employ to permit adhesion driven spreading and invagination. Additionally, the observation of moderate post invagination suggests a possible explanation as to the origin of the transient contractile rebound present in untreated core posts and ROCK/myosin II inhibited perimeter posts. We hypothesize that this rebound results from the invaginated posts assuming a transient orientation normal to the cell membrane to minimize the
energy of the membrane-post interface (Fig. 5.13 B ii). Future experiments using time resolved superconfocal microscopy may be able to quantify the post tip orientation relative to the local membrane curvature during spreading. Additionally, future experiments using arrays with a sparse number of non-printed posts could shed light on the mechanical role of integrin ligation during protrusion.

Conclusions

As first responders to tissue trauma and infection, neutrophils are capable of fast and dramatic shape changes (3). In this work we studied the mechanics associated with a neutrophil’s transition from a quiescent sphere to a spread and integrin-adherent morphology. In vivo spherical neutrophils circulate throughout the vasculature with their shape maintained by an actin cortical shell. Others have demonstrated, using micropipette aspiration, that this shell possesses a characteristic rigidity, tunable by small molecule inhibitors of actin polymerization (22) and depolymerization (21). By observing neutrophil spreading on post arrays in the presence and absence of such inhibitors we quantified protrusive forces associated with spreading and attributed their origin to a biophysical mechanism involving a competition of adhesion energy, cortical tension, and the relaxation of that cortical tension.

Neutrophils were induced to spread on fibronectin (FN) printed post arrays as a result of their haptokinetic interaction with the adhesive ligand alone. This was consistent with our previous demonstration that a fraction of neutrophils in contact with continuous fields of FN could spread and migrate without prior or concurrent stimulation by selectin or chemoattractant (17). This haptokinetic spreading was mediated by the αMβ2 (MAC-1) integrin, a promiscuous receptor of multiple adhesive ligands. Our work with
haptokinetically activated neutrophils suggests MAC-1 promiscuity may serve as a biological safeguard, allowing neutrophils to activate at sites of trauma without executing the earliest rolling stages of the leukocyte adhesion cascade.

On flexible post arrays neutrophil spreading was mechanically detected as a circumferential ring of protrusive force (~ 75 pN/post) that propagated radially outwards (~ 200 nm/s) until the cell reached its maximum spread area. The magnitude of the protrusive force was invariant with respect to the post’s location beneath the cell. Treatment of neutrophils with CK666, an inhibitor of actin branching, had no effect on protrusion suggesting the protrusive phenomenon was not analogous to lamellipodium formation. However, small molecule inhibitors of actin polymerization and depolymerization did reveal that the quiescent-to-spread shape change required relaxation of the quiescent actin cortical shell. Stiffening cortical actin via jaspladkinolide treatment completely eliminated spreading while softening cortical actin via cytochalasin B treatment slowed spreading velocity (~ 60 nm/s). Immediately after maximum protrusion, cell-engaged posts underwent a rapid contractile rebound. At the periphery of the contact zone this contractile rebound continuously evolved into a sustained contractile force floor (~ 100 pN/post) that was five-fold greater in magnitude than the transient contractile dip experienced in the core (~20 pN/post). While initial protrusion was myosin II independent long-time sustained contractility was ROCK and myosin II dependent as demonstrated by treatment of neutrophils with Y27632 and blebbistatin respectively.

Treating cell spreading as a competition between the energy of the adhesive environment driving the cell to spread and the cell’s cohesive forces resisting shape change, has a long history (43). The equilibrium shape of such a droplet in an aqueous
medium is described by Young’s equation relating the angle of the droplet-substrate interface to the substrate-medium, droplet-medium, and substrate-droplet interfacial energies. Historically, micropipette aspiration experiments on quiescent neutrophils have motivated their treatment as viscous liquid droplets with apparent surface tension (21, 44-47). Recently, Cuvelier and coworkers developed an alternative model of cell spreading, validated in mesenchymal carcinoma cells and biotinylated red blood cells, which treats the cell as a liquid droplet surrounded by a viscous shell of finite thickness (42). The model predicts two spreading regimes: contact radius evolves as \( R \sim t^{0.5} \) at short times and \( R \sim t^{0.25} \) at long times when the adhesive patch is comparable to the size of the cell.

While we have limited resolution of the evolution of the spreading neutrophil’s contact interface with time, as a result of tracking discretized post tips and not the cell membrane itself, we can approximate the spreading velocity in terms of the propagation rate of the radial protrusive force (Fig. 5.5 C and Fig. 5.7 B). We estimate that our neutrophil contact interface grows as \( R \sim t^{0.4} \) which is consistent with our previous observations of neutrophil spreading on FN (5) and approaches the short time \( R \sim t^{0.5} \) dependency predicted by the Cuvelier model.

However there are significant differences to be noted. In particular the contact interface in the Cuvelier model and RICM validation experiments grows as a radially symmetric disk. In neutrophils this symmetry is absent. In fact, the regions of intimate cell-substrate contact are found to decorate the neutrophil’s periphery as a ring with virtually no intimate contact at the core (5). An additional discrepancy is the observation that cytochalasin B softening of the cortical shell decreases spreading velocity in neutrophils whereas cytochalasin D treatment in HeLA cells was found to increase
spreading velocity in the Cuvelier work. This later observation coupled with the additional finding that spreading is abrogated in the absence of competent integrin ligation of FN suggests that cell signaling upstream of spreading is critical and a purely physical treatment of neutrophil spreading is insufficient to reconcile the complete body of experimental work.

Our work extends previous measurements of neutrophil spreading via RICM (5) and reveals that regions of close membrane-substrate contact are concurrently regions of high force generation. Our studies also complement recent investigation into the mechanics of T-lymphocyte activation on mPADs (12) by considering the role of the cell cytoskeleton and demonstrating that relaxation of cortical tension is a critical driver of cell shape change. Physiologically, the forces associated with this quiescent-to-spread transition have not been considered as a possible pre-extravasation signal that facilitates transendothelial migration. Work by Rabodzey and coworkers on the forces associated with neutrophil extravasation at endothelial cell junctions demonstrated that nN protrusive forces are exerted by neutrophils when rupturing VE-cadherin junctions (48). These nN forces were attributed directly to neutrophil transmigration and not neutrophil-induced endothelial contraction. That the spherical-to-spread shape change has pN protrusive forces while neutrophil transmigration is a protrusive phenomenon of nN scale suggests a synergistic relationship between transmigrating neutrophils and the underlying endothelial cells.

Future topics to be addressed include the origin of the transient contractile rebound observed in core posts and in the periphery when ROCK/myosin II are inhibited, as well as the organization of the cortical actin shell around posts during invagination.
Additionally, work by Ghassemi and coworkers demonstrated that myosin contractile units form linear chains spanning multiple submicron diameter posts as compared to forming closed rings around single micron diameter posts (10). In our study of adhesion-driven spreading of neutrophils on submicron diameter posts we observe motion or “chatter” in the spatial position of cell engaged posts. This motivates the hypothesis that such motion is biochemically correlated with the organization of these linear contractile units. Furthermore, if these mechanical linkages exist in neutrophils, studies could be performed to search for resulting correlations in neighboring posts.

The role of $\beta_2$ clustering in adhesion-driven neutrophil spreading on post arrays also remains an open question. $\beta_2$ clustering and downstream cytoskeletal rearrangement are critical to neutrophil processes such as reactive oxygen intermediate generation and enzyme secretion (49). Yu and coworkers demonstrated that $\beta_3$ integrin clustering and radially-outward motion of these clusters was upstream of mesenchymal cell spreading on supported lipid bilayers functionalized with RGD and that the basis of the radial motion was actin polymerization (50). In neutrophils, pretreatment with cytochalasin B, an inhibitor of actin polymerization, slowed but did not eliminate spreading. However, a notable difference from the Yu work is that neutrophils on FN printed post arrays spread an order of magnitude faster than mesenchymal cells on supported lipid bilayers functionalized with RGD ($\sim 200$ nm/s vs. $\sim 20$ nm/s).

**Acknowledgements**

We are grateful to Michael T. Yang, PhD for his time and expertise in teaching us mPADs production and printing. Funding for this work was provided by the National Institutes of Health grant HL18208 to DAH.
References


Chapter 6

Single Vesicle Patterning of Uniform, Giant Polymersomes

Into Microarrays

Preface

The content of this chapter has been adapted from its published version in the journal Small (2013, Vol. 9(13):2272-2276, DOI: 10.1002/smll.201202627) by permission of John Wiley and Sons (License: 3540921353732). The published manuscript was coauthored by Neha P. Kamat, Steven J. Henry, Daeyeon Lee, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: NPK produced microfluidic vesicles, performed experiments, and wrote the manuscript; SJH produced patterned substrates, performed experiments, and edited the manuscript; DL shared the microfluidic platform, consulted on data interpretation, and edited the manuscript; DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose can be retrieved from the published version online.

Abstract

Giant, cell-sized polymersomes are functionalized and patterned at the single vesicle level. Microfluidic methods are employed to generate uniform diameter vesicles with high loading efficiencies and microcontact printing is used to generate patterns of
adhesive ligand. A simple sensory capability is demonstrated with the immobilized array of vesicles.

**Introduction**

Studies with artificial cells, or protocells, in which synthetic particles are designed to replicate cellular processes are moving beyond single particles to the engineering of coordinated action among multiple particles (1-2). Cells often display multi-cellular communication and coordinate their activities, such as in quorum sensing (3) and paracrine signaling (4). Vesicles are an ideal particle to serve as the structural basis for a protocell. The design and construction of multi-vesicle systems to induce inter-particle communication, however, is challenging. Minimally, such a system requires spatial control of vesicle positioning, the encapsulation of the signaling agents, and functionalization of the responding vesicle for signal detection. Patterning vesicles with spatial precision on a substrate would enable the design and development of structurally well-defined communication systems, and have utility in other applications, such as building biosensor arrays. Microcontact printing is ideally suited towards the fabrication of an ordered array of inter-communicating artificial cells. Here, we demonstrate for the first time the patterning of individual, monodisperse, and functionalized giant polymersomes. Using microfluidics, we prepare functionalized vesicles of controlled size with high encapsulation efficiency and use microcontact printing to immobilize polymersomes in controlled spatial arrangements. Finally, we demonstrate the sensory capability of the resulting array.

Vesicles, comprised of bilayer membranes surrounding an aqueous lumen, are architecturally similar to cells, and provide the spatial compartmentalization that enable
cells to perform a variety of metabolic and sensory functions. Patterning vesicles has facilitated diverse applications ranging from bioenergetic reactions (5-6) to diagnostic assays based on specific recognition (7). Arrays of both lipid (8-10) and polymer vesicles (6, 11) have been built. Vesicles with thick membrane cores are particularly useful because they are able to incorporate hydrophobic solutes in the core of the membrane as well as aqueous solutes in the vesicle lumen. Polymersomes, bilayer vesicles made from di-block copolymers, not only have hyper-thick membrane cores, but possess additional advantages over lipid vesicles, including increased membrane strength and the flexibility to design a wide range of physical and chemical properties into the polymer through chemical synthesis (12).

Beyond technological and medical applications, immobilized bilayer vesicles can also be used to construct systems that reproduce specific functions of cells, like triggered gene expression or chemical reaction cascades (5, 13). Cellular mimicry with synthetic vesicles is quickly advancing to replicate more complex cellular behaviors, such as particle-to-particle (vesicle-to-vesicle) communication (12). For example, theoretical work by Balazs and coworkers (1, 14) has proposed that inanimate, cell sized capsules can be engineered to communicate and induce movement of one another through the exchange of soluble cues that dynamically modulate the underlying adhesive environment. A key technological advance needed to test the principles of these calculations is the assembly of arrays of vesicles with precise spatial organization.

To date, the majority of studies conducted with immobilized vesicles have been limited to small vesicles (with diameters ≤ 400 nm). Large, micron-sized vesicles, however, are closer to the dimensions of biological cells and are therefore appropriately
sized for the study of vesicle-cell communication at a biologically relevant length scale. Yet, patterning large vesicles has proven difficult. When larger, single, micron-scale vesicles have been immobilized, the vesicle size has generally not exceeded several microns (9). Arrays that are assembled with polydisperse vesicles, limit the precision of the intended pattern. Classical vesicle preparation methods, like thin-film hydration, have made it difficult to prepare monodisperse giant vesicles and high encapsulation efficiencies, and have consequently limited our ability to pattern uniform arrays of large vesicles. The advent of vesicle production methods using microfluidic techniques now enables the generation of single, giant, monodisperse polymersomes (15). These vesicles, formed through solvent evaporation from double emulsion templates, have near perfect encapsulation efficiencies and highly uniform diameters (16).

Materials and Methods

Reagents

A polyethylene oxide-polybutadiene diblock copolymer, PEO\textsubscript{30-}b-PBD\textsubscript{46}, was used for polymersome formation (Polymer Source, Montreal, Canada). Biotinylated polymer was previously functionalized in our laboratory, in which biotin was conjugated onto the terminal polyethylene oxide of PEO\textsubscript{30-}b-PBD\textsubscript{46} via an intermediate 4-fluoro-3-nitrobenzoic acid linkage that yielded polymer that was approximately 65% biotin-modified. Biocytin, Pluronic F-127, and bovine serum albumin (BSA) were purchased from Sigma. NeutrAvidin-Texas Red conjugate and Biotective Green Reagent were purchased from Life Technologies and were used to pattern substrates and demonstrate vesicle communication, respectively.
**Polymersome Preparation**

Giant polymersomes were prepared via double emulsion templates. Water-in-oil-in-water (W/O/W) double emulsions were produced using glass microcapillary devices, described previously (17). The inner aqueous phase consisted of a sucrose solution (290 mOsm), the middle, organic phase consisted of 1 mg/mL polymer in a mixture of toluene and chloroform (72:28 v/v), and the outer, aqueous phase consisted of phosphate buffered saline (PBS) (290 mOsm) containing either 1 wt % BSA or 0.1 wt % F-127. For functionalization studies, polymersomes were prepared with Pluronic F-127 as the stabilizer to ensure carboxy-linked biocytin modification occurs with the carboxy group on the polymer and not on any residual surfactant (e.g. BSA) that remains in the membrane). For all other studies, BSA was used as the stabilizer. The three fluid streams were co-focused to generate PEO<sub>30</sub>-b-PBD<sub>46</sub> double emulsions that were collected in 2 mL of PBS inside 20 mL glass vials. The vials were left loosely capped on a rocker overnight and subsequently tightly capped and rocked until use, generally between 1-2 weeks after formation. The control over vesicle size was demonstrated by changing the outer aqueous phase flow rate between 10-70 mL/hr.

**Polymersome Functionalization**

To demonstrate functionalization of giant, double emulsion-templated polymersomes, polymer vesicles were formed from either carboxy-terminated diblock copolymer, COOH- PEO<sub>30</sub>-b-PBD<sub>46</sub> or biotin-functionalized polymer. In the former case, following polymersome formation, the carboxy-terminated polymer membranes were functionalized via an EDC mediated coupling to biocytin. For polymersomes made with either biotin-conjugated polymer or covalently linked to biocytin post-vesicle formation,
NeutrAvidin-Texas Red was incubated with the vesicles to demonstrate the ability to functionalize biotin after attachment to the vesicle surface.

**Substrate Fabrication**

Micropatterned substrates containing NeutrAvidin-Texas Red islands were fabricated as described by Desai et al (18). Briefly, glass coverslips were spin-coated with poly(dimethylsiloxane) (PDMS, Sylgard 184, Dow-Corning) pre-polymer components at 10:1 (base:curing agent) ratio by weight followed by baking overnight at 60 °C. PDMS stamps were cast against a silicon wafer upon which was etched the negative relief of our desired island array using common photolithography techniques. The stamps were cured overnight at 60 °C, removed from the mask and inked with NeutrAvidin-Texas Red (100 µg/mL) for 1 h. Stamps were rinsed and applied to the substrate, as described previously. Square NeutrAvidin lattices consisted of circular islands either 50 µm in diameter and laterally spaced by 100 µm (50 µm x 100 µm) or of 10 µm diameter islands spaced by 50 µm (10 µm x 50 µm). Texas Red-labeled NeutrAvidin was used to visualize the resulting pattern and uniformity of the printed protein. Vesicle adhesion was restricted to the NeutrAvidin islands by blocking the unprinted surface with 0.2 wt % Pluronic F-127 for at least 10 min and washing thoroughly with PBS without dewetting the printed and blocked surface.

**Vesicle Patterning**

Polymersomes made with biotin-conjugated polymer were incubated on NeutrAvidin patterned substrates at a density of 80 vesicles/mm² and subjected to gentle rotation on a motorized microscope stage. The convection of fluid induced by the stage motion was found to effectively clear unbound vesicles from the patterned and blocked
PDMS substrate provided the island pitch allowed egress. Vesicles made with unmodified PEO_{30-}b-PBD_{46} were used as a control of biotin-avidin specificity as described in the text.

**Results and Discussion**

*Producing Monodisperse Populations of Micron-Scale Microfluidic Vesicles*

We prepared polymersomes from microfluidic-generated, water-in-oil-in-water (W/O/W) double emulsions that contain the amphiphilic diblock copolymer PEO_{30-}b-PBD_{46} (MW 1300 and 2500, respectively). We previously verified the unilamellar structure of these vesicles and elimination of organic solvent from their membranes (17). A sucrose solution, toluene and chloroform mixture, and phosphate buffered saline (PBS) made up the inner, middle, and outer phases, respectively, of the double emulsions. Though we have previously shown that polymersomes can be formed without the use of stabilizers, in order to increase yield in this study, the outer phase contained either 1 wt % bovine serum albumin (BSA) or 0.1 wt % Pluronic F-127. By tuning the continuous phase flow rate, we can robustly control the diameter of the resulting vesicles over a range of 20 – 70 microns (Fig. 6.1 *A*). While small adjustments to the inner and middle flow rates are required to form double emulsions at each continuous flow rate, we find the outer flow rate is the dominant variable in dictating vesicle size (Fig. 6.2 *C*). Polymersome diameter was found to be a linear function of this continuous phase flow rate and was invariant with respect to the polymer formulations tested at a given continuous phase flow rate (Fig. 6.1 *B*). Changing the polymer solution from a control PEO_{30-}b-PBD_{46}, to biotin-functionalized PEO_{30-}b-PBD_{46}, to carboxy-terminated PEO_{30-}
Figure 6.1 Controlling vesicle size and loading. (A) The diameter of polymersomes formed with a microfluidic capillary device is a linear function of the continuous flow rate used to prepare double emulsions ($n > 50$ vesicles for each data point, error bars are standard deviation (s.d.). (B) Vesicle diameter is invariant with respect to the polymer formulations tested at a given flow rate ($n > 100$). (C) Vesicles made to encapsulate a single bead follows the expected Poisson distribution, where $<\text{bead/vesicle}>_{\text{expected}} = 1.5$ ($n = 134$ vesicles, $<\text{bead/vesicle}>_{\text{actual}} = 1.3$, C.O.V. = 0.9). (D) Double emulsions are prepared with 1 µm carboxyl modified polystyrene beads in the interior aqueous compartment at loading quantities calculated in white. Scale bar is 50 µm.
Figure 6.2 Controlling vesicle diameter by adjusting the continuous phase flow rate. (A) A microcapillary device is used to prepare polymersomes by first generating double emulsions. The inner phase (I.P.), middle phase (M.P.) and continuous phase (C.P.) are co-focused to create the double emulsions. (B, C) The diameter of polymersomes can be controlled by changing the flow rates of the different fluid phases. The C.P. flow rate is the dominant variable in dictating vesicle size. Using a device with an inner capillary diameter of 32.6 µm and an outer capillary diameter of 151 µm, polymersomes were prepared with diameters ranging from 20-70 microns. Phase microscopy images of representative polymersomes are depicted for 3 different populations of vesicles appearing in graph C. (n > 50 vesicles for each data point, error bars are standard deviation (s.d.). The I.P., M.P. and C.P. flow rates for each population of vesicles that appears on graph c were: (1) 0.55, 5.0 and 5 mL hr⁻¹ (2) 1, 7 and 10 mL hr⁻¹ (3) 1, 7 and 20 mL hr⁻¹ (4) 1.5, 7 and 30 mL hr⁻¹ (5) 1.2, 7.5 and 40 mL hr⁻¹ and (6) 0.55, 5 and 60 mL hr⁻¹. Scale bar is 70 µm. This figure was presented in the supplementary text of the original manuscript.
$b$-$PBD_{46}$ resulted in the same average diameter of $59.0 \pm 0.5 \, \mu m$ demonstrating the consistency of our preparation method regardless of small changes in polymer chemistry.

**Controlling Microfluidic Payload Encapsulation**

For applications in which arrays of vesicles are to be used as bioreactors, maintaining high encapsulation efficiency and controlling the concentration of encapsulated reactants is critical (19). To illustrate the control microfluidic methods afford in precise payload encapsulation, we prepared double emulsions with different numbers of 1 $\mu m$ carboxylated polystyrene beads. An inner phase solution is prepared to contain the appropriate volume fraction of beads that results in the desired number of particles encapsulated. By determining the actual distribution of beads loaded in a population of vesicles that were prepared to have approximately 1 bead in their interior, we can assess the reproducibility and variation of particle loading in its most variable (i.e. low number) regime. When the volume fraction of particles in a given volume is low and randomly distributed, the distribution of bead loading is expected to follow a Poisson model (20). This distribution was experimentally seen for the volume fraction corresponding to a mean of $1.3 \pm 1.2$ beads/vesicle (Fig. 6.1 C). Having validated that the encapsulated number of beads can be dictated by the starting volume fraction of the inner phase solution and given that the diameter of the inner aqueous droplet is constant, we produced populations of vesicles with controlled numbers of encapsulated beads (calculated loading values are reported in Fig. 6.1 D).

**Surface Functionalizing Vesicles for Controlled Adhesion**

In order to adhere vesicles specifically to a patterned surface, the membranes must be functionalized with an appropriate ligand complementary to a ligand
immobilized on a surface. Polymers can be modified prior to vesicle production or after membrane assembly (21). If functional groups are sufficiently hydrophilic, we can advantageously assemble vesicles with pre-functionalized polymer where the number of reactive molecules on a vesicle is known and reproducible between batches. In this study, we demonstrated that microfluidic polymersomes could be functionalized through both aforementioned routes. As shown in Fig. 6.3, PEO$^{30-b}$-PBD$_{46}$ polymers conjugated to biotin were assembled into vesicles. Alternatively, polymersomes made with carboxy-terminated PEO$^{30-b}$-PBD$_{46}$ polymer could also be modified after vesicle preparation using an EDC/NHS-mediated coupling reaction to link biocytin to the carboxylic acid groups (22). The latter method results in biotin present only on the outer leaflet of the vesicle, allowing the creation of asymmetric membranes with differing functionalities (Fig. 6.4). Both methods of modification were verified to yield vesicles in which biotin was accessible for binding to Texas Red-labeled NeutrAvidin (NAv). Given the reduced number of steps required to produce biotin-functionalized vesicles from pre-modified polymer, this route was employed to prepare vesicles in subsequent patterning studies.

**Spatially Patterning Arrays of Functionalized Vesicles and an Application**

We next set out to spatially organize biotin-modified polymersomes via immobilization onto NAv-printed surfaces. Microcontact printing is a powerful tool for the precise and complex spatial organization of adhesive ligands on surfaces (23-24). Substrates for polymersome array generation were prepared by microcontact printing NAv onto poly(dimethylsiloxane) (PDMS)-spin coated glass coverslips. NAv islands were 50 µm in diameter with 100 µm spacing (Fig. 6.5 A1). Unstamped regions of the substrate were blocked with the triblock copolymer Pluronic F-127 (PEO$_{106-b}$-PPO$_{106-b}$-
Figure 6.3 Functionalizing polymersomes via biotinylation. (A) The diblock copolymer used to prepare polymersomes is modified prior to vesicle formation to contain a reactive biotin group. (B) Polymer membranes are modified to contain biocytin post-vesicle formation via an EDC-mediated coupling to carboxyl-modified polymer. Polymersomes prepared through either route contain available biotin groups on the membrane surface that bind Texas-Red labeled-NeutrAvidin upon incubation (fluorescent (left) and phase images (right) of a representative vesicle functionalized with NeutrAvidin). (C) Control polymersomes prepared with a polymer that does not contain a reactive group do not bind avidin. Scale bar is 25 µm.
Figure 6.4 NeutrAvidin (NAv) functionalization of polymersomes made from biotin-modified polymer. For functionalization studies, polymersomes were prepared with Pluronic F-127 as the stabilizer to ensure carboxy-linked biocytin modification occurs with the carboxy group on the polymer and not on any residual surfactant that remains in the membrane. (A) Polymersomes that are PEO terminated are not functionalized with NAv. The lack of carboxyl groups on the vesicle surface ensures that EDC mediated reactions to biocytin do not result in biocytin linkage to the polymer membrane. (B) Polymersomes can be made by using polymer that already contains biotin. In this case biotin is available on both the inner and outer leaflets of the membrane and is able to bind NAv upon incubation. (C) Carboxy-terminated polymersomes are also functionalized with biocytin after vesicle formation. In this case, biotin is only added to the outer leaflet of the vesicle. This EDC-mediated coupling could be used to link other amine-containing proteins or molecules to a polymersome surface. Scale bar is 20 µm. This figure was presented in the supplementary text of the original manuscript.
PEO$_{106}$), which results in the presentation of PEO groups on bare PDMS not occupied by adhesive ligand. Vesicles that were ~55 µm in diameter were incubated on the substrate. Gentle movement of the microscope stage created a convective flow of PBS across the substrate face inducing non-adherent vesicles to move. Moving vesicles were either captured by printed NAv islands (Fig. 6.5 B1 and Fig. 6.6) or glided along the PEO blocked regions between islands. The high mobility of vesicles on F-127 blocked PDMS (Movie S1) is attributed to the steric repulsion between PEO chains at the vesicle-substrate interface.

Selective biological adhesion requires a combination of adhesive and repulsive interactions. In the absence of Pluronic blocking, vesicles failed to specifically pattern, adhering to both bare PDMS and NAv islands (Fig. 6.5 C1). To explore the role of biotin-avidin specificity on patterning, non-biotinylated vesicles were incubated with NAv printed surfaces. To our surprise, non-biotinylated polymersomes could still be patterned (Fig. 6.5 D1 and Fig. 6.7). The repulsive interactions between the blocking F-127 and the PEO chains on the polymersome drove vesicles onto NAv islands, regions of the substrate that minimized the energetically unfavorable repulsive forces between PEO groups. Capture of non-biotinylated polymersomes on NAv islands suggests a level of favorable non-specific interaction (25) between PEO and NAv which is verified by the absence of vesicle motion on continuous fields of the ligand (Fig. 6.8 and Movie S2).

We hypothesized that non-specific interactions between vesicles and NAv patches could be tuned by changing the surface area over which they occur. To test this hypothesis, NAv was printed with a decreased island size (Fig. 6.5 A2). On smaller islands biotinylated vesicles were again specifically patterned when substrates were
Figure 6.5 Patterning single polymersomes. Giant polymersomes that are functionalized with biotin are patterned in an array by incubation and immobilization onto micropatterned islands of NAv. (A) Fluorescence image of the NAv microcontact-printed array which has islands with a (1) 50 µm diameter and 100 µm spacing or a (2) 10 µm diameter and 50 µm spacing. Scale bar is 100 µm. (B–D) Fluorescence microscopy image of the NAv stamp overlayed with a phase image of polymersomes. (B) Biotinylated polymersomes incubated with a NAv stamped and F-127 blocked surface are specifically patterned. (C) Biotinylated vesicles fail to pattern, binding nonspecifically to bare PDMS. (D) Non-biotinylated control vesicles pattern on a printed and blocked surface provided the island size is sufficiently large. When present, colored tracks indicate the trajectories of mobile vesicles on stamped substrates (B–D).
Figure 6.6 Polymersome capture on NAv-printed surfaces. (A) Polymersomes are patterned by incubating vesicles on a NAv-printed surface and placing both the biotinylated polymersomes and the NAv surface on a microscope stage. Low-level motion of the microscope stage that is rotating between different imaging positions creates a convective flow in the polymersome sample. (B) Polymersomes moving along the NAv printed surface are mobile on the blocked regions that contain Pluronic F-127, but are captured by the NAv islands. Overtime, the capture of biotinylated vesicles and movement of non-captured vesicles out of the field of view results in the patterning of polymersomes. Trajectories for mobile vesicles appear in colored tracks that are overlayed onto the merged image of vesicles and the NAv stamp. Vesicles numbered 2, 3, 4, 6, 7, 9, 10, and 11 are captured by NAv islands. Vesicles 5 and 8 are trapped by two patterned vesicles and unable to move to find a NAv island or exit the field of view. Increasing the spacing between NAv islands would allow unbound vesicles to egress more readily. The possibility of utilizing geometric confinement as a patterning force, however, is compelling given this observation of entrapment. Scale bar is 100 µm. The full time course depicting this vesicle capture can be seen in Movie S1. This figure was presented in the supplementary text of the original manuscript.
Figure 6.7 Polymersomes patterned on NAv islands of 50 µm diameter and 100 µm pitch. Sustained vesicle patterning in the non-biotinylated case motivated our hypothesis that another driving force such as the repulsive interaction between PEO chains on the vesicles and PEO chains on the blocked substrate was at play. This figure was presented in the supplementary text of the original manuscript.
Figure 6.8 Effect of NAv surface area on non-specific binding. To illustrate the effect of available protein surface area on non-specific binding, NAv was printed in a uniform field on the left side of the PDMS substrate and printed in islands on the right side. The fluorescent image of NAv (red) is overlayed with the phase image of polymersomes. The substrate was blocked with Pluronic F-127 and incubated with non-biotinylated polymersomes. Though polymersomes do not contain biotin, they are immobilized through non-specific interactions on the uniform field of NAv. When the area of interaction is decreased to a 10 µm island size, however, the vesicles are mobile and move freely across the substrate. The tracks of motile vesicles are indicated by overlaying the colored vesicle trajectories onto the merged image of the polymersomes and NAv stamp. The full time course of this phenomenon can be seen in Movie S2. This figure was presented in the supplementary text of the original manuscript.
blocked (Fig. 6.5 B2), and bound non-specifically to bare PDMS when left unblocked (Fig. 6.5 C2). When non-biotinylated control vesicles were incubated with the smaller islands of NAv, however, they failed to pattern as previously observed on large 50 µm islands. Instead, these vesicles were found to be continuously motile during observation as indicated by the superimposed trajectories (Fig. 6.5 D2 and Movie S2). By decreasing the island size we effectively eliminated the contribution of nonspecific PEO-NAv interaction allowing us to attribute the high fidelity patterning of biotinylated vesicles to biotin-avidin binding exclusively (Fig. 6.9). Ultimately, we have shown that NAv-printed PDMS, blocked with Pluronic F-127 is ideally suited for the spatial patterning of giant biotinylated polymersomes.

Having successfully patterned giant microfluidic vesicles we sought to demonstrate the array’s future applicability to the design of systems capable of inter-vesicle communication. Towards this end, we demonstrate the vesicle array can be used as a biosensing platform. Immobilized polymersomes can report the presence of a soluble molecule added to the vesicle array by capturing the molecule at the vesicle membrane. Biotective Green reagent, an avidin analogue, was used as the bioactive ligand. This molecule is labeled with a fluorescent donor molecule that is quenched through FRET interactions with an acceptor molecule located in the biotin-binding pockets of the reagent. Upon binding biotin, the quencher molecules become displaced and the signaling ligand fluoresces. When this reagent was added to an array of immobilized biotinylated polymersomes, the ligand was captured at the polymersome surface (Fig. 6.10 B). Fluorescent signals from three representative vesicles over the course of 40 minutes are shown in Fig. 6.10 C.
Figure 6.9 Polymersomes patterned on smaller NA
v islands of 10 μm diameter and 50 μm pitch. Unlike previously, where non-biotynlated vesicles remained patterned, here, reduction in island size prevents patterning of these same control vesicles. That biotynlated vesicles retain their pattern during stage motion suggests the biotin-avidin interaction is stronger than the non-specific PEO-NA
v interaction. This figure was presented in the supplementary text of the original manuscript.
Figure 6.10 Creating sensor arrays. Biotin functionalized vesicles were patterned onto an array of NAv with islands 50 µm in diameter and 100 µm in spacing. (A) At time = 0, when Biotective Green Reagent is added to the system, the vesicles do not fluoresce. (B) At time = 42 min, Biotective Green Reagent bound biotin on the surface of vesicles, the fluorescence signal from the reagent increased and occurred selectively at the vesicle surface. (C) Fluorescence signals from the reagent were tracked at the surface of three vesicles (numbers correspond to panel A) over the course of a 40 min imaging session.
Conclusions

Our system provides a significant advance in the engineering of vesicle-based assemblies. There has been no previous demonstration of the ability to chemically control the spatial organization of single giant polymersomes. We show, by patterning vesicles of precisely controlled diameter and payload encapsulation, that individual polymersomes can be positioned into multi-vesicle arrays that are geometrically governed by the underlying adhesiveness of the surface. In the future, the printing of multiple adhesive ligands (18, 26) or oligonucleotides (27) on a single substrate will enable the patterning of multiple populations of vesicles, each with a distinct biorecognition capability. The precise patterning of giant functionalized polymersomes is an important step towards realizing the full potential of increasingly complex artificial cell systems.

Acknowledgements

This work was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering under award DE-FG02-11ER46810. NPK and SJH are supported by NSF Graduate Fellowships. We thank Eric Johnston for assistance in preparing photomasks.
References


Chapter 7

Future Directions

Visualizing the Cytoskeleton

The cell analyses contained in the previous chapters largely considered effects of various small molecule perturbations (i.e. chemoattractants, cytoskeletal inhibitors, and antibodies) on biological metrics at the whole cell length scale. We were able to infer the effect of these perturbations by quantifying how the cell’s response under treatment differed from the equivalent control cases with respect to metrics of cell shape and motility. However, there is a wealth of information to be gained by visualizing the organization of the cytoskeleton and its rearrangement under the previously explored molecular perturbations. In practice, pilot efforts at fixing neutrophils in the keratocyte-like phenotype have revealed sensitivity in the post-fixation shape to the fixation method. Ubiquitous fixation methodologies such as 10 % neutral buffered formalin and 4 % paraformaldehyde (PFA) (Fig. 7.1 A) were found to abrogate features like the keratocyte-like crescent shape and ruffled lamellipodium visible in live neutrophils (Fig. 7.1 C). With respect to cell area, PFA treatment resulted in a 40 % decrease in total cell area and 60 % increase in nuclear area as compared to live cells in the keratocyte-like phenotype (Fig. 7.1 D).

We previously demonstrated (1) that neutrophils employ the promiscuous integrin receptor MAC-1 (αMβ2) to support their haptokinetic motility on fibronectin (FN) fields
Figure 7.1 Loss of phenotype after paraformaldehyde fixation. (A) Phase contrast image of neutrophils fixed under 4% paraformaldehyde and permeabilized with 0.5% TritonX-100. (B) Corresponding fluorescence image of same neutrophils after vinculin staining. (C) Live phase contrast image of keratocyte-like neutrophil. (D) Quantification of paraformaldehyde (PFA) fixation-induced morphological changes. Compared to live cells the total cell area decreased by 40% whereas the nuclear area increased by 60%. Error bars are ± standard deviation from <n> = 11 cells/condition. Scalebars are 10 µm.
Raptis and coworkers have established that clustering of the $\beta_2$ chain (CD11b) is critical in the execution of terminal effector functions like proteolytic enzyme secretion and reactive oxygen intermediate production (2). In mesenchymal cells integrin clustering is a precursor to the formation of three dimensional adhesive plaques called focal adhesions. These adhesion units are composed of numerous protein scaffolds which individually or cumulatively achieve a mechanosenstive linkage between the extracellular integrins and intracellular cytoskeleton (3). One marker of the formation of these focal adhesions is vinculin (4). When neutrophils exhibiting the keratocyte-like phenotype were fixed under 4% PFA and permeablized under 0.5% TritonX-100, vinculin staining resulted in a uniform signal across the cell body (Fig. 7.1 B) with no evidence of discrete adhesive plaques.

However, when an alternative fixation strategy was employed, developed specifically to stabilize microtubules, we found improved phenotype preservation (Fig. 7.2). This alternative fixation strategy using microtubule stabilizing buffer (MTSB) was recommended to us by Ravi A. Desai, PhD from his first hand experience with locomoting NRK-52E cells (a rat kidney cell line available from ATCC) (5). NRK-52E cells bear a striking resemblance to the shape of fish keratocytes (6-7) and our keratocyte-like neutrophils (1) having a very broad leading edge lamellipodium. In contrast to the diffuse vinculin signal of PFA-fixed keratocyte-like neutrophils, both amoeboid and keratocyte-like neutrophils fixed with MTSB and stained for vinculin, revealed distinct plaques at the periphery of the cell (red double-headed arrows of Fig. 7.2). In the amoeboid case, the vinculin structures were predominately located in the rear and sides of
Figure 7.2 Improved phenotype stability and vinculin plaque detection with microtubule stabilizing buffer fixation. Vinculin staining of microtubule stabilizing buffer fixed neutrophils exhibiting: (A) amoeboid phenotype, (B) keratocyte-like phenotype (one cell), and (C) keratocyte-like phenotype (two cells). Red double headed arrows denote vinculin plaques not observed under PFA fixation and TritonX-100 permeabilization of Fig. 7.1 B. Scalebars = 10 μm.
the cell (Fig. 7.2 A) while they uniformly decorated the periphery in the keratocyte-like morphology (Fig. 7.2 B and C).

Previously, Hammer Lab achieved traction measurements of neutrophils chemotaxing in the amoeboid style and observed that the largest force generation was spatially located in the rearward uropod (8). This asymmetric contractility was interpreted to be the basis of the cell’s ability to propel its cytoplasm forward in the absence of extensive adhesive contact with the underlying substrate. The asymmetric vinculin plaques seen in the rearward uropod of the amoeboid neutrophil (Fig. 7.2 A) are consistent with the highest traction generation also being observed in the rear of the cell (8). That vinculin plaques uniformly decorate the periphery of keratocyte-like neutrophils is also a possible explanation as to why the cells are slower and more directionally persistent than their amoeboid counterparts (Figs. 3.9, 4.6, and 4.7).

It is interesting to speculate as to whether or not the improved resolution of adhesive plaques in neutrophils via MTSB fixation implies that microtubules (MT) are critical to phenotype preservation in living neutrophils. In the zebrafish model of leukocyte migration, the MT organizing center is positioned between the leading edge of the migrating cell and the nucleus (9). Furthermore, perturbation of MT polymerization and depolymerization kinetics was found to impact neutrophil homing and motility. Exploring MT dynamics in the context of the keratocyte-like morphology would be novel.

Also worth considering is the nature of the nonlinear change in cell area as compared to nuclear area under PFA fixation (Fig. 7.1 D). Under PFA conditions, cell area was found to decrease by 40 % while nuclear area was found to increase by 60 %.
The reduction in cell area could be rationalized on the basis of TritonX-100 generating membrane pores and causing cell swelling. A volume increase could have the apparent affect of a reduced projected area. However in this swelling model less nuclear compression would be anticipated and it's known that TritonX-100 also punctures the nuclear envelope (10). Thus, why reduced compression and nuclear swelling would result in an increase in projected nuclear area is unclear.

Alternatively, the nonlinearity could be a consequence of induced pores in the cytoplasmic and nuclear envelopes resulting in differing mechanical properties of the two components. TritonX-100 could be targeting a lipid component differentially expressed in the two envelopes. An increase in nuclear spread area could be interpreted as the envelope becoming more mechanically flaccid after pore formation. The mechanics of the cellular nucleus and its molecular basis represent a large and robust field of study (11). Within this field, human neutrophils are a particularly interesting subset of cells to study owing to their characteristic tri-lobed nuclei. It has been observed that lobulated nuclei have reduced lamin A/C content compared to rounded nuclei (12) and that lamin concentration and composition control nuclear stiffness.

**Fixation and Vinculin Staining Method Notes**

Neutrophils were plated on fields of FN in the usual manner (see Materials and Methods Chapters 3 and 4). For PFA fixation cells were incubated in a final concentration of 4% PFA in phosphate buffered saline (PBS) from a freshly opened stock bottle of 16% electron microscopy grade methanol-free PFA (Electron Microscopy Sciences, #15710) for 10 min at room temperature (RT). Cells were rinsed 3X in PBS and permeabilized via 10 min incubation under 0.5% TritonX-100 solution (MP
Biomedicals, #807423). Cells were rinsed 3X in PBS and blocked with 5% bovine serum albumin (BSA) (Sigma, #A70030-100G) solution for 1 hr at RT (13).

For improved phenotypic preservation via MTSB fixation, a stock of 10X MTSB was prepared in advance. The final working concentration of MTSB consisted of: 0.1 M PIPES at pH 6.75, 1 mM EGTA, 1 mM MgSO₄, 4 % (w/v) Poly(ethylene glycol) 8000, 1 % TritonX-100, and 2 % Paraformaldehyde. Cells were incubated for 10 min at 37 °C under MTSB and rinsed 3X in PBS. Very gentle rinsing was performed to avoid shearing cell membranes. After rinsing, cells were incubated in 2 % BSA in PBS for 1 hr at RT. Cells were rinsed 3X in PBS and stained for vinculin.

Vinculin staining was a two step immunocytochemical preparation. Fixed and permeabilized cells were incubated at a 1:200 volume dilution of stock mouse mAb to vinculin (hVIN-1, Abcam, # ab11194) for 1 hr at RT. Cells were subsequently rinsed 3X with PBS and incubated in a 1:400 volume dilution of stock AlexaFluor488 goat-anti-mouse IgG1 (Invitrogen, #A-11001) for 1 hr at RT. Finally, cells were rinsed 3X with PBS and imaged.

**Neutrophil Motility on mPADs**

An aim not fully realized in this thesis was to elicit neutrophil motility on mPADs and measure the corresponding traction maps. Previously, Brendon Ricart in the Hammer laboratory measured traction maps of dendritic cells (DCs) chemotaxing across large diameter, soft post arrays (14) (post specifications are recorded in Fig. 7.3, “Large, Soft”). The Ricart experiments were an impressive combination of device engineering and biological insight and we initially attempted to simply substitute human neutrophils for DCs in his experimental setup. However, we found that his post geometry failed to
Figure 7.3 **Summary of mPADs specifications.** Error bars are ± standard deviations from scanning electron micrograph measurements.

<table>
<thead>
<tr>
<th></th>
<th>Large, Soft</th>
<th>Small, Soft</th>
<th>Small, Stiff</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen Lab Master Designation</td>
<td>1.83 μm, #12</td>
<td>0.8 μm, #5</td>
<td>0.8 μm, #2</td>
<td></td>
</tr>
<tr>
<td>Post Diameter (μm)</td>
<td>1.83</td>
<td>0.604 ± 0.031</td>
<td>0.635 ± 0.016</td>
<td></td>
</tr>
<tr>
<td>Post Length (μm)</td>
<td>12.9</td>
<td>5.576 ± 0.286</td>
<td>2.116 ± 0.051</td>
<td></td>
</tr>
<tr>
<td>Post Aspect Ratio (L/D)</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Array Pitch (μm)</td>
<td>5</td>
<td>1.932 ± 0.002</td>
<td>1.966 ± 0.003</td>
<td></td>
</tr>
<tr>
<td>$A_{wet}/A_{continuous} \times 100$ (%)</td>
<td>12</td>
<td>18</td>
<td>19</td>
<td>$A_{wet/total} = SA_{wet/total} = 2 \pi r_{p}^2$</td>
</tr>
<tr>
<td>$k_{spring}$ (pNnm)</td>
<td>1.92</td>
<td>0.28 ± 0.09</td>
<td>6.32 ± 0.031</td>
<td>Yang et al. 2011. Nat Protoc</td>
</tr>
<tr>
<td>Warp Correction</td>
<td>0.90</td>
<td>0.93</td>
<td>0.79</td>
<td>Schoen et al. 2010. Nano Letters</td>
</tr>
<tr>
<td>$k_{spring, shear}$ (pNnm)</td>
<td>1.73</td>
<td>0.26</td>
<td>4.97</td>
<td></td>
</tr>
<tr>
<td>G (pure shear) (kPa)</td>
<td>9</td>
<td>5</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>$E_{eff}$ (kPa)</td>
<td>1.5</td>
<td>0.7</td>
<td>14</td>
<td>Ghiasuddin et al. 2008. Soft Matter</td>
</tr>
<tr>
<td>$G_{eff}$ (kPa)</td>
<td>0.5</td>
<td>0.2</td>
<td>5</td>
<td>$v_{Geff} &gt; 0.5$</td>
</tr>
<tr>
<td>Used In</td>
<td>Ricart et al. 2011 (Submitted to Biophys J.)</td>
<td>Chapter 5 Biophys J.</td>
<td>Chapter 7</td>
<td></td>
</tr>
</tbody>
</table>
elicit neutrophil spreading (Fig. 7.4 A) as was previously observed in DCs. The FN coverage of Ricart post arrays relative to a continuous field was only 12 %. In Chapter 4 we previously demonstrated that neutrophils perceive island geometries with 20 % coverage relative to a continuous field as if the ensemble of islands were a continuous field. In that context we can now infer that there is a critical protein coverage threshold that resides between 12 % and 20 %, below which neutrophils no longer perceive islands as continuous. This is highly consistent with the findings of Lehnert and coworkers in mesenchymal cells where protein coverage less than 20 % dramatically reduced cell spreading (15).

However, when small diameter mPADs with protein coverage of 20% were fabricated, neutrophils were induced to spread (Fig. 7.4 B and C) as was established previously in Chapter 5 (Fig. 5.1 B). While increasing protein coverage recovered spreading, the stiffness of the posts became a critical factor in eliciting motility. On small, soft posts neutrophils spread but did not translocate (Fig. 7.4 B). Neutrophil motility in the region of post collapse adjacent to the posts served as a useful control in this FOV. When the posts were shortened to increase rigidity, a fraction of neutrophils (~ 25 % of total cells plated) were motile (Fig. 7.4 C).

There are a variety of stiffness definitions to describe the discretized environment a cell experiences on mPADs. Figure 7.3 summarizes the specifications of each post array discussed in this chapter as well as a number of stiffness metrics. On the simplest level, each pillar can be modeled as a cantilever subjected to a load at its unconstrained terminus (16) in which case the material spring constant (k_{spring}) is a natural description of pillar stiffness. Small, soft and stiff post spring constants were ~ 0.3 pN/nm and ~ 6
Figure 7.4 Neutrophil motility on post arrays is stiffness dependent. (A) The large, soft posts used to study dendritic cell traction forces during chemotaxis (Ricart et al. 2011. *Biophys J*), fail to elicit spreading in human neutrophils. Neutrophils assume a hexagonal phenotype of one unit cell but do not spread (see inset). Neutrophils on the continuous field of fibronectin in the same FOV are motile. (B) On small, soft posts used to study neutrophil spreading forces (Chapter 5) neutrophils spread but are not motile. A region of collapse after printing supports neutrophil motility in the same FOV as the non-motile cells residing on the posts. (C) On small, stiff posts a fraction of neutrophils are motile. Schematics are to scale and correspond to each experimental condition above. Micrograph scalebars = 50 µm. Inset scalebar = 10 µm. Schematic scalebar = 5 µm.
pN/nm, respectively. Theoretical work done by Schoen and coworkers established a series of corrections to these spring constants as a function of post aspect ratio. The corrections account for the contribution of pillar tilting and base warping to the measured free terminus deflection (17). The magnitude of the correction is minimal for the small, soft posts (7 % reduction) and large for the small, stiff posts (21 % reduction), but the relative ten-fold difference in stiffness between substrates is retained. While the spring constant is a natural description of single pillar stiffness, it is ambiguous with respect to the macroscope or ensemble (i.e. multi-post) stiffness perceived by a cell. In a macroscopic context it is more natural to describe the arrays in terms of Young’s moduli (E) or shear moduli (G). Applying a simple definition of pure shear we find \( G_{\text{small, soft}} \sim 5 \text{kPa} \) and \( G_{\text{small, stiff}} \sim 42 \text{kPa} \). Both of these values are within the physiologically relevant domain of stiffnesses (18). The appropriateness of a pure shear model for neutrophil engagement of mPADs is an assumption, but one supported by the empirical work of Lemmon and coworkers which demonstrated that shear is a larger contribution to post deflection than torque (19). Alternatively, Ladoux and coworkers developed a theoretical description of effective array stiffness by solution of the Green’s function for a discretized substrate (under certain governing assumptions) (20). The Ladoux model estimates the Young’s moduli of our post arrays as \( E_{\text{small, soft}} \sim 0.7 \text{kPa} \) and \( E_{\text{small, stiff}} \sim 14 \text{kPa} \), substantially softer than anticipated by the local pure shear model. While different definitions of stiffness clearly yield different values, the transcendent point is that the small, soft and small, stiff arrays used in these analyses differ by an order of magnitude.

Increasing post stiffness was necessary to elicit neutrophil motility, but can the tractions (i.e. substrate deformations) still be resolved? On soft but not stiff arrays
neutrophil-induced pillar deflections were readily observed at 100X magnification (Fig. 7.5 B). To determine if neutrophil-induced pillar deflections on stiff arrays were small but nonzero we performed a complete traction analysis of both conditions. The displacement heat maps for all posts beneath and surrounding the three cells in each FOV reveal comparable background noise (i.e. deflections observed in posts surrounding but not under the cells). Quantifying the mean apparent deflection of background posts around the cells we find comparable noise floors on the order of 20-40 nm. Applying the respective material spring constants to these calculated deflection floors resulted in substantially different force floors (Fig. 7.5 D). The mean force per pillar of neutrophil tractions on soft arrays ($<F_{cell/post}> \sim 123$ pN) was 10.3X in excess of the 12 pN force floor, whereas on stiff arrays the mean force per pillar ($<F_{cell/post}> \sim 182$ pN) was only 1.2X in excess of the 150 pN force floor. Hence, the stiff arrays resulted in a substantially reduced signal-to-noise ratio. While our confidence in the measured forces attributed to neutrophils on stiff arrays is low, it was interesting to compute the average per-post strain energy imparted by the cells on each of the arrays (Fig. 7.5 E). Strain energy was not constant but an order of magnitude less on stiff arrays than soft arrays. This suggests a non-linear response in neutrophil traction generation on stiff substrates. If we make the assumption that total cell energy expenditure is constant, the reduction in strain energy on stiff substrates could be interpreted as the cell shunting more energy to other processes such as motility. Indeed, it was only on the stiff post arrays that neutrophil motility was observed, suggesting an inverse relationship between cell speed and contractility.

The pragmatic implication of this section is that the small but stiff arrays used in these traction measurements are slightly stiffer than ideal. Unfortunately the silicon
Figure 7.5 Neutrophil traction generation is stiffness dependent. Caption over.
Figure 7.5 Neutrophil traction generation is stiffness dependent (continued). Neutrophils on small, soft pillars (left column data, corresponding to 7.3 B condition) or small, stiff pillars (right column data, corresponding to 7.3 C condition). (A) Phase contrast. (B) Fluorescence channel. (C) Per-post displacements in rectangular ROI around cell. (D) Per-post forces of all posts within cell contact area. Force floor was calculated by applying spring constant to mean background displacement of B. (E) Per-post strain energy of all posts within cell contact area. Scalebars = 10 µm. Error bars are ± standard deviation from means of n = 3 cells.
masters supplied to Hammer laboratory by Chen laboratory do not presently have an intermediate stiffness between the two arrays tested. Therefore fabrication of a new master is necessary or off-ratio (i.e. not 10:1 base:cure (w/w)) PDMS formulations will be required. While the later avenue is certainly more facile, the reproducibility of off-ratio PDMS formulations with respect to cured stiffness is of concern. However, off-ratio PDMS formulations have been used in mechanically-sensitive applications with success by Huh laboratory (21).

**Additional Small Molecule Inhibitor Work**

The following comments were informed by discussions with and recommendations by Professor Christopher S. Chen. In Chapter 5 we explored the mechanism by which a neutrophil transitions from a quiescent sphere to a spread and adherent phenotype. Our small molecule inhibitor work identified that spreading was not analogous to lamellipodium formation (no effect with CK666 treatment) but was sensitive to perturbations of actin kinetics which alter actin cortex mechanics (jasplakinolide stiffened the cortex whereas cytocholasin B softened the cortex). These results were previously summarized in Figure 5.7. Our work suggested that neutrophil ligation of FN on the pillar tips induced cell remodeling of the cytoskeleton that resulted in a reduction of cortical stiffness. We previously showed neutrophil haptokinetic engagement of FN was integrin mediated but an outstanding question that remains is the nature of the integrin-actin cortex linkage. Molecular targets thought to be critical in the integrin-actin cortex linkage include focal adhesion kinase (FAK) (22), Src (23), and the TRPV calcium channel family (24). Small molecule inhibitors are commercially
available against these targets and are fast acting, which is important as the timescale of
neutrophil spreading is on the order of seconds.

Vesicle Haptokinesis

The original motivation for pursuing the vesicle patterning work of Chapter 6 was
to develop an experimental platform for the study of autonomous vesicle motion.
Computational modeling done in Professor Anna C. Balazs’s group predicted the
coordinated motion of a system of semi-permeable particles (or capsules) releasing haptic
ligands into their environment (25). The release of haptic ligands generated gradients of
local adhesivity which, coupled with hydrodynamic entrainment, resulted in streams of
particles moving autonomously in two dimensions.

As an intermediate step en route to empirically realizing this ambitious signaling
system, we sought to elicit haptokinetic motion of the Chapter 6 surface-biotinylated
microfluidic vesicles by repeatedly and randomly printing small islands of NeutrAvidin
(Fig. 7.6 A). The hypothesis was that if the density of islands was sufficiently high the
vesicles, by virtue of stochastic formation and dissolution of receptor-ligand bonds, could
be induced to haptokinetically move across the surface. Consistent with our previous
findings (26) we were able to immobilize surface active micronscale vesicles on 10 μm
diameter adhesive islands. Unfortunately, our multi-print method failed to achieve an
adequate density of islands to robustly test our autonomous motion hypothesis. At the
time these pilot experiments were performed the square lattices of 10 μm diameter, 50
μm pitch were the smallest arrays available to us. However, as demonstrated in Chapter
4, we can now reliably achieve submicron arrays of hexagonally arranged islands with
0.9 μm diameter and 1.9 μm pitch via the stamp-off method of microcontact printing
Figure 7.6. Attempted vesicle haptokinesis experiment. (A) Multi-printing of 10 μm islands at 50 μm pitch yielded a few regions where islands were separated by 5 - 10 μm. No multi-island hopping was observed. Scalebar = 100 μm. (B) Expanded view of a single vesicle in A, proximal to 2-3 NeutrAvidin islands. Scalebar = 50 μm. (C) Geometric construction illustrating vesicle surface height above substrate. Islands must be tightly spaced to facilitate multi-island contact of a single vesicle. (D) Island arrays of Chapter 4 will place many more islands in proximity to vesicle surface than the attempt in C yielded.
Switching to Chapter 4 island arrays would achieve a substantial increase in island density beneath the biotinylated vesicles (Fig. 7.6 D)

An additional empirical parameter available for tuning is the rigidity of the vesicle membranes themselves. The double emulsion templating production method results in a homogeneous population of nearly spherical vesicles. To improve yield during production and solvent evaporation, the membranes are stabilized with surfactants such as BSA or Pluronic F-127. As a simple geometric construction of rigid spherical vesicles makes clear (Fig. 7.6 C), the contact interface over which vesicle-substrate, biotin-avidin, interactions can occur is quite limited as the surface of the vesicle rises quadratically over the horizontal plane of islands. Thus, while increasing island density will improve the number of islands available for ligation in the contact zone (Fig. 7.6 D), decreasing vesicle rigidity would increase the size of this contact interface as well. Reducing the surfactant content during production and solvent-evaporation is one avenue for consideration. However in our experience reductions in surfactant concentration usually result in poor vesicle survival during solvent evaporation.

An alternative strategy is to return to the thin film method of rehydration for vesicle generation (27) which results in a more flaccid, pancake-like architecture. The cost will be a substantial increase in polydispersity with respect to vesicle size that is a consequence of this production strategy. While not ideal, the Hammer laboratory has previously demonstrated that porphyrin incorporation in vesicle membranes generated by thin film rehydration renders the vesicles amenable to photo-inducible rupture (27). This controlled rupture mechanism could be ideal in generating local adhesive gradients more
akin to the Balazs computational model than static adhesive fields achieved by microcontact printing.
References


Appendix A

Custom MATLAB Code for Analysis of Neutrophil Motility

Introduction

The purpose of this appendix is to provide the reader with more detail regarding the data analysis workflow employed to compute neutrophil motility statistics. Broadly speaking the workflow consisted of capturing timelapse images of neutrophil migration (Fig. A.1 A), identifying the cells in each image and computing geometric centroids (Fig. A.1 B), linking centroids into trajectories (Fig. A.1 C) and computing metrics of population dispersion (Fig. A.1 D). The appended code is original, customized to accommodate the specific nuances of our neutrophil experimental data such as the file naming convention used on the microscope and the empirically determined segmentation parameters needed to identify cell bodies. However, the general workflow should be amenable to a variety of motility datasets provided the user tunes some of these empirical parameters.

Methodology

1. Time lapse images should be labeled with the following convention: “pXXttt.tif”. An example would be “p06037.tif” for a phase (“p”) from location “06” corresponding to time “037” seconds. This sequence of images should reside within a folder labeled “Loc_XX”. An example would be “Loc_06” containing all phase images from location “06”. Multiple Loc_XX folders can reside within the same directory and can be processed simultaneously.
Figure A.1 Neutrophil motility data analysis workflow. (A) Timelapse images of neutrophil migration are captured. (B) Image segmentation is performed to identify cells and geometric centroids are computed. (C) Centroids are linked into trajectories. (D) Population dispersion statistics are computed.
2. In MATLAB, run “Time_Segment_Driver_v7.m”. You must specify the directory containing “Loc_XX” folders at line 87.

   a. You will be asked if during acquisition you attempted constant time interval imaging. If so specify the attempted imaging period in seconds. This is to handle the real-world acquisition issue of the image not being snapped at an exact integer multiple of the desired imaging period. By specifying the attempted imaging period the program will round the actual acquisition time to the closest integer multiple of the desired acquisition time. This was necessary to improve mean squared displacement (MSD) computation by increasing the statistical power (observation frequency) of a particular $\tau$.

   b. “Time_Segment_Driver_v7.m” calls the following subroutines:

      i. “ReadFolderContents_v3.m”
      ii. “ReadTiffImages_v1.m”
      iii. “Abs_Time_v1.m”
      iv. “Sort_Time_v1.m”
      v. “Bin_Time_v4.m”
      vi. “Segmentation_v4.m”

3. The output from running “Time_Segment_Driver_v7.m” is a “Time_Segment” folder residing within the specified directory. The numeric prefix to the folder is the ISO 8601 dateform (“yyyyMMddTHHmmSS”) for the day and time analysis was performed. Within this “Time_Segment” folder resides “Loc_XX” folders each containing an “Overlay” and “Segmentation” folder. “Overlay” contains the original
data set superimposed with identified object boundaries and “Segmentation” contains the resulting binary image (cells denoted by ones on a field of zeros).

4. In MATLAB, run “CC_Driver_v5.m”. You must specify the directory containing “yyyymmddTHHMMSS_Time_Segment” folder at line 65.

a. “CC_Driver_v5.m” calls the following subroutines:

   i. “ReadFolderContents_v3.m”

   ii. “Track_Centroids_v5.m”

      1. “Pos_Selection_v2.m”

   iii. “IJ_Manual_Track_Prep_v3.m”

5. The output from running “CC_Driver_v5.m” is a “Loc_XX.mat” and “Loc_XX.txt” file within each “Loc_XX” folder of “yyyymmddTHHMMSS_Time_Segment”. These files contain the cell trajectories in a format compatible with the ImageJ Manual Tracking plugin.

a. Open the image sequence in “Overlay” folder of “Loc_XX” as a stack in ImageJ

b. Initialize the Manual Tracking plugin

c. Select “Load Previous Track File” and navigate to the “Loc_XX.txt” file output from “CC_Driver_v5.m”

d. Click on “Show text?” option in Manual Tracking plugin

e. Click “Overlay Dots & Lines button in Manual Tracking plugin

f. Inspect resulting superposition of tracks and data

6. If necessary employ “CC_Output_Editor_v2.m” to make manual exclusions of all or portions of trajectories deemed anomalous. At line 56 hardcode path to “yyyymmddTHHMMSS_Time_Segment”. At line 82 hardcode tracks to be entirely
deleted. At line 90 hardcode portions of tracks to be retained (exclude portions outside these bounds). At line 98 hardcode portions of tracks to be eliminated (retain portions outside these bounds).

7. The output from running “CC_Output_Editor_v2.m” are “Loc_XX_edited.mat” and “Loc_XX_edited.txt” files within the “Loc_XX” folder of “yyyyymmddTHHMMSS_Time_Segment”.

8. Manually copy all “Loc_XX_edited.mat” files up into the parent “yyyyymmddTHHMMSS_Time_Segment” directory. If you did not run “CC_Output_Editor_v2.m” then copy the original “Loc_XX.mat” files. If you ran “CC_Output_Editor_v2.m” twice then copy the “Loc_XX_edited_edited.mat” files or any combination thereof.


10. The output from running “Merge_Mats_v4.m” is a “Merged_Data.mat” file. The numeric prefix to the folder is the ISO 8601 dateform (“yyyyymmddTHHMMSS”) for the day and time analysis was performed. This function simply performs a concatenation of the individual location-specific data sets and renames tracks so each trajectory has a unique identification number.

11. Copy all “yyyyymmddTHHMMSS_Merged_Data.mat” files to the same location. Manually rename files so they conform to the following naming convention: “DXX_yyyyymmdd_yyyyymmddTHHMMSS_Merged_Data_XXXpXX_FN_XXXpXX_fMLP.mat”. An example would be:
“D01_20111014_20150216T123011_Merged_Data_005p00_FN_010p00_fMLP.mat” which is the data from donor “D01” acquired on date “20111014”, Merged_Mats_v4.m was performed on “20150216T123011”, and experimental conditions were “005.00” μg/mL FN and “010.00” nM fMLP.

12. In MATLAB, run “Supra_MSD_Driver_v5.m”. This requires you to select if you are analyzing one or multiple experimental conditions. Navigate to the folder containing .mat file(s) and select the file(s) of interest. Enter a pixel to micron conversion in units of microns per single pixel. Elect to analyze full empirical data or a portion of the empirical data. If analyzing a temporal portion of the empirical data specify the upper time limit in minutes beyond which trajectory data will not be used. Elect whether or not to supply an error estimate (ε) in the position of cell centroids due to camera noise during acquisition. The correction performed on the empirical MSD curve is a subtraction of 4*ε² from the MSD at all τ values. This ε value is experimentally determined via a modeling experiment in which the error in the final MSD as a function of camera noise is established by superimposing additional random noise equal to the camera noise on the centroids of each tracked cell and determining the effect on the final MSD curve. In all of my experiments a correction of ε = 0.4604 pix was employed. Lastly, specify if persistent random walk and power-law model fitting should be performed on the full empirical MSD or a portion thereof.

a. “Supra_MSD_Driver_v5.m” calls the following subroutines:
   i. “MSD_Driver_v15.m”
      1. “Parse_Filename_v2.m”
2. “Post_IJ_Manual_Track_v3.m”
3. “Cell_Track_Plotter_v6.m”
4. “Consec_Differentials_v4.m”
5. “Step_Size_Stationarity_v2.m”
6. “Histograms_v3.m”
7. “Path_Length_v6.m”
8. “Mean_Path_Length_v5.m”
9. “Area_v4.m”
10. “Mean_Area_v5.m”
11. “Filter_Exp_Data_v3.m”
12. “Differentials_v5.m”
13. “Neff_v1.m”
14. “Mean_Differentials_v6.m”
15. “MSD_Epsilon_Subtract_v3.m”
16. “Plot_Mean_Differentials_v5.m”
17. “Filter_Mean_Differentials_v4.m”
18. “SandP_v11.m”
19. “Power_Law_v4.m”
20. “Plot_SandP_Fit_v6.m”
21. “Plot_Power_Law_Fit_v4.m”
22. “Van_Hove_Analysis_v3.m”
23. “Tidy_Up_v1.m”
13. The output from running “Supra_MSD_Driver_v5.m” is a “MSD_Driver_v15” folder. The numeric prefix to the folder is the ISO 8601 dateform (“yyyyymmddTHHMSS”) for the day and time MSD analysis was performed. Each folder within the “yyyyymmddTHHMSS_MSD_Driver_v14” directory contains the dispersion analysis (figures, arrays, and log files) corresponding to a “Merged_Data.mat” file. An Excel spreadsheet logs the final dispersion metrics for each condition analyzed.

**Code**

Note: Missing lines are version history annotation, removed for space considerations.

**Time_Segment_Driver_v7.m**

1 % Steven Henry
2 % 02/16/2015
41 %**************************************************************************
42 % PURPOSE:
43 % From all image files in a series of Loc_XX folders within a given
44 % experimental condition the aims are to:
45 %
46 % Aim 1: Extract the time stamps from filenames and, if applicable, round
47 % these time stamps to the nearest integer multiple of the user-attempted
48 % constant interval imaging
49 %
50 % Aim 2: Perform image segmentation to identify cell boundaries. This goal
51 % is achieved via adaptation of MATLAB's:
52 % "Detecting a Cell Using Image Segmentation" demo found at:
54 % shipping/images/ipexcell.html> and
55 % "Batch Processing Image Files in Parallel" demo found at:
57 % shipping/images/ipexbatch.html>
58 %
59 % ASSUMPTIONS:
60 % (1) User has reserved "Loc_XX" naming to only those folders within a
61 % particular experimental condition to be analyzed.
62 % (2) User does not have extraneous ".tif" images or stacks present within
63 % a given "Loc_XX" folder.
64 % (3) Only "tif" images of a particular type (e.g. phase or fluor) reside
65 % within a particular "Loc_XX" folder. The present filtering logic used in
66 % "ReadTiffImages_v1.m" does not differentiate phase images from fluor
67 % images.
68 % (4) Raw image files have time embedded in seconds and take the form
69 % "pXXttt....ttt.tif" or "sXXttt....ttt.tif" where "p/s" denotes "phase" or
70 % "fluor", "XX" denotes the location corresponding to "Loc_XX", "ttt....ttt"
71 % denotes some time stamp in seconds of non-constant length
72 %
73 % FUNCTIONS CALLED:
74 % ReadFolderContents_v3.m
75 % ReadTiffImages_v1.m
76 % Abs_Time_v1.m
clc
clear all
close all

% Have user select the directory:
start_path = 'EnterPathToYourDataHere';
directory = uigetdir(start_path,'Set Directory');

% Set directory to user-specified directory:
cd(directory);

% Create a folder that will hold all data analysis from this run: Determine
date and time. Create a string in "dateform" "30" (ISO 8601) which has
% the format 'yyyyymmddTHHMSS'.
dstr = datestr(now, 30);

% Concatenate with "_Time_Segment":
analysis_folder = [dstr '_Time_Segment'];

% Create folder in current experimental condition directory:
mkdir(analysis_folder);

% Log path to the "Analysis" folder to store data:
analysis_path = [directory '/' analysis_folder];

% Start a log file:
logfile = [dstr '_Time_Segment_Log.txt'];
cd(analysis_path);
fid = fopen(logfile,'wt');
cd('..');

% Print directory to log file:
fprintf(fid,'%s

',dstr);
fprintf(1,'Selected directory is:
');
fprintf(1, '%s 

', directory);
fprintf(fid,'Selected directory is:
');
fprintf(fid, '%s 
', directory);

% Send user-specified path to ReadFolderContents.m for generation of a list
% of folders that conform to "Loc_XX" naming convention:
[num_folders, folderlist] = ReadFolderContents_v3(directory, fid);

% Send list of "Loc_XX" folders to "ReadTiffImages.m" for generation of a
% cell array of ".tif" filenames corresponding to a particular folder
% (organized by columns):
[frame_array] = ReadTiffImages_v1(num_folders, folderlist, fid);

% Extract time values from all ".tif" file names in 'frame_array'
[t_abs] = Abs_Time_v1(frame_array, fid);

% Sort t_abs array in ascending order so that first imaging frame occupies
% row 1 of t_abs_sorted and any locations with unequal numbers of frames
% have padding zeros at end of column.
[t_abs_sorted] = Sort_Time_v1(t_abs, fid);

% At this point we have an array of time sorted values that are "absolute"
% time values. We will now generate an array of time values that are
% rounded according to the attempted time interval between frames specified
% by the user. As an example if the user specified imaging interval was 60
% sec and a frame is taken at 64 sec this frame will be rounded to 60 sec.
choice = menu('Did you attempt constant time-interval imaging?',...
'Yes', 'No');
if choice == 1 % Yes
    % Have user supply what the attempted imaging rate was in seconds:
    interval = input('
What was the intended imaging time-interval(sec)? :');
    % Record progress:
    fprintf(1, '

User attempted constant time interval imaging
');
    fprintf(1, 'Attempted imaging rate was = %.0f sec

', interval);
    fprintf(fid, '

User attempted constant time interval imaging
');
    fprintf(fid, 'Attempted imaging rate was = %.0f sec

', interval);
    % Go to 'Bin_Time.m' function which will zero the origin of each
    % location and bin each time stamp to its closest multiple of
    % 'interval'.
    [t_bin] = Bin_Time_v4(t_abs_sorted, interval, fid);
elseif choice == 2 % No
    % Record progress:
    fprintf(1, '
User did not attempt constant time interval imaging
');
    fprintf(1, 'As such no binning of time data in "t" was attempted

');
    fprintf(fid, '
User did not attempt constant time interval imaging
');
    fprintf(fid, 'As such no binning of time data in "t" was attempted

');
end
% Automatically save time arrays:
cd(analysis_path);
fprintf(1, '

Saving time arrays...

');
fprintf(fid, '

Saving time arrays...

');
save t_abs.mat t_abs
save t_abs_sorted.mat t_abs_sorted
save t_bin.mat t_bin
fprintf(1, '

Items saved to:
%s

', pwd);
fprintf(fid, '

Items saved to:
%s

', pwd);
cd('..');
% Perform image segmentation:
Segmentation_v4(frame_array, folderlist, directory, analysis_path, fid)
fprintf(1, '
Program terminated

');
fprintf(fid, '
Program terminated

');
close(fid);

ReadFolderContents_v3.m

%Steven J. Henry
% 04/08/2011
%**************************************************************************
% PURPOSE:
% This function determines which folders in a user-specified directory
% conform to a user-specified name. Currently the function is coded to
% identify folders of the form "Loc_XX". The idea is that the user directs
% the program to a folder corresponding to a specific experimental
% condition in which reside multiple imaging locations. Each of these
% locations is to contribute data to the same experimental condition pool
% of data and so need to be analyzed in aggregate.
% %
% % ASSUMPTIONS:
% % The following filter logic has two criteria for considering whether or not
% % a particular element of the directory conforms to the "Loc_XX" naming
% % criteria. (1) The element must be exactly 6 characters long. (2) The
% % first four characters must be "Loc_". Thus an assumption is that the last
% % two characters of "Loc_XX" are integers from 0-9 in the X positions. It
% % would be possible for "Loc_LL" folders to pass where L represents a
% % letter A-Z. Why such a folder would exist is not apparent and not
% % anticipated to occur frequently. A more stringent filter could be coded
% that incorporates a third criteria whereby the last two entries are
% confirmed to be integer values. This is not done in the current version
% (Version 3).
%
% INPUT:
% directory = user specified path to a folder containing multiple imaging
% locations.
% fid = file ID to which warnings and progress is printed as a text
% file.
%
% OUTPUT:
% num_folders = number of folders that conform to "Loc_XX" naming
% convention after search is performed and deletions executed.
% folderlist = structural array containing list of folders conforming to
% "Loc_XX" naming convention as generated via MATLAB's intrinsic 'dir'
% command.

function [num_folders, folderlist] = ReadFolderContents_v3(directory, fid)

% Set the current directory to that specified by the user:
% cd(directory);

% Define a structural array consisting of the information corresponding to
% the folders contained within the current directory. The 'pwd' function is
% an intrinsic MATLAB function that lists the elements in the current
% directory.
folderlist = dir(pwd);

% Initialize the 'del' vector that holds the position of elements that
% require deleting from 'folderlist':
del = [];

% Initialize the 'pass' counter that holds the number of elements in
% 'folderlist' that passed the "Loc_XX" filter.
pass = 0;

% Determine the number of elements in the 'folderlist' array containing the
% contents of the current directory.
num_folders = numel(folderlist);

% This is a loop that identifies elements in 'folderlist' that are not
% named in the form "Loc_XX". After these elements are identified the
% necessary deletions are made from 'folderlist'. Note nothing in the
% actual physical directory is harmed.
for i = 1:num_folders
    % Initialize 'del_flag' to "off". This flag will denote whether or not
    % a particular entry needs to be deleted. It needs to be reset at the
    % start of each pass through the loop in the event that a previous
    % element in 'folderlist' did not pass the filter logic and needed to
    % be deleted, placing 'del_flag' in the "on" state. Ignore MATLAB's
    % warning at this line.
    del_flag = 0;
    % Log the name of element (i) of 'folderlist'
    name = folderlist(i).name;
    % Determine the number of characters in 'name'
    num_char = length(name);
    % The folder must contain exactly 6 characters (Loc_XX) to be a
    % location folder and not an extraneous file, folder, or directory
    % operator.
    if num_char ~= 6
% If folder name is not exactly 6 characters turn on 'del_flag' for
% this element 'i'
_del_flag = 1;

% Otherwise the length of the element name is 6 characters and so
% we determine if first four characters of the element name are
% "Loc_"
else
% Log the first four characters of the element:
lead_actual = name(1:4);
lead_desired = 'Loc_';

% If the actual leading characters are not "Loc_"
if strcmp(lead_actual,lead_desired) ~= 1

% If folder name does not begin with 'Loc_' turn on 'del_flag'
_del_flag = 1;

else
% If you made it to this point the given element (i) of
% 'folderlist' is **likely** a folder with the form "Loc_XX" so
% placed on "likely" because we've only ensured to this point
% that the folder name allowed to pass the filter is 6
% characters and has "Loc_" as the first four entries. Thus a
% folder with some error such as "Loc_LL" where L is a letter
% (A-Z) would not be caught by the filter. However such an
% error is not anticipated to occur frequently if at all.
pass = pass + 1;

% Ensure that 'del_flag' is still off (this is redundant).
_del_flag = 0;

end

end

% If del_flag is "on" than this element 'i' needs to be deleted from
% 'folderlist' so record element 'i' position in 'del' vector:
if del_flag == 1
% 'del' vector is empty then element 'i' is first entry to be
% logged:
if isempty(del) == 1

del(1,1) = i;
else
% Otherwise 'del' already contains element positions and so we
% need to expand the vector by one entry:
num_dels = length(del);
clear del;
del = zeros(num_dels+1,1);
del(1:num_dels,1) = del_old;
del(num_dels+1,1) = i;
end
end

end

% Clear 'num_dels' variable for future use:
clear num_dels

% Now we have a vector 'del' mapping to elements in 'folderlist' requiring
% deletion. It is important that we delete these elements in descending
% order (from last element position to first element position) to retain
% proper mapping. It is not necessary to sort the 'del' vector because the
% filter is performed in ascending order (top to bottom), so 'del' must
% consist of a vector of 'ascending' entries.
% If 'del' vector is not empty there are deletions to be made.
if isempty(del) == 0
% Determine number of deletions to be made:
num_dels = length(del);
% Note the following decreasing count works even if 'num_dels' = 1.
for j = num_dels:-1:1
% Delete position 'del(j)' from 'folderlist'
folderlist(del(j)) = []; end
end
%Verify that new size of 'folderlist' is equal to 'pass' the counter that
% stores number of entries that passed filter criteria. If not, tell user.
clear num_folders
num_folders = numel(folderlist);
if num_folders ~= pass
fprintf(1,'WARNING: Number of entries in "folderlist" is not equal to "pass"
');
fprintf(1,'Error occured in "ReadFolderContents_v3.m" function"
');
fprintf(fid,'WARNING: Number of entries in "folderlist" is not equal to "pass"
');
fprintf(fid,'Error occured in "ReadFolderContents_v3.m" function"
');
end

Parameters

ReadTiffImages_v1.m

% Steven J. Henry
% 04/08/2011
%**************************************************************************
% PURPOSE:
% This function identifies files of the form ".tif" in a given
% folder. The idea is that a driver cycles through the "Loc_XX" folders
% % previously identified by "ReadFolderContents_v3.m". This function takes
% % one of these folders and and determines what elements of the directory are
% % TIFF image files. These frames belong to a sequence of images presumably
% % of multiple cells from a single location in a single experimental
% % condition.
% %ASSUMPTIONS:
% %Short Answer:
% Folder being explored only contains TIFF files
% %that comprise the image stack. That is no extraneous TIFF files or stacks
% %are present.
% % Long Answer:
% The following filter logic has two criteria for
% considering whether or not a particular element of the directory is an
% image file. (1) The element must be minimally 5 characters long. (2) The
% last four characters must be ".tif". Thus an assumption is that the
% remainder of the ".tif" file is of the form ".XXtitttt...ttt" or
% % corresponding to "Loc_XX": ttt...ttt denotes some time stamp in seconds
% % of non-constant length. It is possible to include more stringent filters
% % that could separate phase from fluor images but as of the writing of
% % this function (Version 1, 04/08/2011) this need is not necessary
% % (anticipated to be required in the future). For now the present design is
% % sufficient, but requires that the operator only have the desired image
frames present in the "Loc_XX" folder being analyzed. That is the folder
should not contain both phase and fluor images or image stacks such as
from ImageJ.

% INPUT:
% num_folders = number of folders that conform to "Loc_XX" naming
% convention via "ReadFolderContents.m"
% folderlist = structural array containing list of folders conforming to
% "Loc_XX" naming convention via "ReadFolderConents.m"
% fid = file ID to which warnings and progress is printed as a text
% file.

% OUTPUT:

function [frame_array] = ReadTiffImages_v1(num_folders, folderlist, fid)
for i = 1:num_folders
    % Specify the location of folder 'i' in 'folderlist'
    loc_directory = [pwd '\' folderlist(i).name];
    % Set the current directory to the "Loc_XX" folder containing the
    % imaging files of interest:
    cd(loc_directory);
    % Output the path of this new directory (i.e. the path to the current
    % location folder).
    fprintf(1,'The current directory is:
 %s 

',pwd);
    fprintf(fid,'The current directory is:
 %s 

',pwd);
    % Define a structural array consisting of the information corresponding
    % to the folders contained within the current directory. The 'pwd'
    % function is an intrinsic MATLAB function that lists the elements in
    % the current directory.
    filelist = dir(pwd);
    % Initialize the 'del' vector that holds the position of elements that
    % require deleting from 'filelist':
    del = [];
    % Initialize the 'pass' counter that holds the number of elements in
    % 'filelist' that passed the "pXX" and ".tif" filters.
    pass = 0;
    % Determine the number of elements in 'filelist' containing the
    % contents of the current directory.
    num_files = numel(filelist);
    % Iterate over the present number of elements in 'filelist'
    for j = 1:num_files
        % Initialize 'del_flag' to "off". This flag will denote whether or
        % not a particular entry needs to be deleted. It needs to be reset
        % at the start of each pass through the loop in the event that a
        % previous element in 'filelist' did not pass the filter logic
        % and needed to be deleted, placing 'del_flag' in the "on" state.
        % Ignore MATLAB's warning at this line.
        del_flag = 0;
        % Log the name of element 'j' of 'filelist'
        name = filelist(j).name;
        % Determine the number of characters in 'name'
        num_char = size(name,2);
% Minimally the filename must contain 5 characters (x.xxx) to be a
% file and not a directory operator such as ' .' or '..' or a folder
% with less than 5 characters.
if num_char < 5

% If the element 'j' name is not minimally 5 characters turn
% 'del_flag' on
del_flag = 1;

% Otherwise the length of the element name is greater than or
% equal to 5 characters and so we determine if the last four
% characters of the name are a ' .tif' extension
else

% Flip the element name order:
eman = fliplr(name);

% Log the first four characters which in reverse order
% correspond to the extension of the file:
ext_actual = eman(1:4);
ext_desired = ' .tif';

% If the reversed extension is not ' .tif'
if strcmp(ext_actual, ext_desired) ~= 1

% Turn 'del_flag' "on"
del_flag = 1;

else

% If you made it to this point the given element 'j' of
% 'filelist' is a file with extension ' .tif' so advance the
% successful iteration counter by one.
pass = pass + 1;

% Ensure that 'del_flag' is still off (this is redundant)
del_flag = 0;
end
end
end

% Clear 'num_dels' variable for future use:
clear num_dels
% Now we have a vector 'del' mapping to elements in 'folderlist'
% requiring deletion. It is important that we delete these elements in
% descending order (from last element position to first element
% position) to retain proper mapping. It is not necessary to sort the
% 'del' vector because the filter is performed in ascending order (top
% to bottom). So 'del' must consist of a vector of 'ascending' entries.
% If 'del' vector is not empty there are deletions to be made.
if isempty(del) == 0
    % Determine number of deletions to be made:
    num_dels = length(del);
    % Note the following decreasing count works even if 'num_dels' = 1.
    for k = num_dels:-1:1
        % Delete position 'del(k)' from 'filelist'
        filelist(del(k)) = [];
    end
end
% Verify that new size of 'filelist' is equal to 'pass' the counter
% that stores number of entries that passed filter criteria. If not,
% tell user.
clear num_files
num_files = numel(filelist);
if num_files ~= pass
    fprintf(1,'
WARNING: Number of images in "filelist" is not equal to "pass"
');
    fprintf(1,'Error occurred in "ReadTiffImages.m" function on %s 
', foldername(i));
    fprintf(fid,1,'WARNING: Number of images in "filelist" is not equal to "pass"
');
    fprintf(1,'Error occurred in "ReadTiffImages.m" function on %s 
', foldername(i));
end
% If this is the first folder (or only folder being considered)
if i == 1
    % Preallocate a cell array (which is used so we can log text
    % strings in each cell) memory. We will use the number of frames in
    % the first folder as an estimate of the size of the array. If the
    % array needs to grow this can be done during processing.
    frame_array = cell(num_files, num_folders);
end
% Write the filename for this location to frame_array:
for n = 1:num_files
    frame_array{n,i}= = filelist(n).name;
end
% Jump back one level in the directory to the experimental condition
% folder
cd('..');

Abs_Time_v1.m

% Steven J. Henry
% 04/08/2011
%**************************************************************************
% PURPOSE:
% This function takes in the array 't' containing absolute time
% values and performs two functions. First each location (column of data)
% is shifted linearly such that the origin frame (first row of each column)
% is zero. Second each location is rounded to the nearest multiple of the
% user-specified imaginge rate 'interval'. At the end of processing a
% matrix 't_bin' results of the same dimensions as 't' but now possessing
% either zeros or integer multiples of 'interval.'

% ASSUMPTIONS:
% This function assumes that the general form of the filename is
% "pXxttt..ttt.tif" or "sXxttt..ttt.tif". "p"/"s" denotes "phase"/"fluor".
% "XX" denotes location number. "ttt..ttt" is a time-stamp in seconds of
% unspecified length. Then it is always true that the time stamp begins at
% position '4' and ends at position 'num_char-4' where 'num_char' is the
% total length of the name.

% INPUT:
% frame_array = cell array containing names of all ".tif" files associated
% with a given location (column)
% fid = file ID to which warnings and progress is printed as a text
% file.

% OUTPUT:
% t = array of absolute time values. Rows represent frames. Columns
% represent locations. This array is passed sorted from lowest to highest
% value. It is not necessarily true that all locations (columns) have the
% same number of frames (nonzero rows). The array is designed such that
% the shorter columns is padded with zeros. For example consider the
% following hypothetical three location matrix where the first location
% has 5 frames, the second location 3 frames, and the third location 4
% frames:
%           61 73 87
%           123 138 144
%           183 195 206
%           245 0   266
%           306 0   0
%
%**************************************************************************

function [t_abs] = Abs_Time_v1(frame_array, fid)

% Determine size of cell array 'frame_array'
[slice_max, loc_max] = size(frame_array);

% Create a matrix that will contain the embedded times in a given filename.
% Rows correspond to slice number and each column is a unique location. It
% is NOT necessarily the case that column number represents location
% number. For example its possible that a particular imaging set defined
% Loc_06 through Loc_09 as a particular experimental condition. However
% Loc_06 would be column 1, Loc_07 => column 2, etc...
% t_abs = zeros(slice_max, loc_max);

for i = 1:loc_max
    for j = 1:slice_max
        % Load the file name string 'title':
        title = frame_array{j,i};

        % If 'title' is NOT empty (meaning 'frame_array' entry (j,i) is
        % not empty:
        if isempty(title) == 0
            % If 'title' is NOT empty:
            if isempty(title) == 0
                % You have a problem because the entry you're dealing with
                % is not a string:
                fprintf(fid,'Occurred in "Abs_Time.m" for row = %.0f and col = %.0f in file %.

    % Make sure the entry is a character array (i.e. string).
    % If 'title' is not a string tell user:
    if ischar(title) == 0
        % You have a problem because the entry you're dealing with
        % is not a string:
        fprintf(fid,'Occurred in "Abs_Time.m" for row = %.0f and col = %.0f in file %.

fprintf(fid,'Occurred in "Abs_Time.m" for row = %.0f and col = %.0f in file %.

%**************************************************************************
% Otherwise you have a string so extract the embedded time stamp:
else
    % Compute number of characters in file name:
    num_char = length(title);
    % Isolate time portion of file name. This assumes that the general form of
    % the filename is "pXXtt...ttt.tif" or "sXXtt...ttt.tif". "p"/"s" denotes
    % location number. "ttt...ttt" is a time-stamp in seconds of unspecified
    % length. Then it is always true that the time stamp begins at position '4'
    % and ends at position 'num_char-4' where 'num_char' is the total length
    % of the name.
    t_abs_char = title(4:num_char-4);
    % Convert time portion of file name (a character string) to a numeric value:
    t_abs_num = str2double(t_abs_char);
    % Load numeric time corresponding to slice 'j' in array
    t_abs(j,i) = t_abs_num;
end
end
end

% Check post-processing dimensionality
[t_slices, t_locs] = size(t_abs);
if t_slices ~= slice_max
    fprintf(1, '
WARNING: # of rows in "t_abs" ~= "frame_array" after time stamp extraction

');
    fprintf(fid, '
WARNING: # of rows in "t_abs" ~= "frame_array" after time stamp extraction

');
end
if t_locs ~= loc_max
    fprintf(1,'
WARNING: # of cols in "t_abs" ~= "frame_array" after time stamp extraction

');
    fprintf(fid,'
WARNING: # of cols in "t_abs" ~= "frame_array" after time stamp extraction

');
end

Sort_Time_v1.m

% Steven J. Henry
% 04/08/2011

%**************************************************************************
% PURPOSE:
% The purpose of this function is to sort an array of number time values in
% ascending order. Since not every column will have the same number of
% nonzero entries (frame) it is necessary to correct the fact that applying
% a global sort to the entire column will result in zeros being the leading
% column entries. This function sorts all nonzero elements and then ensures
% that the first row of the resulting sorted array has the first imaging
% frame of the particular location. Empirically it is never the case that
% real imaging frame exists at absolute time t = 0 so we use zero entries
% in a column to denote nonexistent frames.
%
% ASSUMPTIONS:
% Zero entries in the time array 't_abs' denote a non-existent frame and
% not a real imaging frame. This assumption is reasonable as it has never
% been observed where an image is snapped at the instant the imaging
% LABVIEW program is started. Typically the first imaging frame begins at t
% = 1-5 sec.
% 
% INPUT:
% t_abs = array of UNSORTED absolute time values. Rows represent frames.
% Columns represent locations. This array is passed UNSORTED. It is not
% necessarily true that all locations (columns) have the same number of
% frames (nonzero rows). For example consider the following hypothetical
% three location matrix where the first location has 5 frames, the second
% location 3 frames, and the third location 4 frames:
%           123  195  206
%           61   138  144
%           183  73   87
%           306  0    266
%           245  0    0
% 
% fid = file ID to which warnings and progress is printed as a text
% file.
% 
% OUTPUT:
% t_abs_sorted = array of SORTED absolute time values. Rows represent
% frames. Columns represent locations. This array is passed SORTED from
% lowest to highest value. It is not necessarily true that all locations
% (columns) have the same number of frames (nonzero rows). The array is
% designed such that when two columns do not have an equal number of
% frames the remainder of the shorter columns is padded with zeros. For
% example consider the following hypothetical three location matrix where
% the first location has 5 frames, the second location 3 frames, and the
% third location 4 frames:
%           61   73   87
%           123  138  144
%           183  195  206
%           245  0    266
%           306  0    0
% 
% As Windows may have altered the order in which the files were saved to
% the location folder (perhaps as a result of the user applying a sort
% within the folder) we now sort the columns of t_abs in ascending order:
% t_abs_sorted = sort(t_abs,1,'ascend');
% 
% However, in the process of sorting entries in a given column of t we
% may end up placing zeros in the leading rows if a given location has less
% total slices than the location for a given experimental condition with
% the most number of slices.
% 
% Example: If Location 1 has 8 slices but Location 2 has 10 slices after
% sorting zeros are now placed in row 1 and row 2 of Location 2's column.
% Empirically it is never the case that an image is taken at absolute time
% "0" so we can filter zero values to mean non-existent slices. In general
% the soonest an image is taken is on the order of 1-5 sec after hitting
% "start" on the data collection LabView program.
% 
% As a check on our manipulations log the total number of nonzero elements
% in t_abs before anything is done:
% tot_nz_presort = nnz(t_abs);
% 
% Determine size of t_abs_sorted:
% [max_slices, max_locs] = size(t_abs_sorted);
% 
% Iterate through each column in t_abs_sorted
% for i = 1:max_locs
% 
% Determine the number of nonzero elements in each column:
% nnz_slices = nnz(t_abs_sorted(:,i));
% If the number of nonzero entries does not equal the total number of rows (i.e. value 'max_slices') then there are zeros in this column:
if nnz_slices ~= max_slices
    % Save the nonzero entries of column 'i' in array 't_abs_sorted'
temp_time = nonzeros(t_abs_sorted(:,i));
    % Write zeros to column 'i' in array 't_abs_sorted'
t_abs_sorted(:,i) = 0;
    % Print sorted nonzero values in 'temp_time' vector to column 'i'
t_abs_sorted(1:nnz_slices,i) = temp_time;
end
end

% As a check on our manipulations log the total number of nonzero elements after manipulations are done:
tot_nz_postsort = nnz(t_abs_sorted);

% IF total number of nonzero elements pre/post sorting do not equal you have a problem:
if tot_nz_postsort ~= tot_nz_presort
    fprintf(1,'
WARNING: Total number of nonzero elements after sorting does not
');
    fprintf(1,'equal total number of nonzero elements before sorting."
');
    fprintf(fid,'
WARNING: Total number of nonzero elements after sorting does not
');
    fprintf(fid,'equal total number of nonzero elements before sorting."
');
end

% Determine the number of slices in each location. This corresponds to the number of nonzero elements in each column. This can be used to verify that the proper number of entries were achieved for each folder.
slice_check = zeros(max_locs,2);
for j = 1:max_locs
    % Column 1 is location number
    slice_check(j,1) = j;
    % Column 2 is number of slices in location folder
    slice_check(j,2) = nnz(t_abs_sorted(:,j));
end

% Output information to user:
fprintf(1,'
Time retrieval completed

');
fprintf(fid,'
Time retrieval completed

');
fprintf(1,'Number of locations analyzed = %1.0f

',max_locs);
fprintf(fid,'Number of locations analyzed = %1.0f

',max_locs);
fprintf(1,'The number of slices analyzed for each location:
');
fprintf(fid,'The number of slices analyzed for each location:
');
for k = 1:max_locs
    fprintf(1,'%.0f	%.0f
', slice_check(k,:));
    fprintf(fid,'%.0f	%.0f
', slice_check(k,:));
end

%%%%%%%%%%%%%%%%%%%%%%%%

% Steven J. Henry
% 02/16/2015

%**************************************************************************
% PURPOSE:
% This function takes in absolute time values in seconds and rounds (bins)
% all values to the closest integer multiple of a user-specified time
% interval. It avoids systematic upwards rounding bias at remainders of
% exactly 0.5 by essentially flipping a coin to see whether such remainders
% are rounded up or down when such cases arise.

% ASSUMPTIONS:
% n/a
%

% INPUT:
% t_abs_sorted = array of SORTED absolute time values. Rows represent
% frames. Columns represent locations. This array is passed SORTED from
% lowest to highest value. It is not necessarily true that all locations
% (columns) have the same number of frames (nonzero rows). The array is
designed such that when two columns do not have an equal number of
frames the remainder of the shorter columns is padded with zeros. For
example consider the following hypothetical three location matrix where
the first location has 5 frames, the second location 3 frames, and the
third location 4 frames:

<table>
<thead>
<tr>
<th></th>
<th>61</th>
<th>73</th>
<th>87</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123</td>
<td>138</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>183</td>
<td>195</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>245</td>
<td>0</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>306</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

% interval = user-specified constant imaging rate attempted (during actual
% data collection) in seconds. This is the value the user set the LABVIEW
% program on the Nikon microscope.

% fid = file ID to which warnings and progress is printed as a text
% file.
%
% OUTPUT:
%**************************************************************************
60 function [t_bin] = Bin_Time_v4(t_abs_sorted, interval, fid)
61 % Get function name:
62 func_name = mfilename;
63
64 % Update log file that function is running:
65 fprintf(1,'
%s running ...
',func_name);
66 fprintf(fid,'
%s running ...
',func_name);
67
68 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
69 % warning dialog the log file records no errors/warnings generated:
70 warn = 0;
71
72 % Determine dimensionality of 'data':
73 [rows, cols] = size(t_abs_sorted);
74
75 % Reserve space for 't_bin':
76 t_bin = zeros(rows,cols);
77
78 % Before entering the analysis loop set the random number stream generator
79 to a seed based upon the current time. This will help to increase the
80 independence of two runs through the program, otherwise the same sequence
81 of coin flips could result each time this function is called from
82 % start-up:
83
84 % Save a 'defaultStream' that has parameters equivalent to those when
85 % MATLAB first starts up:
86 defaultStream = RandStream('mt19937ar','Seed',0);
87
88 % Generate a time-dependent seed:
89 mySeed = sum(100*clock);
90 % Create a stream based upon this seed:
91 algorithm = 'mt19937ar';
92 myStream = RandStream(algorithm,'Seed',mySeed);
93
94 % Set the stream from which 'rand', 'randn', and 'randi' draw numbers to
95 % 'myStream':
96 RandStream.setGlobalStream(myStream);
% Reset internal state or pointer within stream:
reset(myStream);

% Log the seed used:
fprintf(1,'\n\nAlgorithm type set to: %s\n',algorithm);
fprintf(1,'\n\nSeed used was: %f\n',mySeed);
fprintf(fid,'\n\nAlgorithm type set to: %s\n',algorithm);
fprintf(fid,'\n\nSeed used was: %f\n',mySeed);

% Record how many times remainders of 0.5 occur and of these how many were
% rounded up vs. down:
num_rem = 0;
num_down = 0;
num_up = 0;

% Loop over all "Loc_XX" folders
for i = 1:cols
    % Loop over all frames within a given folder:
    for j = 1:rows
        % Since supplied data is sorted we can treat first entry in each
        % column as the origin frame. We will normalize all subsequent
        % frames (for the given track 'i') with respect to this origin
        % frame:
        if j == 1
            t_abs_orig = t_abs_sorted(j,i);
            t_bin(j,i) = 0;
        else
            % Record the current time-stamp value:
            t_abs_now = t_abs_sorted(j,i);
            % If it is not zero (in which case it is a padding entry) shift
            % the value with respect to the track origin and round it to
            % the nearest multiple of the attempted constant imaging
            % interval:
            if t_abs_now ~= 0
                t_abs_shifted = t_abs_now - t_abs_orig;
                Q = floor(t_abs_shifted/interval);
                R = rem(t_abs_shifted,interval)/interval;
                if R ~= 0.5
                    % Shift current frame's time-stamp with respect to origin
                    % frame's time-stamp:
                    t_abs_shifted = t_abs_now - t_abs_orig;
                    % % Q is the quotient after division by the user-specified
                    % % 'interval'. For example if interval = 60 sec and the
                    % % given image has a shifted time stamp of 160 sec than the
                    % % corresponding entry in 'Q' is 2 (b/c 160/60 = 2 + 40/60).
                    % % Use of MATLAB's 'floor' function is to ensure we always
                    % % round down to the nearest whole integer value (the
                    % % quotient). To continue with the above example
                    % % round(160/60) = 3 but floor(160/60) = 2.
                    Q = floor(t_abs_shifted/interval);
                    % % R is the remainder after division normalized by
                    % % 'interval':
                    % % if R is not exactly 0.5:
                    R = rem(t_abs_shifted,160)/160;
                    % % Round it according to normal rules:
                    R = round(R);
                    if R ~= 0.5
                        % If R is exactly 0.5 randomly choose to round up or
                        % % down using a uniform random number generator so that
                        % % over many flips Prob(roundup) ~ Prob(rounddown) ~ 0.5
                        % % (50%).
                        if rand < 0.5
                            R = 0.5;
                        else
                            R = 0.5;
                        end
                    end
                else
                    % If R is exactly 0.5 randomly choose to round up or
                    % % down using a uniform random number generator so that
                    % % over many flips Prob(roundup) ~ Prob(rounddown) ~ 0.5
                    % % (50%).
                    if rand < 0.5
                        R = 0.5;
                    else
                        R = 0.5;
                    end
                end
            end
        end
    end
end
\texttt{\textcolor{red}{166} \% Update \# times remainders of 0.5 occur: } \\
\texttt{\textcolor{red}{167} num\_rem = num\_rem + 1; } \\
\texttt{\textcolor{red}{168} } \\
\texttt{\textcolor{red}{169} \% Flip the coin by drawing a number from (0,1): } \\
\texttt{\textcolor{red}{170} coin = rand; } \\
\texttt{\textcolor{red}{171} fprintf(1,'\n\textcolor{red}{172} coin tossed: rand = %f',coin); } \\
\texttt{\textcolor{red}{173} fprintf(fid,'\n\textcolor{red}{174} coin tossed: rand = %f',coin); } \\
\texttt{\textcolor{red}{175} } \\
\texttt{\textcolor{red}{176} \% If the coin landed on it's edge (0.5) keep flipping } \\
\texttt{\textcolor{red}{177} \% until it lands on a face: } \\
\texttt{\textcolor{red}{178} while coin == 0.5 } \\
\texttt{\textcolor{red}{179} \quad \textcolor{red}{180} coin = rand; } \\
\texttt{\textcolor{red}{181} end } \\
\texttt{\textcolor{red}{182} } \\
\texttt{\textcolor{red}{183} \% Do a redundant check that coin is not 0.5: } \\
\texttt{\textcolor{red}{184} if coin == 0.5 } \\
\texttt{\textcolor{red}{185} \quad \textcolor{red}{186} fprintf(1,'\n\textcolor{red}{187} WARNING: Coin is still on its edge (0.5) but escaped "while" loop.\n'); } \\
\texttt{\textcolor{red}{188} \quad \textcolor{red}{189} fprintf(fid,'\n\textcolor{red}{190} WARNING: Coin is still on its edge (0.5) but escaped "while" loop.\n'); } \\
\texttt{\textcolor{red}{191} \quad \textcolor{red}{192} warn = 1; } \\
\texttt{\textcolor{red}{193} end } \\
\texttt{\textcolor{red}{194} } \\
\texttt{\textcolor{red}{195} \% Now invoke the normal rounding rules. Because of the } \\
\texttt{\textcolor{red}{196} \% above logic we need not worry about cases where coin } \\
\texttt{\textcolor{red}{197} \% = 0.5. At this point coin must be less than or } \\
\texttt{\textcolor{red}{198} \% greater but not equal to 0.5. } \\
\texttt{\textcolor{red}{199} R = round(coin); } \\
\texttt{\textcolor{red}{200} } \\
\texttt{\textcolor{red}{201} \% Log how many roundups and rounddowns occured: } \\
\texttt{\textcolor{red}{202} if coin < 0.5 } \\
\texttt{\textcolor{red}{203} \quad num\_down = num\_down + 1; } \\
\texttt{\textcolor{red}{204} elseif coin > 0.5 } \\
\texttt{\textcolor{red}{205} \quad num\_up = num\_up + 1; } \\
\texttt{\textcolor{red}{206} end } \\
\texttt{\textcolor{red}{207} } \\
\texttt{\textcolor{red}{208} % Do a redundant check that R is binary. Note MATLAB gives } \\
\texttt{\textcolor{red}{209} % a warning with the use of "&" instead of short-circuit } \\
\texttt{\textcolor{red}{210} % operator "&&". Here we need MATLAB to evaluate both } \\
\texttt{\textcolor{red}{211} % criteria, satisfying a single criterion is not sufficient } \\
\texttt{\textcolor{red}{212} % in this case so we can't use a short-circuit operator. } \\
\texttt{\textcolor{red}{213} if R ~= 0 \& R ~= 1 } \\
\texttt{\textcolor{red}{214} \quad \textcolor{red}{215} fprintf(1,'\n\textcolor{red}{216} WARNING: Remainder value R is not binary but should be.\n'); } \\
\texttt{\textcolor{red}{217} \quad \textcolor{red}{218} fprintf(fid,'\n\textcolor{red}{219} WARNING: Remainder value R is not binary but should be.\n'); } \\
\texttt{\textcolor{red}{220} \quad \textcolor{red}{221} warn = 1; } \\
\texttt{\textcolor{red}{222} end } \\
\texttt{\textcolor{red}{223} } \\
\texttt{\textcolor{red}{224} } \\
\texttt{\textcolor{red}{225} % Generate the binned time value: } \\
\texttt{\textcolor{red}{226} t\_bin(j,i) = (Q+R)\*interval; } \\
\texttt{\textcolor{red}{227} end } \\
\texttt{\textcolor{red}{228} } \\
\texttt{\textcolor{red}{229} end } \\
\texttt{\textcolor{red}{230} } \\
\texttt{\textcolor{red}{231} % Output stats on special rounding cases: } \\
\texttt{\textcolor{red}{232} fprintf(1,'\n\textcolor{red}{233} Number of remainders exactly = 0.5 was %f',num\_rem); } \\
\texttt{\textcolor{red}{234} fprintf(fid,'\n\textcolor{red}{235} Number of remainders exactly = 0.5 was %f',num\_rem); } \\
\texttt{\textcolor{red}{236} end } \\
\texttt{\textcolor{red}{237} } \\
\texttt{\textcolor{red}{238} }
fprintf(fid,'\t%.0f rounded up and %.0f rounded down\n',num_up,num_down);
235
% Check post-processing dimensionality:
236 [rows2, cols2] = size(t_bin);
238 if rows2 ~= rows
239 fprintf(1, '\n\tWARNING: # of rows in output "t_bin" different than input "t_abs_sorted"\n');
240 fprintf(fid,'\n\tWARNING: # of rows in output "t_bin" different than input "t_abs_sorted"\n');
241 warn = 1;
242 end
243 if cols2 ~= cols
244 fprintf(1, '\n\tWARNING: # of cols in output "t_bin" different than input "t_abs_sorted"\n');
245 fprintf(fid,'\n\tWARNING: # of cols in output "t_bin" different than input "t_abs_sorted"\n');
246 warn = 1;
247 end
248
% Return default stream back to conditions at MATLAB startup:
250 RandStream.setGlobalStream(defaultStream);
252
% If no warnings generated report so in log file:
255 if warn == 0
256 fprintf(1,'\n\tFunction completed without errors/warnings\n');
257 fprintf(fid,'\n\tFunction completed without errors/warnings\n');
258 end
259
% Update log file that function is completed:
261 fprintf(1, '\n%s completed\n',func_name);
262 fprintf(fid, '\n%s completed\n',func_name);
263 end
264

Segmentation_v4.m

% Steven J. Henry
% 06/15/2011

%**************************************************************************
% PURPOSE:
% The purpose of this function is to segment grayscale TIFF images and
% identify cell boundaries.
% % ASSUMPTIONS:
% % All images are of similar quality such that the hardcoded segmentation
% % algorithm is valid for all frames. Raw images were generated such that
% % the cells have dark-body centers (~black) and light halo outlines
% % (~white). A histogram of the raw image(s) should reveal a spike at an
% % intermediate grayscale intensity with no saturation at [0, 255].
% %**************************************************************************

1 % frame_array = cell array containing names of all "tif" files associated
2 % with a given location (column) via "ReadTiffImages.m"
3 % folderlist = structural array containing list of folders conforming to
4 % "Loc_XX" naming convention via "ReadFolderContents.m"
5 % raw_path = absolute path to folder containing "Loc_XX" folders with TIFF
6 % files to be analyzed
7 % analyzed_path = absolute path to folder containing "Loc_XX" folders with
8 % segmented TIFF files and associated time arrays
9 % fid = file ID to which warnings and progress is printed as a text
10 % file.
11 %
12 % OUPTUT: (not returned to driver)
13 % For each image file processed two segmentation files result:
14 % 1) "BW_pXXttt...ttt.tif" is a black/white thresholded image with black
15 % background and white cell bodies.
16 % 2) "pXXttt...ttt.tif" is the original image file
17 % It is necessary to duplicate the
function [] = Segmentation_v4(frame_array, folderlist, raw_path, analyzed_path, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Determine dimensionality of 'frame_array'
[slices, locs] = size(frame_array);

% Loop over all "Loc_XX" folders residing in 'folderlist'
for i = 1:locs

% Log current folder name:
folder_name = folderlist(i).name;

% Create a "Loc_XX" folder in the "Analysis" folder to store segmented
% images:
cd(analyzed_path);
mkdir(folder_name)
analyzed_loc_path = [analyzed_path ' ' folder_name];

% Specify the location of raw data folder 'i' in 'folderlist':
raw_loc_path = [raw_path ' ' folder_name];

% Set the current directory to the "Loc_XX" folder containing the
% imaging files of interest:

cd(raw_loc_path);

% Output the path of the folder you are about to perform segmentation
% on:
fprintf(1,'
	Segmenting images in the following directory (%.0f of %.0f):
	%s 

',i,locs,pwd);
fprintf(fid,'
	Segmenting images in the following directory (%.0f of %.0f):
	%s 

',i,locs,pwd);

% Loop over all *.tif* files in folder 'i':
for j = 1:slices

% If this is the first slice in this location create folders to
% contain segmented data:
if j == 1

% Change directory to the "Loc_XX" folder in "Analysis" folder:
    cd(analyzed_loc_path);
% Make folder names to hold segmentation results:
    seg_folder = 'Segmentation';
    overlay_folder = 'Overlay';
% Create segmentation results folder:
    mkdir(seg_folder);
    mkdir(overlay_folder);
% Get paths to these folders:
    seg_path = [analyzed_loc_path ' ' seg_folder];
    overlap_path = [analyzed_loc_path ' ' overlay_folder];
% Go back to location of raw data files:
    cd(raw_loc_path);
end

% Get filename from 'frame_array'
filename = frame_array(j,i);

% If 'filename' is not empty then use it to load the corresponding
% image file:
if isempty(filename) == 0

% Output the slice you’re on
if mod(j,5)==0
    fprintf(1,’%Segmenting image %.0fn’,j);
end

% Load image given by ‘filename’:
I = imread(filename);

% Perform edge detection using ‘roberts’ algorithm. This was
% empirically determined to give the best signal to noise
% ratio.
BW = edge(I,’roberts’);

% Create disk structuring element using MATLAB’s ‘strel’
% function.
SE_dil = strel(‘disk’, 3);

% Use this structuring element to dilate the BW image:
BWdil = imdilate(BW, SE_dil);

% Fill dilated image holes:
BWfill = imfill(BWdil, ‘holes’);

% Clear border elements:
BWNobord = imclearborder(BWfill, 8);

% The ability to perform the complementary erosion operation
% after the dilation is commented out below. As both dilation
% and erosion are nonlinear operations it is not necessarily
% true that eroding a dilated image removes the previous
% operation’s effect(s). Generally it has been found that the
% roberts edge detection underestimates the cell boundary and
% so the dilation is not tremendously deleterious.

% Create disk structuring element using MATLAB’s ‘strel’
% function.
SE_ero = strel(‘disk’,1);

% Erode all edges:
BWerode = imerode(BWNobord, SE_ero);

% Clear elements less than and empirically determined number of
% pixels.
% Imaging at 20XLWD, binning 1:
% hNeutrophils on uCP hFN are on the order of 4000 pixels in
% area. Over the period of an hour area will reduce to
% approximately 2000 pixels. Thus set threshold at 1500 pixels.
% Imaging at 20XLWD, binning 2:
% Set value at 500 pixels.
BWfinal = bwareaopen(BWerode, 250);

% Save segmented image:
BWname = [‘BW_‘ filename];
cd(seg_path);
imwrite(BWfinal, BWname, ‘tif’, ‘Compression’, ‘none’);
cd(raw_loc_path);

% Overlay segmentation boundary on original ‘.tif’ file:
BWoutline = bwperim(BWfinal);
Segout = I;
Segout(BWoutline) = 0; % 0 = black, 255 = white
BWoutline_name = [‘Perim_‘ filename];
cd(overlay_path);
imwrite(Segout, BWoutline_name, ‘tif’, ‘Compression’, ‘none’);
cd(raw_loc_path);
end
% Jump back to the experimental condition folder containing all raw
% Loc_XX folders:
cd(raw_path);
end

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'%nFunction completed without errors/warnings%n');
    fprintf(fid,'%nFunction completed without errors/warnings%n');
end

% Update log file that function is completed:
fprintf(1,'%n%s completed
',func_name);
fprintf(fid,'%n%s completed
',func_name);

end

%==================================================================
% PURPOSE:
% After "Time_Segment_Driver.m" has been run to generate folders containing
% segmented TIFF images this driver is run to track cell centroids.
%==================================================================
% ASSUMPTIONS:
% (1) In a given experimental condition's "Analysis" folder (generated via
% "Time_Segment_Driver.m" user has reserved "Loc_XX" naming for only those
% folders containing segmented images.
% (2) Segmented images reside in a folder entitled "Segmentation" within
% each "Loc_XX" folder.
% (3) "Segmentation" folders contain binary (logical) TIFF images
% (4) Segmentation image files have names that take the form
% "BW_pXXttt...ttt.tif" or "BW_sXXttt...ttt.tif" where p/s denotes "phase"
% or "fluor". XX denotes the location corresponding to "Loc_XX".
% "ttt...ttt" denotes some time stamp in seconds of non-constant length
%==================================================================
% DRIVER/FUNCTION MAP:
% Level     Name:
% 0         CC_Driver_v5.m
% 1             ReadFolderContents_v3.m
% 1             Track_Centroids_v5.m
% 2                 Pos_Selection_v3.m
% 1             IJ_Manual_Track_Prep_v3.m
%==================================================================

clc
clear all
close all

% Have user select the directory:
directory = uigetdir('EnterPathToYourDataHere','Set Directory');

% Set directory to user-specified directory:
cd(directory);

% Determine date and time. Create a string in "dateform" "30"
% (ISO 8601) which has the format 'yyyyMMddTHHmmss':
dstr = datestr(now, 30);

% Start a log file. Save in user-specified 'directory':
% Note: The notation "CC" stands for "connected components"
logfile = [dstr '_CC_Log.txt'];
fid = fopen(logfile,'wt');

% Print directory to log file:
fprintf(fid,'Selected directory is:
');
fprintf(fid,%'\ls \n', directory);
fprintf(fid,dstr);
fprintf(fid,'Selected directory is:
');

% Send user-specified path to ReadFolderContents.m for generation of a list
% of folders that conform to "Loc_XX" naming convention:
[num_folders, folderlist] = ReadFolderContents_v3(directory, fid);

% Load 't_abs_sorted.mat' from "Sort_Time.m" called in
% "Time_Segment_Driver.m". Rows represent frames. Columns represent
% locations. This array is passed SORTED from lowest to highest value. It
% is not necessarily true that all locations (columns) have the same number
% of frames (nonzero rows). The array is designed such that when two
% columns do not have an equal number of frames the remainder of the
% shorter columns is padded with zeros. For example consider the following
% hypothetical three location matrix where the first location has 5 frames,
% the second location 3 frames, and the third location 4 frames:
%           61    73     87
%           123   138   144
%           183   195   206
%           245   0     266
%           306   0     0
load t_abs_sorted.mat

% Load 't_bin.mat' from "Bin_Time.m" called in "Time_Segment_Driver.m"
% t_bin = array of binned time-values. This array has taken the absolute
% time array in 't_abs_sorted.mat', zeroed each location's origin image
% timestamp and binned all remaining times with respect to the closest
% multiple of 'interval'. That is all time stamps are now either zero or an
% integer multiple of 'interval'.
load t_bin.mat

% Prompt user to supply the maximum number of pixels that an object can
% travel between consecutive frames.
fprintf(fid,'Specify the maximum euclidean distance (pixels) that a cell
can travel between two consecutive frames. Typically for neutrophils imaged at 20XLWD at 60 sec/frame this value is:
');
fprintf(fid,'\%dx %dx %d
');
d_max_pixels = 30;
fprintf(fid,'User set d_max_pixels = %.0f

',d_max_pixels);

% Ask the user whether or not these segmented images were from phase
% microscopy or fluorescent microscopy. This is required to know whether
% the files to be loaded have "BW_pXXttt...ttt.tif" or
% "BW_sXXttt...ttt.tif" names:
choice = menu('Are segmented images from phase or fluor imaging?'...
    ,'Phase','Fluor');
if choice == 1 % Phase
    prefix = 'p';
    prefix_print = 'phase "p"
else
    prefix = 's';
    prefix_print = 'phase "s"
end
prefix = 's';
prefix_print = 'fluor "s"';
end

% Pause to allow menu graphic to clear from view before proceeding:
pause(2);

fprintf(1,'
User selected that segmented images are %s
',prefix_print);
fprintf(fid,'
User selected that segmented images are %s
',prefix_print);

% Have user input values for:
% max_frame_skips = user specified # of frames that can be skipped when
% linking an object's centroids into a trajectory. Two centroids
% separated by a number of frames greater than this value will not be
% connected and the object's trajectory will be terminated.
% min_frame_track = user specified # of total frames that an object must be
% tracked for it to be included in final data set fid = handle to log

fprintf(1,'
Specify max # frames an object can skip and still be tracked.
');
fprintf(1,'
Empirically 3 frames has been found to be a reasonable value.
');
acceptable = 0;
while acceptable == 0;

    max_frame_skips = 10;
    fprintf(1,'

	User set max_frame_skips = %.0f',max_frame_skips);
    fprintf(fid,'

	User set max_frame_skips = %.0f',max_frame_skips);
    if mod(max_frame_skips,1) ~= 0 || max_frame_skips < 0;
        fprintf(1,'
	WARNING: Entry must be 0 or a positive integer
');
        fprintf(fid,'
	WARNING: Entry must be 0 or a positive integer
');
        acceptable = 0;
    else
        fprintf(1,'
	Entry acceptable.
');
        fprintf(fid,'
	Entry acceptable.
');
        acceptable = 1;
    end

end

fprintf(1,'
Specify min # frames an object must be tracked for inclusion in final data set.
');
fprintf(1,'
Minimally this value must be 2 frames.
');
acceptable = 0;
while acceptable == 0;

    min_frame_track = 4;
    fprintf(1,'

	User set min_frame_track = %.0f',min_frame_track);
    fprintf(fid,'

	User set min_frame_track = %.0f',min_frame_track);
    if mod(min_frame_track,1) ~= 0 || min_frame_track < 2;
        fprintf(1,'
	WARNING: Entry must be an integer >= 2
');
        fprintf(fid,'
	WARNING: Entry must be an integer >= 2
');
        acceptable = 0;
    else
        fprintf(1,'
	Entry acceptable.
');
        fprintf(fid,'
	Entry acceptable.
');
        acceptable = 1;
    end

end

fprintf(1,'

% Loop over all "Loc_XX" folders in 'folderlist' which has length
% 'num_folders'
for i = 1:num_folders
    loc_folder_name = folderlist(i).name;
    t_abs_vec = t_abs_sorted(:,i);
    t_bin_vec = t_bin(:,i);
    [obj] = Track_Centroids_v5(t_abs_vec, t_bin_vec, loc_folder_name, directory, d_max_pixels, prefix, max_frame_skips, min_frame_track, fid);
end

fprintf(1,'Program terminated

');
fprintf(fid,'
Program terminated

');
fclose(fid);

---

Track_Centroids_v5.m

% Steven J. Henry
% 06/20/2011
%**************************************************************************
% PURPOSE:
% This function tracks cell centroids using binary TIFF images previously
% segmented via "Segment.m".
% %
% % ASSUMPTIONS:
% % There is a direct mapping between the entries of the sorted time matrix
% % 't_abs_sorted' and the files residing within the "Segmentation" folder
% % of the "Time_Segment" folder for a particular experimental condition.
% % Basically, this is a long way of saying, the assumption is the user
% % hasn't removed segmented files from the "Segmentation" folder prior to
% % analysis.
% % INPUT:
% % t_abs_vec = row vector of SORTED absolute time values. Rows represent
% % frames. Vector is passed SORTED from lowest to highest value. It is not
% % necessarily true that all rows have data as a particular Loc_XX may
% % have had less frames than another Loc_XX in the same experimental
% % condition. If this is the case the remainder of the column is padded
% % with zeros. For example consider the following hypothetical three
% % location matrix where the first location has 5 frames, the second
% % location 3 frames, and the third location 4 frames:
% % 61    73      87
% % 123   138     144
% % 183   195     206
% % 245   0       266
% % 306   0       0
% % In this case t_abs_vec would be a single column of this array.
% % t_bin_vec = binned or rounded absolute time values. Same structure and
% % rules apply as in t_abs_vec, except that the first row should now
% % contain a zero (i.e. frame 1 occurs at a relative time of zero).
% % loc_folder_name = string in form of "Loc_XX"
% % directory = user specified path to an "Analysis" folder containing the
% % results of segmentation analysis on multiple imaging locations
% % d_max_pixels = user specified upper bound on euclidean distance (in
a cell can be considered to move during a single frame
prefix = string ‘p’ for phase or ‘s’ for fluor to appropriately load
max_frames_skips = user specified # of frames that can be skipped when
linking an object’s centroids into a trajectory. Two centroids
separated by a number of frames greater than this value will not be
connected and the object’s trajectory will be terminated.
min_frame_track = user specified # of total frames that an object must be
tracked for it to be included in final data set fid = handle to log

% OUTPUT
% obj = cell array of length equal to number of objects identified in first
% frame of stack. Each cell contains a matrix with the following
% organization:
%   col 1 = space holder for ImageJ Manual Tracking compatibility
%   col 2 = object number
%   col 3 = frame number in which object ('obj' row position) is found
%   col 4 = x coordinate (pixels) of centroid object ('obj' row position)
%   col 5 = y coordinate (pixels) of centroid object ('obj' row position)
%   col 6 = absolute time corresponding to frame in which object is found
%   col 7 = binned time corresponding to frame in which object is found
%   col 8 = area of object

% FUNCTIONS CALLED:
% Pos_Selection_v2.m
function [obj] = Track_Centroids_v5(t_abs_vec, t_bin_vec,...
loc_folder_name, directory, d_max_pixels, prefix, max_frame_skips, ...
min_frame_track, fid)
% Load frame:
BW = imread(BW_image_name);

% Generate list of connected components:
cc = bwconncomp(BW);

% Create a structural array to hold information on geometric centroid
% and area of each object (connected component in 'cc') identified in
% this BW image:
cc_stats = regionprops(cc, 'Centroid', 'Area');

% If this is the first frame, generate a cell array to hold object
% information (abs_time, bin_time, centroid x pos, centroid y pos, and
% object area):
if j == 1
    obj = cell(cc.NumObjects,1);
else
    obj_match = zeros(num_obj,1);
end

% Loop over all detected objects in frame 1:
for k = 1:cc.NumObjects
    % Create a row vector of 8 positions:
    obj_info = zeros(1,8);
    % Position 1 is a spacer that will not hold any meaningful
    % data. It is necessary for compatibility with ImageJ's Manual
    % Tracking Plugin.
    % Position 2 holds object number:
    obj_info(2) = k;
    % Position 3 holds frame (slice) number:
    obj_info(3) = j;
    % Position 4 holds x component of object 'k' centroid
    obj_info(4) = cc_stats(k).Centroid(1);
    % Position 5 holds y component of object 'k' centroid
    obj_info(5) = cc_stats(k).Centroid(2);
    % Position 6 holds absolute time value corresponding to frame
    % j:
    obj_info(6) = t_abs_vec(j);
    % Position 7 holds binned time value corresponding to frame j:
    obj_info(7) = t_bin_vec(j);
    % Position 8 holds object area (# pixels)
    obj_info(8) = cc_stats(k).Area;
    % After all positions have proper values logged, write the
    % single vector to the appropriate cell in 'obj':
    obj(k) = obj_info;
end

% Retain the total number of initial objects detected. This is the
% number of objects tracked for the remainder of processing, it
% will not increase. After the first frame the program only seeks
% to identify these initialized objects in all following frames.
% This prevents tracking of objects that enter the field of view
% during data collection. It is not efficient in the sense that the
% program will attempt to find objects that may have already left
% the field of view. However, from an empirical perspective the
% current efficiency is satisfactory.
num_obj = cc.NumObjects;

% Otherwise j ~= 1 so this 'BW' frame is not the first frame and
% 'obj' array already exists
else
    % Create a column vector of length equal to the number of objects
    % being tracked from frame 1. Row position corresponds to the object
    % being tracked from frame 1. The vector will hold the IDs of the
    % objects in this current non-origin frame (j==1) that are found to
    % be continuations of the objects being tracked from frame 1.
    obj_match = zeros(num_obj,1);
% Loop over the number of original objects being tracked:
for k = 1:num_obj
    % Load the last observed centroid of object 'k':
    obj_info = obj{k};
    x_o = obj_info(end,4);
    y_o = obj_info(end,5);
    % Reserve a temporary row vector with length = to the number of
    % newly detected objects in the current frame (~= 1). That is
    % length(obj) does not necessarily have to equal
    % length(d_vector):
    d_vector = zeros(cc.NumObjects,1);
    % Loop over the number of new objects detected in this "new"
    % frame 'j':
    for kk = 1:cc.NumObjects
        % Load the centroid of the new object:
        x_new = cc_stats(kk).Centroid(1);
        y_new = cc_stats(kk).Centroid(2);
        % Compute Euclidean distance d^2 = x^2 + y^2:
        d = sqrt((x_new - x_o)^2 + (y_new - y_o)^2);
        % Log this distance value into 'd_vector'. Note this vector
        % contains the distances from object 'k' of frame 1 to all
        % objects 1:'kk' of the present frame corresponding to
        % image 'BW':
        d_vector(kk) = d;
    end
    % Identify the minimum euclidean distance between objects in
    % frame 'j' and objects in frame '1':
    d_min = min(d_vector);
    % If this minimum value is less than the maximum permissible
    % travel between consecutive frames:
    if d_min <= d_max_pixels
        % Determine which object(s) correspond to minimum euclidean
        % distance. Here 'pos' is a vector which contains the
        % indices (row #) of entries in 'd_vector' that are set
        % equal to 'd_min':
        pos = find(d_vector == d_min);
        if length(pos) > 1
            % Select which object corresponds to object 'k' via
            % area consideration:
            areas = regionprops(cc,'Area');
            [pos] = Pos_Selection_v2(pos, areas, obj, k);
            clear areas
            fprintf(1,\n            Note: In frame %.0f object %.0f required area consideration to track centroid.out', j,k);
            fprintf(fid,\n            Note: In frame %.0f object %.0f required area consideration to track centroid.out', j,k);
        end
    end
    % Determine the number of frames that have elapsed
    % between this frame 'j' and the last frame a centroid
    % was recorded for object 'k' (column 3):
    frame_elapse = j - obj_info(end,3);
    % If the number of elapsed frames less than or equal to an
    % allowed threshold proceed to incorporate the latest 'pos'
    % info, otherwise skip it:
    if frame_elapse <= max_frame_skips
        obj_match(k) = pos;
    end
% At this point all object matching is completed and we must check
% if multiple objects have declared the same position in frame 'j'
% for their next occupancy (implying a collision):
clear pos
% Identify all nonzero entries in 'obj_match' vector:
pos_all = nonzeros(obj_match);
% Identify all unique nonzero entries in 'obj_match' vector:
pos_unique = unique(pos_all);
% If the number of nonzero entries in 'obj_match' is greater than
% the number of unique nonzero entries in 'obj_match' a collision
% has occurred:
if length(pos_all) > length(pos_unique)
    for m = 1:length(pos_all)
        [pos_ind] = find(obj_match(:) == pos_all(m));
        if length(pos_ind) > 1
            for mm = 1:length(pos_ind)
                obj_match(pos_ind(mm)) = 0;
            end
        end
    end
end
% Once all collisions have been eliminated update object
% information:
for k = 1:num_obj
    if obj_match(k) ~= 0
        % Update object 'k' info by adding a new row to the
        % existing object's information matrix:
        obj_info_old = obj{k};
        [row, col] = size(obj_info_old);
        obj_info_new = zeros(row+1, col);
        obj_info_new(1:row, 1:col) = obj_info_old;
        % Position 2 holds object number:
        obj_info_new(row+1, 2) = k;
        % Position 3 holds frame (slice) number:
        obj_info_new(row+1, 3) = j;
        % Position 4 holds x component of object 'pos' centroid:
        obj_info_new(row+1, 4) = cc_stats(obj_match(k)).Centroid(1);
        % Position 5 holds y component of object 'pos' centroid
        obj_info_new(row+1, 5) = cc_stats(obj_match(k)).Centroid(2);
        % Position 6 holds absolute time value corresponding to
        % frame 'j':
        obj_info_new(row+1, 6) = t_abs_vec(j);
        % Position 7 holds binned time value corresponding to frame
        % 'j':
        obj_info_new(row+1, 7) = t_bin_vec(j);
        % Position 8 holds object area (# pixels)
        obj_info_new(row+1, 8) = cc_stats(obj_match(k)).Area;
        % After all positions have proper values logged, write the
        % single vector to the appropriate cell in 'obj':
        obj{k} = obj_info_new;
    end
end
obj[k] = obj_info_new;
end
end
end

% Now filter out objects that have been tracked for less than a minimum
% number of frames:
min_frames = min_frame_track;

for n = 1:num_obj %Reverse order not required b/c 'obj' is cell array
    obj_frames = length(obj{n}(:,2));
    if obj_frames < min_frames
        obj{n} = [];
    end
end

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'Function completed without errors/warnings
');
    fprintf(fid,'Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(1,'
%s completed
',func_name);
fprintf(fid,'
%s completed
',func_name);
end

Pos_Selection_v2.m

% Steven J. Henry
% 04/19/2011
%**************************************************************************
% PURPOSE:
% This function is written in conjunction with "Track_Centroids.m". In the
% event that multiple objects in the present frame reside a distance
% 'd_min' from the object 'k' of interest the object with the minimum
% change in area is selected as the continuation of object 'k' in this
% frame.
%**************************************************************************

function [pos] = Pos_Selection_v2(pos, areas, obj, k)
How many connected components in the present frame need to be compared to object 'k'?

num_to_comp = length(pos);

Retrieving the 'k' info matrix:

obj_info = obj{k};

Isolating area column vector (col 8) from info matrix:

obj_area = obj_info(:,8);

Compute mean of area entries (minimally this has a single entry):

obj_m_area = mean(obj_area);

Get the areas and euclidean distance values corresponding to objects in 'pos' store these in 'pos_data' with column 1 = row position reference, column 2 = area of object 'i' in 'pos', column 3 = euclidean distance of object 'i' to tracked object 'k'

pos_data = zeros(num_to_comp, 2);
pos_data(:,1) = pos;
for i = 1:num_to_comp
    pos_data(i,2) = areas(pos(i)).Area;
end

Compute change in area from object 'k' (previous frame) and neighbor objects:

delta_area = abs(pos_data(:,2) - obj_m_area);

Find minimum change in area:

delta_area_min = min(delta_area);

Find the indices of the entries in 'delta_area' that contain 'delta_area_min' values:

[area_ind] = find(delta_area == delta_area_min);

Since 'delta_area', 'pos', and 'pos_data' have the same order, use 'area_ind' to determine which rows of 'pos_data' and 'pos' to retain

pos = pos(area_ind);

At this point it is still theoretically possible that 'pos' contains multiple values. If this is the case just output the first entry as the correct position:

if length(pos) > 1
    pos = pos(1);
end

IJ_Manual_Track_Prepp_v3.m

% Steven J. Henry
% 06/20/2011

% PURPOSE:
% This function takes the 'obj' cell array that contains information of object centroid tracking from "Track_Centroids.m" and performs manipulations necessary to make it suitable for import into "ImageJ Manual Tracking Plugin".
% INPUT:
% obj_cell_array = cell array of length equal to number of objects identified in first frame of stack. Each cell contains a matrix with the following organization:
% col 1 = space holder for ImageJ Manual Tracking compatibility
% col 2 = object number
% col 3 = frame number in which object ('obj' row position) is found
% col 4 = x coordinate (pixels) of centroid object ('obj' row position)
% col 5 = y coordinate (pixels) of centroid object ('obj' row position)
% col 6 = absolute time corresponding to frame in which object is found
% col 7 = binned time corresponding to frame in which object is found
% col 8 = area of object for filter applied in "Area_Consistency.m"
% directory = user specified path to an "Analysis" folder containing the
% results of segmentation analysis on multiple imaging locations
% fid = file ID to which warnings and progress is printed as a text
% file.
% dstr = date and time string in "dateform" "30" (ISO 8601) which has the
% format 'yyyyymmddTHHMMSS'.

function [obj_num_array] = IJ_Manual_Track_Prep_v3(obj_cell_array, loc_folder_name, directory, fid)

    % Determine number of objects tracked:
    num_objs = length(obj_cell_array);

    % Reserve variable name:
    obj_num_array = [];

    % Loop over these objects
    for i = 1:num_objs
        % Extract info array for object 'i' from 'obj_cell_array' supplied:
        obj_info = obj_cell_array{i};
        if isempty(obj_info) == 0
            % If this is the first cell in object with actual data:
            if isempty(obj_num_array) == 1
                % Set the output array equal to the info array of this object:
                obj_num_array = obj_info;
            else
                % Otherwise if there are more than one objects:
                clear obj_num_array
                obj_num_array = vertcat(obj_num_array_old, obj_num_array);
            end
            % Record the previous output information to be expanded
            clear obj_num_array
            obj_num_array = vertcat(obj_num_array_old, obj_num_info);
        end
    end

end
% Determine dimensions of output array:
[rows, cols] = size(obj_num_array);

% If the number of columns is not equal to 8 then some error occurred upon
% concatenation or "Track_Centroids.m" provided erroneous 'obj' info:
if cols ~= 8
    fprintf(1,"nWARNING: vertcat resulted in array without 8 columns

');
    fprintf(fid,"nWARNING: vertcat resulted in array without 8 columns

');
end

% Add a 9th column to hold "Track Change" flag:
obj_num_array_old = obj_num_array;
clear obj_num_array
obj_num_array = zeros(rows, cols+1);
obj_num_array(1:rows, 1:cols) = obj_num_array_old;

% Turn on "track change flag" when new object ID is detected:
% Loop over all entries in 'obj_num_array'
for j = 1:rows
    % If this is the first entry set Track Change flag "on" = 1
    if j == 1
        obj_num_array(1,9) = 1;
    % Otherwise if this is not the first entry determine if this entry
    % has the same track ID as the previous entry. If so keep track
    % change flag "off" (0), if not turn track change flag "on" (1).
    else
        % Load previous track ID and current track ID:
        obj_ID_prior = obj_num_array(j-1,2);
        obj_ID_now = obj_num_array(j, 2);
        % If they are not the same
        if obj_ID_now ~= obj_ID_prior
            % Turn track chang flag on for this 'j' entry:
            obj_num_array(j,9) = 1;
        else
            % Otherwise ensure track change flag is off (this is
            % redundant).
            obj_num_array(j,9) = 0;
        end
    end
end

% Now re-assign object ID so objects have consecutive track ID starting at
% 1:
for jj = 1:rows
    if jj == 1
        obj_num_array(jj,2) = 1;
    else
        obj_ID_prior = obj_num_array(jj-1,2);
        track_chng_flag = obj_num_array(jj,9);
        if track_chng_flag == 1;
        obj_num_array(jj,2) = obj_num_array(jj-1,2);
        track_chng_flag = obj_num_array(jj,9);
        if track_chng_flag == 1;
        end
    end
end

end
obj_ID_now = obj_ID_prior+1;
else
    obj_ID_now = obj_ID_prior;
end
obj_num_array(jj,2) = obj_ID_now;
end
end

% Set directory to "Loc_XX" folder within "Analysis" folder:
loc_path = [directory '\loc_folder_name'];
cd(loc_path);

% Write 'obj_num_array' to a text file and save as a .mat file for later
% revision:
txt_fname = [loc_folder_name '.txt'];
fid2 = fopen(txt_fname, 'wt');
fprintf(1,'
Saving %s in:
', txt_fname);
fprintf(1,'%s

', pwd);
fprintf(fid2, 'n/a	ID	Slice	X(pixel)	Y(pixel)	Time(s)	BinTime(s)	Area(pixels)	TrackChange
');
for k = 1:rows
    fprintf(fid2,'%.0f	%.0f	%.0f	%f	%f	%.0f	%.0f	%.0f	%.0f	
',obj_num_array(k,:));
end
mat_fname = [loc_folder_name '.mat'];
save(mat_fname, 'obj_num_array');
cd('..');
fclose(fid2);
end

CC_Output_Editor_v2.m

% Steven J. Henry
% 02/16/2015
%**************************************************************************
% PURPOSE:
% This program is run on individual "Loc_XX.mat" files after "CC_Driver.m"
% has generated a data set compatible with ImageJ's Manual Tracking Plugin.
% It operates on "Loc_XX.mat" to eliminate entire tracks and portions of
% tracks specified by the user (entered manully) to generate an edited
% "Loc_XX_edited.mat" and "Loc_XX_editied.txt" file that only contains
% cell centroids to be used in MSD computation.
% Note: MATLAB generates warnings related to this program (see orange flags
% to the right in the Editor window). These are notifying the user that
% care has not been taken with respect to memory conservation. Because the
% data sets being processed at a time are relatively small this program is
% sloppy and allows arrays to grow and shrink without reserving the
% appropriate block of memory.
% ASSUMPTIONS:
% n/a
% INPUT:
% obj_num_array = numeric array of length equal to number of objects
% thought to be cells times the number of total frames those objects were
% tracked having the following organization:
% col 1 = space holder for ImageJ Manual Tracking compatibility
% col 2 = unique track number ID assigned to each object
% col 3 = frame number in which object ("obj" row position) is found
% col 4 = x coordinate (pixels) of centroid object ("obj" row position)
% col 5 = y coordinate (pixels) of centroid object ("obj" row position)
% col 6 = absolute time corresponding to frame in which object is found
% col 7 = binned time corresponding to frame in which object is found
% col 8 = area of object in pixels
% col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
% if no (i.e. continuation of an existing track.
% OUTPUT:
% obj_num_array = same structure as input but excluding tracks and portions
% tracks manually deemed unsuitable for inclusion in the final data set
% for MSD computation.
%**************************************************************************
clc
clear all
close all

% Get "Loc_XX.mat" file name and path:
[filename, pathname] = uigetfile(....
    'EnterPathToYourDataHere',....
    'Select File');

% Determine date and time. Create a string in "dateform" "30" (ISO 8601)
% which has the format 'yyymmmdddTHHMMSS'.
dstr = datestr(now, 30);
log_fname = [dstr '_' filename_no_ext '_CC_Edits_Log.txt'];
fid1 = fopen(log_fname, 'wt');

% Load "Loc_XX.mat" file which contains variable 'obj_num_array':
load(filename);

% User supplied vector of track IDs to be completely eliminated. This can
% be a row or column vector
entire_dels = [];
num_entire_dels = length(entire_dels);

% Eliminate data OUTSIDE (not including) user-specified boundaries (i.e.
% frames) for a given track. Input structure should be:
% Column 1 = track ID
% Column 2 = lower bound, first frame cell should be followed
% Column 3 = upper bound, last frame cell should be followed
partial_dels_out = [];
num_partial_dels_out = size(partial_dels_out,1);

% Eliminate data INSIDE (and including) user-specified boundaries (i.e.
% frames) for a given track. Input structure should be:
% Column 1 = track ID
% Column 2 = lower bound, first frame cell data should be eliminated from
% Column 3 = upper bound, last frame cell data should be eliminated from
partial_dels_in = [];
num_partial_dels_in = size(partial_dels_in,1);

% Perform entire track deletions:
if num_entire_dels > 0

%**************************************************************************
for i = 1:num_entire_dels
    obj_ID = entire_dels(i);
    fprintf(fid1,'%.0f
',obj_ID);
    obj_ind = find(obj_num_array(:,2) == obj_ID);
    for ii = 1:length(obj_ind)
        obj_num_array(obj_ind(ii),:) = [];
    end
end

if num_partial_dels_out > 0
    fprintf(fid1,'
Partial deletions:
Data retained INSIDE (including) the following bounds:
Track ID	Start Frame	Stop Frame
');
    for j = 1:num_partial_dels_out
        obj_ID = partial_dels_out(j,1);
        obj_first_frame = partial_dels_out(j,2);
        obj_last_frame = partial_dels_out(j,3);
        fprintf(fid1,'%.0f	%.0f	%.0f
',obj_ID,obj_first_frame,obj_last_frame);
        obj_ind_lb = find((obj_num_array(:,2) == obj_ID) & ...
            (obj_num_array(:,3) < obj_first_frame));
        obj_ind_ub = find((obj_num_array(:,2) == obj_ID) & ...
            (obj_num_array(:,3) > obj_last_frame));
        obj_ind = union(obj_ind_lb, obj_ind_ub);
        for jj = 1:length(obj_ind)
            obj_num_array(obj_ind(jj),:) = [];
        end
    end
end

for i = 1:num_entire_dels
    obj_ID = entire_dels(i);
    fprintf(fid1,'%.0f
',obj_ID);
    obj_ind = find(obj_num_array(:,2) == obj_ID);
    obj_ind = sort(obj_ind, 'descend');
    for ii = 1:length(obj_ind)
        obj_num_array(obj_ind(ii),:) = [];
    end
end

for j = 1:num_partial_dels_out
    obj_ID = partial_dels_out(j,1);
    obj_first_frame = partial_dels_out(j,2);
    obj_last_frame = partial_dels_out(j,3);
    fprintf(fid1,'%.0f	%.0f	%.0f
',obj_ID,obj_first_frame,obj_last_frame);
    obj_ind_lb = find((obj_num_array(:,2) == obj_ID) & ...
        (obj_num_array(:,3) < obj_first_frame));
    obj_ind_ub = find((obj_num_array(:,2) == obj_ID) & ...
        (obj_num_array(:,3) > obj_last_frame));
    obj_ind = union(obj_ind_lb, obj_ind_ub);
    for jj = 1:length(obj_ind)
        obj_num_array(obj_ind(jj),:) = [];
    end
% Perform deletions inside of user-specified bounds:
if num_partial_dels_in > 0
    fprintf(1,'nPartial deletions:\n');
    fprintf(1,'Data retained OUTSIDE (not including) the following bounds:\n');
    fprintf(1,'Track ID	Start Frame	Stop Frame\n');
    fprintf(fid1,'nPartial deletions:\nData retained OUTSIDE (not including) the following bounds:\nTrack ID	Start Frame	Stop Frame\n');
    for j = 1:num_partial_dels_in
        obj_ID = partial_dels_in(j,1);
        obj_first_frame = partial_dels_in(j,2);
        obj_last_frame = partial_dels_in(j,3);
        fprintf(1,'%.0f	%.0f	%.0f\n',obj_ID,obj_first_frame,obj_last_frame);
        fprintf(fid1,'%.0f	%.0f	%.0f\n',obj_ID,obj_first_frame,obj_last_frame);
        % Find indices in obj_num_array corresponding to track 'j' that
        % violate lower bound set by user:
        obj_ind_lb = find((obj_num_array(:,2) == obj_ID) & (obj_num_array(:,3) >= obj_first_frame));
        % Find indices in obj_num_array corresponding to track 'j' that
        % violate upper bound set by user:
        obj_ind_ub = find((obj_num_array(:,2) == obj_ID) & (obj_num_array(:,3) <= obj_last_frame));
        % Retain the intersection of these two vectors:
        obj_ind = intersect(obj_ind_lb, obj_ind_ub);
        obj_ind = sort(obj_ind, 'descend');
        for jj = 1:length(obj_ind)
            obj_num_array(obj_ind(jj),:) = [];
        end
    end
end

% Note the following two loops were taken straight out of
% 'IJ_Manual_Track_Prep_v3.m'.
% Determine dimensions of output array:
[rows, cols] = size(obj_num_array);
% Turn on "track change flag" when new object ID is detected: Loop over all
% entries in 'obj_num_array'. We need to repeat this step previously
% performed in 'IJ_Manual_Track_Prep.m' in the event that the user has
% deleted an object's position in frame 1 but tracks that object in
% subsequent frames, thereby eliminating it's TCF marker.
for k = 1:rows
    % If this is the first entry set Track Change flag "on" = 1
    if k == 1;
        obj_num_array(1,9) = 1;
    elseif
        % Otherwise if this is not the first entry determine if this entry
        % has the same track ID as the previous entry. If so keep track
        % change flag "off" (0), if not turn track change flag "on" (1).
        obj_num_array(1,9) = 0;
    end
end

% 218
else

% Load previous track ID and current track ID:
obj_ID_prior = obj_num_array(k-1,2);
obj_ID_now = obj_num_array(k, 2);

% If they are not the same
if obj_ID_now ~= obj_ID_prior
    % Turn track chang flag on for this 'k' entry:
    obj_num_array(k,9) = 1;
else
    % Otherwise ensure track change flag is off (this is 
    % redundant).
    obj_num_array(k,9) = 0;
end
end
end

% Now re-assign object ID so objects have consecutive track ID starting at 
% 1:
for kk = 1:rows
    if kk == 1
        obj_num_array(kk,2) = 1;
    else
        obj_ID_prior = obj_num_array(kk-1,2);
        track_chng_flag = obj_num_array(kk,9);
        if track_chng_flag == 1;
            obj_ID_now = obj_ID_prior+1;
        else
            obj_ID_now = obj_ID_prior;
        end
        obj_num_array(kk,2) = obj_ID_now;
    end
end

% Write 'obj_num_array' to a text file and save as a .mat file for later 
% revision:
ifd2 = fopen(txt_fname, 'wt');
fprintf(1,'
Saving %s in:
', txt_fname);
fprintf(1,'%s

', pwd);
fprintf(fid1,'
Saving %s in:
', txt_fname);
fprintf(fid1,'%s

', pwd);
fprintf(fid1, ... 
'n/a	ID	Slice	X(pix)	Y(pix)	Time(s)	BinTime(s)	Area(pix)	TrackChange
');
fprintf(fid2, ... 
'n/a	ID	Slice	X(pix)	Y(pix)	Time(s)	BinTime(s)	Area(pix)	TrackChange
');
for r = 1:rows
    fprintf(fid1,... 
'.0f%.0f%.0f%.0f%.0f%.0f%.0f%.0f
');
    fprintf(fid2,... 
'.0f%.0f%.0f%.0f%.0f%.0f%.0f%.0f
');
end
% Steven Henry  
% 02/16/2015
%**************************************************************************
% PURPOSE:  
% After "CC_Driver.m" and "CC_Output_Editor.m" have been run and a series  
% of '.mat' files containing centroid tracking data result, this program is  
% run to merge all these location specific files into a single data file  
% % representative of all cells tracked for the given experimental condition.  
% %**************************************************************************
% % ASSUMPTIONS:  
% % (1) User only supplis '.mat' files relevant to the given experimental  
% % condition being analyzed  
% % INPUT:  
% % Individual .mat files containing the array "obj_num_array" that has the  
% % following structure:  
% %   col 1 = space holder for ImageJ Manual Tracking compatibility  
% %   col 2 = unique track number ID assigned to each object  
% %   col 3 = frame number in which object ('obj' row position) is found  
% %   col 4 = x coordinate (pixels) of centroid object ('obj' row position)  
% %   col 5 = y coordinate (pixels) of centroid object ('obj' row position)  
% %   col 6 = absolute time cooresponding to frame in which object is found  
% %   col 7 = binned time corresponding to frame in which object is found  
% %   col 8 = area of object in pixels  
% %   col 9 = track change flag. Entry = 1 if start of new track (yes) or 0  
% % if no (i.e. continuation of an existing track.  
% % OUTPUT:  
% % Single .mat file and .txt file with same structure as stated in INPUT but  
% % having unique and consecutive track IDs assigned to all cells.  
% %**************************************************************************

clc
clear all

directory = uigetdir('EnterPathToYourDataHere', 'Select folder containing .mat files to be merged:');

mat_file = uigetfile([directory '\*.mat'], 'Select .mat files to merge:', 'MultiSelect', 'on');

mat_fname = [filename_no_ext '_edited.mat'];
save(mat_fname, 'obj_num_array');

fprintf(fid1, 'Program terminated

');
fclose(fid1);
fclose(fid2);
% Start a log file. Save in user-specified directory:
logfile = ['dstr '_Merge_Mats_Log.txt'];
fid = fopen(logfile,'wt');

% Before merging '.mat' files make sure each '.mat' file has internally
% consistent track change flag assignments. This is to make sure that in
% the course of using "CC_Output_Editor.m" We did not inadvertantly
% eliminate a start row and thereby wipeout the track change flag as well.
if iscell(mat_file) == 0
    num_mats = 1;
else
    num_mats = length(mat_file);
end

% If user only selected one filename put this into a single cell array:
if num_mats == 1
    temp{1} = mat_file;
    clear mat_file
    mat_file = temp;
end

fprintf(1,'Checking each .mat file for correct track change flag (TCF) assignments:

');
fprintf(fid, 'Checking each .mat file for correct track change flag (TCF) assignments:

');

pass = 0;
for i = 1:num_mats
    fprintf(1,'
Processing %s now:
',mat_file{i});
    fprintf(fid,'
Processing %s now:
',mat_file{i});

    all_clear = 1;
    load(mat_file{i});
    [rows cols] = size(obj_num_array);

    if cols ~= 9
        fprintf(1,'WARNING: file has %.0f cols not 9 as required
',cols);
        fprintf(fid,'WARNING: file has %.0f cols not 9 as required
',cols);
        all_clear = -1;
    end

    all_clear = 1;
    if i == 1:rows
        if obj_num_array(ii,9) ~= 1
            fprintf(1,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
            fprintf(fid,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
            all_clear = -1;
        end
    end

    if ID_now ~= ID_prior
        fprintf(1,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
        fprintf(fid,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
        all_clear = -1;
    end

    if ID_now == ID_prior
        ID_prior = obj_num_array(ii-1,2);
        ID_now = obj_num_array(ii,2);
        TCF = obj_num_array(ii,9);
    end
if TCF ~= 1
  fprintf(1,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
  fprintf(fid,'WARNING: Track %.0f does not have TCF = 1 at row %0.f
',obj_num_array(ii,2),ii);
  all_clear = -1;
end
else
  if TCF ~= 0
    fprintf(1,'WARNING: Track %.0f does not have TCF = 0 at row %0.f
',obj_num_array(ii,2),ii);
    fprintf(fid,'WARNING: Track %.0f does not have TCF = 0 at row %0.f
',obj_num_array(ii,2),ii);
    all_clear = -1;
  end
end
end
end
end
% If you didn't trip the 'all_clear' flag tell user file is OK:
if all_clear == 1
  fprintf(1,'File is internally consistent
');
  fprintf(fid,'File is internally consistent
');
  pass = pass +1;
end
% Merge mats:
for j = 1:num_mats
  load(mat_file{j});
  if j == 1
    data = obj_num_array;
  else
    [new_rows new_cols] = size(obj_num_array);
    data_old = data;
    [old_rows old_cols] = size(data_old);
    clear data
    data = zeros(old_rows+new_rows, old_cols);
    data(1:old_rows, 1:old_cols) = data_old;
    data(old_rows+1:old_rows+new_rows,1:old_cols) = obj_num_array;
  end
  clear obj_num_array
end
% Now re-assign object ID so cells have unique and consecutive track
% IDs starting at 1:
[data_rows data_cols] = size(data);
for jj = 1:data_rows
  if jj == 1
    data(jj,2) = 1;
  else
    ID_prior = data(jj-1,2);
    TCF = data(jj,9);
    if TCF == 1;
ID_now = ID_prior+1;
else
    ID_now = ID_prior;
end
data(jj,2) = ID_now;
end
end

% Write 'data' to a text file and save as a .mat file for later analysis
fid2 = fopen(txt_fname, 'wt');
fprintf(fid2, 'n/a	ID	Slice	X(pix)	Y(pix)	Time(s)	BinTime(s)	Area(pix)	TrackChange
');
for k = 1:data_rows
    fprintf(fid2,'%.0f	%.0f	%.0f	%.12f	%.12f	%.0f	%.0f	%.0f
',data(k,:));
end
mat_fname = [dstr '_Merged_Data.mat'];
save(mat_fname, 'data');
fclose(fid2);

else
    fprintf(1, '
Not all .mats internally consistent. Program terminated.
');
fclose(fid);
end

%**************************************************************************
PURPOSE:
This program calls on 'MSD_Driver_v14.m' without requiring user-input
upon every iteration. It is intended to operate on a folder containing a
series of .mat files, each corresponding to a different condition, within
given experiment (i.e. donor/day).

ASSUMPTIONS:
Folder that Supra_MSD_Driver.m operates on contains .mat centroid files
with necessary structure and all files are to be processed in identical
fashion (for example same objective calibration, same max tau values,
etc...).
 filenames conform to the following 53 character naming convention:
 DXX_yyyymmdd_yyyymmddThhmmss_Merged_Data_XXXpXX_FN_XXXpXX_fMLP.mat
An example of this format referencing a real filename in the 03/25/2011
data set is:
D05_20110325_20110624T183042_Merged_Data_050p00_FN_000p00_fMLP.mat

 Level  Name:
clc
clear all
close all

% BEGIN USER INPUT:

% Have user select file(s) to be analyzed:
choice = menu('Analyze multiple conditions or a single condition?','Multiple','Single');
if choice == 1
    [file_name_list, file_path] = uigetfile('EnterPathToYourDataHere','Select *.mat file(s);','MultiSelect','on');
else choice == 2
    [file_name_list, file_path] = uigetfile('EnterPathToYourDataHere','Select *.mat file(s);','MultiSelect','off');
end

% Have user supply a calibration value for conversion of pixels to microns
% in units of microns/pixel:
pixel_calib = input('
Set pixel to micron conversion factor in units of microns per single pixel:
');

% Determine if user wants to perform MSD analysis using entire empirical
% data or only up to a user-specified 'exp_t_max' absolute experimental
% time:
choiceA = menu('Perform MSD analysis on full empirical data or a portion?','Full','Portion');
if choiceA == 1
    exp_t_max = [];
elseif choiceA == 2
    exp_t_max = input('
Enter maximum experimental imaging time (min) to be used in analysis: ');
end

% Determine if user wishes to supply a random noise value to subtract from
% all MSD data points?
choiceB = menu('Do you wish to supply a random noise estimate (epsilon) to be subtracted uniformly from all MSD
values?','Yes','No');
if choiceB == 1
    epsilon = 0.4604; % units are pix not pix^2
elseif choiceB == 2
    epsilon = [];
end

% Determine if user wants to:
% (1) fit models to MSD from t = 0 min to t = exp_t_max min
% (2) fit models to a portion of MSD data between t = 0 min and t =
% exp_t_max min. This means the user will set two tau bounds
% ('fit_tau_min' and 'fit_tau_max') that will denote the extent of the
% MSD data used when fitting. Logically these bounds can at most be
% fit_tau_min = 0 and fit_tau_max = exp_t_max.
choiceC = menu('Fit complete (i.e. tau = [0 exp_t_max]) or partial MSD data?','Complete','Partial');
if choiceC == 1
    fit_tau_bounds = [];
else choiceC == 2
    ok_bounds = 0;
end
fit_tau_min = input('Enter minimum tau interval (minutes) to fit: ');
fit_tau_max = input('Enter maximum tau interval (minutes) to fit: ');

if isempty(exp_t_max) == 0
    if (0>fit_tau_min) || (fit_tau_min >= fit_tau_max) || (fit_tau_max > exp_t_max)
        fprintf(1,'
	WARNING: entered values do not obey 0 <= fit_tau_min < fi_tau_max <= exp_t_max
');
        ok_bounds = 0;
    else
        fprintf(1,'
	Entered boundaries are acceptable.
');
        fit_tau_bounds(1) = fit_tau_min;
        fit_tau_bounds(2) = fit_tau_max;
        ok_bounds = 1;
    end
else
    if (0>fit_tau_min) || (fit_tau_min >= fit_tau_max)
        fprintf(1,'
	WARNING: entered values do not obey 0 <= fit_tau_min < fi_tau_max
');
        ok_bounds = 0;
    else
        fprintf(1,'
	Entered boundaries are acceptable.
');
        fit_tau_bounds(1) = fit_tau_min;
        fit_tau_bounds(2) = fit_tau_max;
        ok_bounds = 1;
    end
end

elseif isempty(exp_t_max) == 1
    if (0>fit_tau_min) || (fit_tau_min >= fit_tau_max)
        fprintf(1,'
	WARNING: entered values do not obey 0 <= fit_tau_min < fi_tau_max
');
        ok_bounds = 0;
    else
        fprintf(1,'
	Entered boundaries are acceptable.
');
        fit_tau_bounds(1) = fit_tau_min;
        fit_tau_bounds(2) = fit_tau_max;
        ok_bounds = 1;
    end
end

% END USER INPUT:
% *************************************************************************
% Set directory to user-specified directory:
cd(file_path);

% Determine date and time. Create a string in "dateform" "30" (ISO 8601)
% which has the format 'yyymmddTHHMMSS'.
dstr = datestr(now, 30);

% Create a folder to hold results of this plotting run:
complete_results_folder_name = [dstr '_MSD_Driver_v15'];
mkdir(complete_results_folder_name);

% Prepare an array for holding all the data that will be in the Excel file
% for weighted averaging:
xls_mimc = zeros(length(file_name_list),16);

% Loop over all conditions in folder:
for i = 1:length(file_name_list)
fprintf(1,'\nProcessing file %s of %s\n',num2str(i),num2str(length(file_name_list)));% Extract filename
file_name = file_name_list(i);% Call MSD_Driver.m to process file
[Donor, Donation, FN, fMLP, t_max, mi_counts, avg_all_disp, ...
  std_all_disp, avg_max_disp, std_max_disp, m_counts, ...
  Sout, Pout, mufit, Afit, alphasup, results_folder_name] = MSD_Driver_v15(file_name, file_path, pixel_calib, exp_t_max, epsilon, fit_tau_bounds);
% If this is the first file to be processed log header info in Excel
if i == 1
    header_info = {'Donor','Donation','FN','fMLP','t_max',...
                   'model indep n','<|All Disp|>','STD |All Disp|',...
                   '<|Max Disp|>','STD |Max Disp|','model n',...
                   'Sfit','Pfit','mufit','Afit','alpha fit',...
                   'ID','yyyyymmdd','ug/mL','nM','min',...
                   'counts','um','um',...
                   'um','um','counts',...
                   'um/min','min','um^2/min';
    xls_name = [file_path complete_results_folder_name '.xlsx'];
    xlswrite(xls_name,header_info,'MSD_Driver_v15','A1');
end
% Otherwise update current excel file with new condition info:
data_to_log = {Donor, Donation, FN, fMLP, t_max,...
               mi_counts, avg_all_disp, std_all_disp,...
               avg_max_disp, std_max_disp, m_counts,...
               Sout, Pout, mufit, Afit, alphasup};
row = i+2;
print_start = {'A' num2str(row)};
xlswrite(xls_name,data_to_log,'MSD_Driver_v15',print_start);
% Move condition-specific results folder into experiment folder
movefile([file_path results_folder_name],[file_path complete_results_folder_name]);
% When all files are processed move Excel file to experiment folder
movefile(xls_name,[file_path complete_results_folder_name]);

MSD_Driver_v15.m
% Steven J. Henry
% 03/03/2015
%**************************************************************************
% PURPOSE:
% This driver performs model independent analysis of individual cell
% centroid data. This driver also computes time and ensemble mean square
% displacements from the individual cell tracks and applies various models
% of motility to this data.
% % ASSUMPTIONS:
% (1) Merged data file (.mat) contains an array with the variable name
% 'data' and has the following structure:
% % col 1 = space holder for ImageJ Manual Tracking compatibility
% % col 2 = unique track number ID assigned to each cell
% % col 3 = frame number in which cell is found
% % col 4 = x coordinate (pixels) of centroid
% % After "Merge_Mats_m" has been run to generate a '.mat' and .txt' file
% % containing all cell tracks for a given experimental condition compiled
% % from multiple imaging locations within the same experimental condition,
% % this driver is run.
% col 5 = y coordinate (pixels) of centroid
% col 6 = absolute time corresponding to frame in which cell is found
% col 7 = binned time corresponding to frame in which cell is found
% col 8 = area of cell in pixels
% col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
% if no (i.e. continuation of an existing track.
% INPUT:
% file_name = name of experimental condition .mat file to be loaded
% file_path = path to folder containing 'file_name' above
% pixel_calib = user-specified objective calibration (um/pix)
% exp_t_max = user-specified maximum experimental time to use in data
% analysis (min) or empty
% epsilon = user-specified correction for camera noise in pix or empty
% fit_tau_bounds = user-specified bounds for fitting motility models to a
% portion of MSD data between tau = [tau_bounds(1) tau_bounds(2)] min
% OUTPUT:
% Donor = number unique to that donor
% Donation = date in 'yyyymmdd' format of 'Donor' blood draw
% FN = [FN] as auto-read from .mat file (ug/mL)
% fMLP = [fMLP] as auto-read from .mat file (nM)
% t_max = maximum experimental imaging time (min) used in analysis. This
% value is either exp_t_max as set by the user or the maximum time value
% observed in the loaded data set depending on whether or not exp_t_max
% is defined
% mi_counts = number of tracks contributing data to model independent
% analysis
% avg_all_disp = mean of all absolute displacements observed (um)
% std_all_disp = standard deviation of all absolute displacement observed
% avg_max_disp = mean of max absolute displacements observed (um)
% std_max_disp = standard deviation of max absolute displacements observed
% m_counts = number of tracks contributing data to model dependent analysis
% Sout = best-fit speed parameter from biased random walk model (um/min)
% Pout = best-fit persistence parameter from biased random walk model (min)
% muout = random motility coefficient using best-fit biased random walk
% parameters = 0.5*Sout^2*Pout (um^2/min)
% Aout = best-fit coefficient parameter from power law model
% alphaout = best-fit power parameter from power law model (unitless)
% analysis is stored
% DRIVER/FUNCTION MAP:
% Level Name:
% -1 Supra_MSD_Driver_v5.m
% 0 MSD_Driver_v15.m
% 1 Parse_Filename_v2.m
% 1 Post_IJ_Manual_Track_v3.m
% 1 Cell_Track_Plotter_v6.m
% 1 Consec_Differentials_v4.m
% 1 Step_Size_Stationarity_v2.m
% 1 Histograms_v3.m
% 1 Path_Length_v6.m
% 1 Mean_Path_Length_v5.m
% 1 Area_v4.m
% 1 Mean_Area_v5.m
% 1 Filter_Exp_Data_v3.m
% 1 Differentials_v5.m
% 1 Neff_v1.m
% 1 Mean_Differentials_v6.m
% 1 MSD_Epsilon_Subtract_v3.m
% 1 Plot_Mean_Differentials_v5.m
% 1 Filter_Mean_Differentials_v4.m
% 1 SandP_v11.m
% 1 Power_Law_v4.m
% Load the file which must contain the variable 'data':
load(file_name,'data');

% Determine date and time. Create a string in "dateform" "30" (ISO 8601)
% which has the format 'yyymmddTHHMMS'.
dstr = datestr(now, 30);

% Extract condition information from filename (requires filenames obey
% standard naming convention):
[Donor, Donation, FN, fMLP, run_title] = Parse_Filename_v2(file_name);

% Create a folder to hold results of analysis on this condition:
results_folder_name = [dstr '_' run_title];
mkdir(results_folder_name);

% Set directory to analysis folder:
cd([file_path results_folder_name]);

% Start a log file. Save in new directory:
logfile = [results_folder_name '_Log.txt'];
fid = fopen(logfile,'wt');

% Update log file on progress:
fprintf(1,'Merged data file imported:
');
fprintf(1,'%s

',file_path file_name);
fprintf(fid,'%s
',dstr);

fprintf(1,'Title of run:
');
fprintf(1,'%s

',run_title);
fprintf(fid,'Title of run:
');
fprintf(fid,'%s

',run_title);

fprintf(1,'Results of analysis saved at:
');
fprintf(1,'%s

',file_path results_folder_name);
fprintf(fid,'Results of analysis saved at:
');
fprintf(fid,'%s

',file_path results_folder_name);

% Record pixel calibration value supplied:
 fprintf(fid,'User set pixel_calib = %s um/pixel

',num2str(pixel_calib));

% Eliminate unnecessary components of 'data' that are artifacts from ImageJ
% Manual Tracking Plugin compatibility requirements previously. Also ensure
% (again) that start of each unique track ID is consistent with placement
% of track change flags:
[data] = Post_IJ_Manual_Track_v3(data, fid); %#ok<NODEF>

% Plot cell trajectories eminating from single origin and compute
% associated model-independent statistics:
[mi_counts, avg_allDisp, std_allDisp, avg_maxDisp, std_maxDisp] = Cell_Track_Plotter_v6(exp_t_max, pixel_calib, data, run_title, fid);

% Close figures generated here to prevent Java overload:
close all

% Compute absolute differentials in displacement between two consecutive
% frames of all tracked objects. This data will be used for bias analysis.
% and determination of population stationarity.
[CAD] = Consec_Differentials_v4(data, fid);

% A plot of the absolute value of consecutive absolute differentials in x
% and y are plotted as a function of elapsed experimental to determine
% extent of population stationarity. Essentially we are plotting the mean
% step size as a function of experimental time. We wish to identify the
% time period in which this stepsize is essentially constant.
% <|delta_x(tau_min)|> vs. experimental time
% <|delta_y(tau_min)|> vs. experimental time
[SSD, tau_min, t_max] = Step_Size_Stationarity_v2(CAD, run_title, fid);

% Generate histograms to check for tracking bias:
Histograms_v3(data, CAD, tau_min, run_title, fid);

% Compute cumulative distance traveled (path length) of individual cell
% tracks as a function of elapsed experimental imaging time and plot
% results:
[Path_Length] = Path_Length_v6(data, pixel_calib, run_title, fid);

% Compute mean (ensemble averaged) cumulative distance traveled (path
% length) of individual cell tracks as a function of elapsed experimental
% imaging time and plot results:
Mean_Path_Length_v5(Path_Length, pixel_calib, run_title, fid);

% Determine area of individual cell tracks as a function of elapsed time
% and plot results:
[Area] = Area_v4(data, pixel_calib, run_title, fid);

% Compute and plot mean (ensemble averaged) area over all cells as a
% function of lag time tau in terms of absolute or binned time
% (not time intervals):
Mean_Area_v5(Area, pixel_calib, run_title, fid);

% If 'exp_t_max' is empty utilize complete empirical data available to
% compute MSD:
if isempty(exp_t_max) == 1

    % Update log:
    fprintf(1,'User opted to perform MSD analysis on complete empirical data.
');
    fprintf(fid,'User opted to perform MSD analysis on complete empirical data.
');

    fprintf(1,'Total experimental imaging duration was %s min
',num2str(t_max));
    fprintf(fid,'User set max experimental imaging time to consider in MSD analysis as %s
',num2str(exp_t_max));

    % If 'exp_t_max' is not empty, work with only a portion of complete
% empirical data available:
    elseif isempty(exp_t_max) == 0

        % Update log:
        fprintf(1,'User opted to perform MSD analysis on PORTION of empirical data.
');
        fprintf(fid,'User opted to perform MSD analysis on PORTION of empirical data.
');

        fprintf(1,'Total experimental imaging duration was %s min
',num2str(t_max));
        fprintf(fid,'User set max experimental imaging time to consider in MSD analysis as %s
',num2str(exp_t_max));

        % Record exp_t_max value sent from 'Supre_MSD_Driver.m'
        fprintf(1,'User set max experimental imaging time to consider in MSD analysis as %s
',num2str(exp_t_max));

        % Filter data array so that it contains only rows corresponding to
% absolute image time stamps less than or equal to user-specified
% 'exp_t_max':
        [data] = Filter_Exp_Data_v3(data, exp_t_max, fid);

        % Since user has already selected a maximum imaging time we will set
% this as the upper bound on any plots that have an abscissa of
% time.
\texttt{t\_max = exp\_t\_max;} \\
\texttt{end} \\

% Report number of tracks that will contribute intervals to model-dependent 
% analysis: 
\texttt{m\_counts = length(unique(data(:,1));} \\

% Compute squared displacements of all cells using MOVING origin 
% strategy in preparation for computation of mean (time and ensemble 
% average) squared displacement as a function of lag time (tau) in terms of 
% both absolute and binned time intervals: 
\texttt{[SD] = Differentials\_v5(data, fid);} \\

% Compute the number of total independent observations possible ("Neff") 
% corresponding with tau values used in MSD analysis. 
\texttt{[Indep\_Obs\_tabs, Indep\_Obs\_tbin] = Neff\_v1(data, SD, fid);} \\

% Compute mean (time and ensemble averaged) squared displacements of all 
% cells as a function of lag time (tau) in terms of both absolute and 
% binned time intervals: 
\texttt{[MSD\_tabs, MSD\_tbin] = Mean\_Differentials\_v6(SD, Indep\_Obs\_tabs, Indep\_Obs\_tbin, fid);} \\

% Yes, supply an estimate of random noise (epsilon value): 
\texttt{if isempty(epsilon) == 0} 
\% Update log: 
\texttt{fprintf(1,"\texttt{User supplied estimate of random noise for this experimental condition.\n\texttt{4*epsilon^2 will be subtracted uniformly from all MSD values.\n\texttt{User set epsilon = \texttt{num2str(epsilon)} pix\n\texttt{User supplied estimate of random noise for this experimental condition.\n\texttt{4*epsilon^2 will be subtracted uniformly from all MSD values.\n\texttt{User set epsilon = num2str(epsilon)} pix\n\texttt{User did NOT supply estimate of random noise.\n\texttt{MSD values are left uncorrected.\n\texttt{User did NOT supply estimate of random noise.\n\texttt{MSD values are left uncorrected.\n\texttt{epsilon\_flag = 1;\n\texttt{else if isempty(epsilon) == 1} 
\% Update log: 
\texttt{fprintf(1,"\texttt{User did NOT supply estimate of random noise.\n\texttt{MSD values are left uncorrected.\n\texttt{User did NOT supply estimate of random noise.\n\texttt{MSD values are left uncorrected.\n\texttt{epsilon\_flag = 0;\n\texttt{end} 
\% Plot mean (time and ensemble averaged) squared displacements of all cells 
% as a function of lag time (tau) in terms of both absolute and binned time 
\% intervals: 
\texttt{Plot\_Mean\_Differentials\_v5(SD, MSD\_tabs, MSD\_tbin, pixel\_calib, t\_max, run\_title, epsilon\_flag, fid);} \\
\texttt{if isempty(fit\_tau\_bounds) == 1} 
% Fit full MSD data available between \([0 \ exp\_t\_max]\) minutes 
\texttt{fprintf(fid,"\texttt{User opted to fit model(s) to MSD data between tau = [0 \ %s] min\n\texttt{num2str(exp\_t\_max));\n\texttt{fprintf(fid,"\texttt{User opted to fit model(s) to MSD data between tau = [0 \ %s] min\n\texttt{num2str(exp\_t\_max));\n\texttt{end}
% Fit filtered MSD array with biased random walk model:
[fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v11(MSD_tabs, MSD_tbin, pixel_calib, fid);

% Fit filtered MSD array with power law model:
[fit_PL_tabs fit_PL_tbin Aout alphaout] = Power_Law_v4(MSD_tabs, MSD_tbin, pixel_calib, fid);

% Fit portion of MSD data between user-defined [fit_tau_bounds(1) fit_tau_bounds(2)] minutes:
elseif isempty(fit_tau_bounds) == 0

% Update log:
fprintf(1,\n% User opted to fit model(s) to MSD data between tau = [%s %s]
\n
User opted to fit model(s) to MSD data between tau = [%s %s]

',num2str(fit_tau_bounds(1)),num2str(fit_tau_bounds(2)),num2str(fit_tau_bounds(1)),num2str(fit_tau_bounds(2)), fid);

% Filter MSD arrays so that they contain only those values
% corresponding to taus in the range [fit_tau_bounds(1),
% fit_tau_bounds(2)]:
[MSD_tabs_part MSD_tbin_part] = Filter_Mean_Differentials_v4(MSD_tabs, MSD_tbin, fit_tau_bounds(1),
fit_tau_bounds(2), epsilon_flag, fid);

% Lin spaced data on log axes:
keep_dt = 1.15.^((0:99));
[dummy, ind] = unique(keep_dt,'first');
keep_dt = keep_dt(ind);
ind = find(keep_dt <= round(max(MSD_tbin_part(:,5))/min(MSD_tbin_part(:,5))));

% Fit filtered MSD array with biased random walk model:
[fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v10(MSD_tabs_part, MSD_tbin_part, pixel_calib, fid);

% Fit filtered MSD array with power law model:
[fit_PL_tabs fit_PL_tbin Aout alphaout] = Power_Law_v4(MSD_tabs_part, MSD_tbin_part, pixel_calib, fid);

eend

% Overlay full empirical MSD data with fit data and plot results:
Plot_SandP_Fit_v6(MSD_tabs, MSD_tbin, fit_BRW_tabs, fit_BRW_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid);
Plot_Power_Law_Fit_v4(MSD_tabs, MSD_tbin, fit_PL_tabs, fit_PL_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid);

close all

% Close figures so Java doesn't overload
% Close figures so Java doesn't overload
% Close figures so Java doesn't overload

% Perform Van Hove analysis as a check on the MSD analysis completed
% previously:
Van_Hove_Analysis_v3(pixel_calib, data, MSD_tbin, epsilon_flag, run_title, fid)

% Update log file that program is terminated:
fprintf(1,\n% Program Terminated\n
',fid);

fclose(fid);

% Sort all files generated into folders of .fig, .mats, and .txt files
% leaving the master log file residing outside the three folders:
Tidy_Up_v1(logfile);

% Close any figures remaining open

close all;

dd
Parse_FILENAME_v2.m

% Steven J. Henry
% 11/03/2011
%**************************************************************************
% PURPOSE:
% This function extracts the concentrations of FN and fMLP from the
% filename.
%**************************************************************************
% REMARKS:
% This function is meant for ease of analysis of the hNeutrophils on uCP
% hFN in aqueous fMLP study and can be eliminated from 'MSD_Driver.m' to
% make that driver and its functionality more general in the future.
% %
% % ASSUMPTIONS:
% filenames conform to the following 53 character naming convention:
% DXX_yyyymmdd_yyyymmddThhmmss_Merged_Data_XXXpXXFN_XXXpXXfMLP.mat
% An example of this format referencing a real filename in the 03/25/2011
% data set is:
% D05_20110325_20110624T183042_Merged_Data_050p00FN_000p00fMLP.mat
% %
% % INPUT:
% filename = 566 character string with structure as stated previously in
% % ASSUMPTIONS
% %
% % OUTPUT:
% run_title = 'XXXpXXFN_XXXpXXfMLP' portion of filename
% FN = numeric value of FN concentration in ug/mL
% fMLP = numeric value of fMLP concentration in nM
% %**************************************************************************

function [Donor, Donation, FN, fMLP, run_title] = Parse_FILENAME_v2(filename)

% Make sure loaded filename has 66 characters
if length(filename)~= 66
    fprintf(1,'WARNING: file name loaded does not adhere to naming convention required to auto-read conditions\n\n');
    cancel
else
    % Retain FN and fMLP portion of current filename for future reference
    run_title = filename(1:end-4);
    % Extract numeric FN value:
    FN_str = filename(end-24:end-19);
    FN_str(4)='.';
    FN = str2num(FN_str);

    % Extract numeric fMLP value:
    fMLP_str = filename(end-14:end-9);
    fMLP_str(4)='.';
    fMLP = str2num(fMLP_str); %#ok<*ST2NM>

    % Extract Donation Date:
    Donation_str = filename(end-61:end-54);
    Donation = str2num(Donation_str);

    % Extract Donor ID:
    Donor_str = filename(2:3);
    Donor = str2num(Donor_str);
end
end
% Steven J. Henry
% 04/30/2011
%**************************************************************************
% PURPOSE:
% This is a very simple function called on by "MSD_Driver_v1.m". It has two
% purposes:
% (1) Eliminates unnecessary components of supplied data array that are
% artifacts as a result of compatibility requirements for interfacing with
% ImageJ's Manual Tracking Plugin.
% (2) Confirm that start of each unique track ID is consistent with
% placement/location of track change flags (values of 1). In theory this is
% redundant because "Merge_Mats_v2.m" does this same task prior to saving
% the 'data' file imported into this program. However, my concern is that
% if the user manually adjusted the 'data' array after "Merge_Mats_v2.m"
% but prior to import here the result could be internal inconsistency
% especially if the user were to delete a track initiation row (a row in
% which the track change flag is set to "on" or value of 1).
% %
% % ASSUMPTIONS:
% %
% % INPUT:
% % data = an array containing all pertinent tracking information for each
% % cell in the given experimental condition having the following structure:
% %   col 1 = space holder for ImageJ Manual Tracking compatibility
% %   col 2 = unique track number ID assigned to each cell
% %   col 3 = frame number in which cell is found
% %   col 4 = x coordinate (pixels) of centroid
% %   col 5 = y coordinate (pixels) of centroid
% %   col 6 = absolute time corresponding to frame in which cell is found
% %           (sec)
% %   col 7 = binned time corresponding to frame in which cell is found (sec)
% %   col 8 = area of cell in pixels
% %   col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
% %           if no (i.e. continuation of an existing track.
% % fid = file ID of log file to which progress is recorded
% %
% % OUTPUT:
% % data = same array but now having the following structure:
% %   col 1 (prev. col 2) = unique track number ID assigned to each cell
% %   col 2 (prev. col 4) = x coordinate (pixels) of centroid
% %   col 3 (prev. col 5) = y coordinate (pixels) of centroid
% %   col 4 (prev. col 6) = absolute time corresponding to frame in which
% %                         cell is found (sec)
% %   col 5 (prev. col 7) = binned time corresponding to frame in which cell
% %                         is found (sec)
% %   col 6 (prev. col 8) = area of cell in pixels
% %   col 7 (prev. col 9) = track change flag. Entry = 1 if start of new
% %                         track (yes) or 0 if no (i.e. continuation of an
% %                         existing track.
% %**************************************************************************
% function [data_edited] = Post_IJ_Manual_Track_v3(data,fid)
% % Get function name:
% func_name = mfilename;
% % Update log file that function is running:
% fprintf(1,"%n%s running ...
",func_name);
% fprintf(fid,"%n%s running ...
",func_name);
% % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% % warning dialog the log file records no errors/warnings generated:
% warn = 0;
% % Determine dimensions of array:
%
[rows cols] = size(data);

% Reserve appropriate memory block:
data_edited = zeros(rows, 7);

% Write data to be saved:
data_edited(:,1) = data(:,2);
data_edited(:,2:7) = data(:,4:9);

% Check dimensionality:
[rows2 cols2] = size(data_edited);
if rows2 ~= rows
    fprintf(1,'\n\tWARNING: # of rows not preserved during editing.\n');
    fprintf(1,'\tOriginal # rows = %.0f, post editing # rows = %.0f, rows, rows2);
    fprintf(fid,'\n\tWARNING: # of rows not preserved during editing.\n');
    fprintf(fid,'\tOriginal # rows = %.0f, post editing # rows = %.0f, rows, rows2);
    warn = 1;
end
if cols2 ~= 7
    fprintf(1,'\n\tWARNING: # of cols not = 7 after editing.\n');
    fprintf(1,'\tInstead # cols = %.0f
',cols2);
    fprintf(fid,'\n\tWARNING: # of cols not = 7 after editing.\n');
    fprintf(fid,'\tInstead # cols = %.0f\n',cols2);
    warn = 1;
end

% Check that track change flags are internally consistent with track IDs:
ID_vector = data_edited(:,1);
TCF_vector = data_edited(:,7);
[junk, ID_ind] = unique(ID_vector, 'first');
clear junk
[TCF_ind] = find(TCF_vector == 1);
check = (ID_ind == TCF_ind);
if length(ID_ind) ~= length(TCF_ind)
    fprintf(1,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
    fprintf(fid,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
    warn = 1;
else
    if nnz(check) ~= length(ID_ind)
        fprintf(1,'\n\tWARNING: Start of each unique track ID not consistent with placement of track change flags.\n');
        fprintf(fid,'\n\tWARNING: Start of each unique track ID not consistent with placement of track change flags.\n');
        warn = 1;
    else
        fprintf(1,'\n\tStart of each unique track ID confirmed consistent w/placement of track change flags.\n');
        fprintf(fid,'\n\tStart of each unique track ID confirmed consistent w/placement of track change flags.\n');
    end
end
% Clear original data array to save memory:
clear data

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'\n\tFunction completed without errors/warnings\n');
    fprintf(fid,'\n\tFunction completed without errors/warnings\n');
end
% Update log file that function is completed:
fprintf(1,'\n%s completed\n',func_name);
fprintf(fid,'\n%s completed\n',func_name);
% PURPOSE:
% This program plots all cell trajectories emanating from a single
% origin.

% ASSUMPTIONS:
% User loads proper file.

% INPUT:
% exp_t_max = maximum experimental time to include in analysis (min)
% pixel_calib = user specified objective calibration (um/pixel)
% data = an array containing all pertinent tracking information for each
% cell in the given experimental condition having the following structure
% col 1 = unique track number ID assigned to each cell
% col 2 = x coordinate (pixels) of centroid
% col 3 = y coordinate (pixels) of centroid
% col 4 = absolute time corresponding to frame in which cell is found
% (sec)
% col 5 = binned time corresponding to frame in which cell is found (sec)
% col 6 = area of cell in pixels
% col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
% if no (i.e. continuation of an existing track.
% run_title = user specified string description of experimental condition
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% mi_counts = 'model independent counts' = number of tracks contributing to
% model independent analysis
% avg_allDisp = mean of the set of all absolute displacements of each
% cell's centroid from its origin.
% std_allDisp = standard deviation of the set of all absolute displacements
% of each cell's centroid from its origin.
% avg_maxDisp = mean of the set of all maximum absolute displacements
% of each cell's centroid from its origin.
% std_maxDisp = standard deviation of the set of all maximum absolute
% displacements of each cell's centroid from its origin.
% Three Matlab figures:
% Trajectories.fig is a plot of all cell trajectories emanating from a
% common origin. Only plotted are those cells that were tracked minimally
% 'exp_t_max' in duration. A MATLAB figure (.fig) containing a plot of cell
% centroid positions
% Trajectories_means.fig is the previous plot but containing two circles.
% One of radius 'avg_allDisp' and the other of radius 'avg_maxDisp'
% Trajectories_hist.fig is a figure containing the histograms of the
% absolute displacements computed and the maximum absolute displacements
% computed.

function [mi_counts, avg_allDisp, std_allDisp, avg_maxDisp, std_maxDisp] = Cell_Track_Plotter_v6(exp_t_max, pixel_calib, data, run_title, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,\n's running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Retrieve number of tracks in filtered 'data' array. Recall at this point
% 'data' array has been processed via 'Post_IJ_Manual_Track_v2.m' and
% possibly 'Filter_Exp_Data.m':
uniq_IDs = unique(data(:,1));
num_tracks = length(uniq_IDs);
max_disp = zeros(num_tracks,1);

% Reserve variable name 'all_disp' that will hold all displacements of each
% track's centroid from its origin position (not just the maximum
disp as stored in 'max_disp' above):
all_disp = [];

% Create a figure to which we will iteratively plot track trajectories:
fig_handle = figure;
axes_handle = axes;

% Create a counter that will keep track of the total number of cells
% plotted on the final graph (note it is not necessarily true that the
% number of cells plotted will be equal to the number of total cells in
% 'data'. To be plotted a track must exist through exp_t_max. In this way we do
% not bias the average by incorporating truncated tracks.
counts = 0;

for i = 1:num_tracks
    track = uniq_IDs(i);
    obs_check = data(:,1)==track & data(:,5)>=exp_t_max*60;
    if sum(obs_check)>0
        track_ind = data(:,1)==track & data(:,5)<=exp_t_max*60;
        if sum(track_ind)>0
           coords = data(track_ind,2:3);
            x_orig = coords(1,1);
y_orig = coords(1,2);
            coords(:,1) = coords(:,1)-x_orig;
            coords(:,2) = coords(:,2)-y_orig;
            coords = coords*pixel_calib;
            % Compute displacement of all centroid positions from
% initial centroid position (origin):
            track_disp = sqrt(coords(:,1).^2+coords(:,2).^2);
            % Record only max displacement observed from list of all
% displacement for this track:
            if isempty(all_disp)==1
                all_disp = track_disp(2:end);
            else
                allDisp(end+1:end+length(track_disp)-1) = track_disp(2:end);
            end
    end
end
max_disp(i) = max(track_disp);

% Plot track trajectory:
hold all
plot(coords(:,1),coords(:,2),'LineStyle','-','Color','k','LineWidth',1.5,'Marker','none','HandleVisibility','off');

% Update counter:
counts = counts + 1;

end

end

% Send counts out of function:
mi_counts = counts;

% Eliminate zero rows
num_del_theory = length(max Disp)-nnz(max Disp);
del_ind = max Disp(:)==0;
max Disp(del_ind)=[];
um_del_practice = sum(del_ind);
if num_del_practice ~= num_del_theory
    fprintf(1,'\n	WARNING: # deletions from "max Disp" ~= # deletions predicted\n');
    warn = 1;
    keyboard
end

% Compute mean and std of 'all Disp' and 'max Disp':
avg_all Disp = mean(all Disp);
std_all Disp = std(all Disp);

avg_max Disp = mean(max Disp);
std_max Disp = std(max Disp);

% Record number cells plotted on trajectory graph:
fprintf(1,'\n	Number of trajectories plotted = %d\n',num2str(counts));
fprintf(fid,'\n	Number of trajectories plotted = %d\n',num2str(counts));

% Record 'all Disp' mean and std:
fprintf(1,'\n\tALL absolute displacement statistics:\n\t	Mean = %0.5f um',avg_all Disp);
fprintf(fid,\n\t\tMean = %0.5f um',avg_all Disp);

% Record 'max Disp' mean and std:
fprintf(1,'\n\tMAX absolute displacement statistics:\n\t	Mean = %0.5f um',avg_max Disp);
fprintf(fid,\n\t\tMean = %0.5f um',avg_max Disp);

% Plot a figure that has both a circle of radius 'avg all Disp' as well
% as a circle of radius 'avg_max Disp' on the same axes. This is
% the master plot from which we will a second without these circles.
plot_avg_all Disp = round(avg_all Disp);
plot_avg_max Disp = round(avg_max Disp);

all_circle_x = plot_avg_all Disp^2-1:1:plot_avg all Disp;
all_circle_y_upper = sqrt(plot_avg all Disp^2-all_circle_x.^2);
all_circle_y_lower = all_circle_y_upper^-1;
max_circle_x = plot_avg_max_disp*-1:.1:plot_avg_max_disp;
max_circle_y_upper = sqrt(plot_avg_max_disp^2-max_circle_x.^2);
max_circle_y_lower = max_circle_y_upper*-1;

hold all
AU = plot(all_circle_x,all_circle_y_upper,'LineStyle','--','Color','b','LineWidth',2,'Marker','none','HandleVisibility','on');
hold all
AL = plot(all_circle_x,all_circle_y_lower,'LineStyle','--','Color','b','LineWidth',2,'Marker','none','HandleVisibility','off');
hold all
MU = plot(max_circle_x,max_circle_y_upper,'LineStyle','--','Color','r','LineWidth',2,'Marker','none','HandleVisibility','on');
hold all
ML = plot(max_circle_x,max_circle_y_lower,'LineStyle','--','Color','r','LineWidth',2,'Marker','none','HandleVisibility','off');

% Annotation
title(run_title,'FontName','Arial','FontSize',18);
xlabel('x position (\mum)','FontName','Arial','FontSize',18);
ylabel('y position (\mum)','FontName','Arial','FontSize',18);
set(axes_handle,'FontName','Arial');
set(axes_handle,'FontSize',16);
set(axes_handle,'DataAspectRatio',[1 1 1]);
h_legend = legend('<|\Delta r|>','<Max(|\Delta r|)>','Location','NorthEast');
set(h_legend,'FontName','Arial');
set(h_legend,'FontSize',14);

% Get automatically generated axes limits and adjust so square:
v = axis;
lim = max(abs(v));
axis([lim*-1 lim lim*-1 lim]);

% Generate base filename:
fig_title = 'Trajectories';

% Save figure with mean circles:
saveas(fig_handle, [fig_title '_means.fig'], 'fig');

% Eliminate means and save:
delete(AU);
delete(AL);
delete(MU);
delete(ML);
delete(h_legend);
saveas(fig_handle, [fig_title '.fig'], 'fig');

% Plot histogram figure
num_all_bins = round(sqrt(length(all_disp)));
num_max_bins = round(sqrt(length(max_disp)));

h1 = subplot(1,2,1);
[freq bin_loc] = hist(all_disp,num_all_bins);
bar(bin_loc,freq,1);
axis([min(all_disp) max(all_disp) 0 max(freq)]);
title('All |\Delta r|','FontName','Arial','FontSize',18);
xlabel('|\Delta r| (\mum)','FontName','Arial','FontSize',18);
ylabel('Count','FontName','Arial','FontSize',18);
set(h1,'FontName','Arial','FontSize',16);

h2 = subplot(1,2,2);
[freq bin_loc] = hist(max_disp,num_max_bins);
bar(bin_loc,freq,1);
axis([min(max_disp) max(max_disp) 0 max(freq)]);
title('Max |\Delta r|','FontName','Arial','FontSize',18);
xlabel('|\Delta r| (\mum)','FontName','Arial','FontSize',18);
ylabel('Count','FontName','Arial','FontSize',18);
%**************************************************************************
% PURPOSE:
% This function computes the absolute differentials in displacement between
% two consecutive frames of all tracked objects.
% %
% % ASSUMPTIONS:
% % n/a
% %
% % INPUT:
% % data = array having following structure:
% %   col 1 = unique track number ID assigned to each cell
% %   col 2 = x coordinate (pixels) of centroid
% %   col 3 = y coordinate (pixels) of centroid
% %   col 4 = absolute time corresponding to frame in which cell is found
% %          (sec)
% %   col 5 = binned time corresponding to frame in which cell is found (sec)
% %   col 6 = area of cell in pixels
% %   col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
% %          if no (i.e. continuation of an existing track.
% % fid = file ID of log file to which progress is recorded
% %
% % OUTPUT:
% % CAD = ("consecutive absolute differentials") array containing absolute
% % displacement differentials between consecutive frames across all tracks
% % in a given 'data' array. Data from frames separated by an interval of
% % time greater than the attempted constant imaging rate are excluded.
% % col 1 = track ID to which this interval belongs
% % col 2 = dx (absolute displacement along x coordinate axis in pixels)
% % col 3 = dy (absolute displacement along y coordinate axis in pixels)
% % col 4 = elapsed experimental imaging time at end of corresponding
% %       displacement (sec)
% % col 5 = elapsed binned time (sec) between consecutive frames (i.e.
% %       tau_min)
% %**************************************************************************

function [CAD] = Consec_Differentials_v4(data, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,'\n%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;
% Determine indices of start and stop positions of each unique track in the
% overall 'data' array:
[junk ind_start] = unique(data(:,1),'first');
clear junk
[junk ind_stop] = unique(data(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,\'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n\');
    fprintf(fid,\'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n\');
    warn = 1;
end

% Total number of tracks to be analyzed:
num_tracks = length(ind_start);

% Reserve 'CAD' variable name:
CAD = [];

% Reserve 'num_consec_int_tot' and 'num_dels_tot' for computation of total
% number of consecutive intervals computed:
num_consec_int_tot = 0;
num_del_tot = 0;

% Begin 08/28/2011 Version 3 edit in determination of constant imaging rate
% 'im_rate':
% Generate a vector of size length(data) - num_tracks to hold total number
% of adjacent frame time differences:
num_adj_ints = size(data,1)-num_tracks;
bin_time_diff_list = zeros(num_adj_ints,1);
print_row = 1;

% Loop over tracks:
for ii = 1:num_tracks

    % Define start and stop position (row #s):
    r_start = ind_start(ii);
    r_stop = ind_stop(ii);

    % Loop over all rows between start and second-to-last stop rows and
    % compute absolute displacements for track
    for kk = r_start:r_stop-1
        bin_time_now = data(kk,5);
        bin_time_next = data(kk+1,5);
        bin_time_diff = bin_time_next-bin_time_now;
        bin_time_diff_list(print_row) = bin_time_diff;
        print_row = print_row+1;
    end

end
im_rate = mode(bin_time_diff_list);
% End version 3 edits

% Loop over tracks:
for i = 1:num_tracks

    % Define start and stop position (row #s):
    r_start = ind_start(i);
    r_stop = ind_stop(i);

    % Loop over all rows between start and second-to-last stop rows and
    % compute absolute displacements for track
    for kk = r_start:r_stop-1
        bin_time_now = data(kk,5);
        bin_time_next = data(kk+1,5);
        bin_time_diff = bin_time_next-bin_time_now;
        bin_time_diff_list(print_row) = bin_time_diff;
        print_row = print_row+1;
    end
end
im_rate = mode(bin_time_diff_list);
% End version 3 edits

% Loop over tracks:
for i = 1:num_tracks

    % Define start and stop position (row #s):
    r_start = ind_start(i);
    r_stop = ind_stop(i);

    % Determine the maximum number of consecutive intervals that could
% possibly be observed for this track:
num_consec_int_track = r_stop - r_start;
track_CAD = zeros(num_consec_int_track,5);
num_consec_int_tot = num_consec_int_tot + num_consec_int_track;
print_row = 1;

% Loop over all rows between start and second-to-last stop rows and
% compute absolute displacements for track
for k = r_start:r_stop-1
    bin_time_now = data(k,5);
    bin_time_next = data(k+1,5);
    bin_time_diff = bin_time_next-bin_time_now;

    if bin_time_diff == im_rate
        orig_row = k;
        adv_row = k+1;

        % Load origin info:
        track_ID = data(orig_row,1);
        x_orig = data(orig_row,2);
        y_orig = data(orig_row,3);
        t_bin_orig = data(orig_row,5);

        % Load advance row info:
        x_adv = data(adv_row,2);
        y_adv = data(adv_row,3);
        t_bin_adv = data(adv_row,5);

        % Compute differentials:
        dx = x_adv - x_orig;
        dy = y_adv - y_orig;
        dt_bin = t_bin_adv - t_bin_orig;

        % Determine absolute experimental time at end of this
        % displacement:
        abs_exp_t = data(adv_row,4);

        % Log values:
        track_CAD(print_row,1) = track_ID;
        track_CAD(print_row,2) = dx;
        track_CAD(print_row,3) = dy;
        track_CAD(print_row,4) = abs_exp_t;
        track_CAD(print_row,5) = dt_bin;

        % Advance print_row:
        print_row = print_row+1;
    end
end

% Remove rows not utilized in 'track_CAD' because time elapsing between
% frames corresponding to that row entry was greater than 'im_rate':
del_ind = track_CAD(:,5)==0;
num_del_track = sum(del_ind);
if num_del_track>0
    track_CAD(del_ind,:) = [];
    num_del_tot = num_del_tot+num_del_track;
end

% After all consecutive intervals for 'track_ID' are computed make sure
% the length of 'track_CAD' is = 'num_consec_int_track - num_del_track':
if track_CAD_r >= num_consec_int_track-num_del_track
fprintf(1,'\n\tWARNING: # consec intervals computed ~= expected #in');
fprintf(1,'\n\tWarning generated for track = %.0f',track_ID);
fprintf(fid,'\n\tWARNING: # consec intervals computed ~= expected #in');
warn = 1;
end

% Concatenate 'track_CAD' with existing 'CAD'
if isempty(CAD) == 1
    CAD = track_CAD;
    clear track_CAD
    % In the event only 1 track exists in this 'data' array we want to
    % have the dimensions for the post-processing dimensionality check.
    [CAD_new_r CAD_new_c] = size(CAD);
else
    CAD_old = CAD;
    [CAD_old_r CAD_old_c] = size(CAD_old);
    clear CAD
    CAD = zeros(CAD_old_r+track_CAD_r,CAD_old_c);
    CAD(1:CAD_old_r,1:track_CAD_c) = CAD_old;
    CAD(CAD_old_r+1:CAD_old_r+track_CAD_r,1:track_CAD_c) = track_CAD;
    [CAD_new_r CAD_new_c] = size(CAD);
end

% Dimensionality check. There should be five columns after the
% concatenations are finished:
if CAD_new_c ~= 5
    fprintf(1,'\n\tWARNING: # cols = %.0f not 5 as expected\n',CAD_new_c);
    fprintf(1,'\n\tWarning generated for track = %.0f',track_ID);
    fprintf(fid,'\n\tWARNING: # cols = %.0f not 5 as expected\n',CAD_new_c);
    fprintf(fid,'\n\tWarning generated for track = %.0f',track_ID);
    warn = 1;
end

% Dimensionality check. The total number of rows in 'CAD' should be equal
% to the sum of each track's previously identified consecutive intervals in
% theory minus the number of deletions made in practice:
num_consec_int_tot_actual = num_consec_int_tot - num_del_tot;
if CAD_new_r ~= num_consec_int_tot_actual
    fprintf(1,'\n\tWARNING: # rows (consec_intervals) = %.0f not %.0f as
expected\n',CAD_new_r,num_consec_int_tot_actual);
    fprintf(1,'\n\tWARNING: # rows (consec_intervals) = %.0f not %.0f as
expected\n',CAD_new_r,num_consec_int_tot_actual);
    warn = 1;
end

% Save 'CAD' array in .mat and .txt format:
save 'CAD.mat' CAD;
fid2 = fopen('CAD.txt','wt');
fprintf(fid2,\nTrack	dx (pix)	dy (pix)	Abs. Exp. t (sec)	ttau (sec)\n');
for k = 1:CAD_new_r
    fprintf(fid2,\n%.0f	%f	%f	%.0f	%.0f\n',CAD(k,:));
end fclose(fid2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'\n\tFunction completed without errors/warnings\n');
    fprintf(fid,'\n\tFunction completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(1,’\n%s completed\n’,func_name);
fprintf(fid,’\n%s completed\n’,func_name);
Step_Size_Stationarity_v2.m

1 % Steven J. Henry
2 % 08/11/2012
3 %**************************************************************************
4 % PURPOSE:
5 % This function plots the mean step size (in x and y) of a population of
6 % cells as a function of experimental time:
7 % <delta_x(tau_min)> vs. experimental time
8 % <delta_y(tau_min)> vs. experimental time
9 %**************************************************************************
10 function [SSD, tau_min, t_max] = Step_Size_Stationarity_v2(CAD, run_title, fid)
11 % Get function name:
12 func_name = mfilename;
13
14 % Update log file that function is running:
15 fprintf(1,'%n%s running ...
',func_name);
16 fprintf(fid,'%n%s running ...
',func_name);
17
18 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
19 % warning dialog the log file records no errors/warnings generated:
20 warn = 0;
21
22 % Determine 'tau_min' value:
23 tau_min = unique(CAD(:,5)); % sec
24 % All data residing in CAD should be for the same 'tau_min' value. If multiple
25 % 'tau_min' values are returned from the above operation then there is
26 % more/different data in CAD than you think there is so tell user:
27 if length(tau_min)>1
28 %**************************************************************************
29 %**************************************************************************
30 %**************************************************************************
fprintf(1,\"WARNING: multiple tau_min values in column 5 of CAD array\n\");
fprintf(fid,\"WARNING: multiple tau_min values in column 5 of CAD array\n\");
warn = 1;
else
fprintf(1,\"\n\tPlots in terms of tau_min = %.0f sec\n\",tau_min);
fprintf(fid,\"\n\tPlots in terms of tau_min = %.0f sec\n\",tau_min);
end

% Place absolute experimental time (column 4 of CAD) in terms of multiples
% of 'tau_min'. Note unlike function 'Bin_Time_v2.m' which takes care to account
% for upwards rounding bias when quotients of exactly 0.5 are generated,
% for the purposes of this program such bias is not of major concern.
CAD(:,4) = round(CAD(:,4)/tau_min)*tau_min;

% Determine unique absolute experimental times observed:
abs_exp_t = unique(CAD(:,4));
abs_exp_t = sort(abs_exp_t,'ascend');
num_abs_exp_t = length(abs_exp_t);

% Loop over all unique experimental time entries:
for i = 1:num_abs_exp_t

% Load present experimental time:
time = abs_exp_t(i);
% Determine indices of those entries in CAD that correspond to
% observations of steps over an elapsed period 'tau_min' at the elapsed
% experimental imaging time 'time'
ind = CAD(:,4)==time;
% Extract the corresponding steps in x and y:
dx = abs(CAD(ind,2));
dy = abs(CAD(ind,3));
% Compute the number of observations of cell steps over an elapsed
% period 'tau_min' at the elapsed experimental imaging time 'time'
reps = length(dx);

% Consistency check. If dr(i)^2 = dx(i)^2 + dy(i)^2 then there should
% always be a pair dx and dy. Check this fact.
if length(dx)~=length(dy)
    fprintf(1,\"\n\tWARNING: # dx ~= # dy for experimental time = %.0f\n\",time);
    fprintf(fid,\"\n\tWARNING: # dx ~= # dy for experimental time = %.0f\n\",time);
end

% Compute means and standard deviations:
m_dx = mean(dx);
std_dx = std(dx);
m_dy = mean(dy);
std_dy = std(dy);

% Log results in appropriate column and row:
SSD(i,1) = time;
SSD(i,2) = m_dx;
SSD(i,3) = std_dx;
SSD(i,4) = m_dy;
SSD(i,5) = std_dy;
SSD(i,6) = reps;
end

% Rename columns of SSD for ease of plotting:
% minutes
m_X = SSD(:,2); % pix
std_X = SSD(:,3); % pix
m_Y = SSD(:,4); % pix
std_Y = SSD(:,5); % pix

% Find the min and max ordinate values across both m_X and m_Y sets of data % so we can configure identical axes on each subplot:
yaxis_min = zeros(1,2);
yaxis_min(1) = min(m_X-std_X);
yaxis_min(2) = min(m_Y-std_Y);
yaxis_min = min(yaxis_min);
if yaxis_min < 0
    yaxis_min = 0;
end

yaxis_max = zeros(1,2);
yaxis_max(1) = max(m_X+std_X);
yaxis_max(2) = max(m_Y+std_Y);
yaxis_max = max(yaxis_max);

% Find max experimental time value:
xaxis_min = 0;
xaxis_max = max(T);
t_max = xaxis_max;

% First plot will include standard deviations:
h1 = figure;
sub_h1(1) = subplot(1,2,1);

errorbar(T,m_X,std_X,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold all;
sub_h1(2) = subplot(1,2,2);

errorbar(T,m_Y,std_Y,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold all;

set(sub_h1,'ylim',yaxis_min yaxis_max);
set(sub_h1,'xlim',xaxis_min xaxis_max);
set(sub_h1,'FontName','Arial');
set(sub_h1,'FontSize',14);

% Save figure
saveas(h1, 'Stationarity_with_std.fig','fig');

% Second plot will not include standard deviations:
h2 = figure;
sub_h2(1) = subplot(1,2,1);

plot(T,m_X,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold all;
sub_h2(2) = subplot(1,2,2);

plot(T,m_Y,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold all;

set(sub_h2,'ylim',yaxis_min yaxis_max);
set(sub_h2,'xlim',[xaxis_min xaxis_max]);
set(sub_h2,'FontName','Arial');
set(sub_h2,'FontSize',14);

h2_x_axis_handles = cell2mat(get(sub_h2,'xlabel'));
set(h2_x_axis_handles,'String','Elapsed Experimental Time (min)',
     'FontName','Arial','FontSize',14);

h2_y_axis_handles = cell2mat(get(sub_h2,'ylabel'));
set(h2_y_axis_handles(1),'String','<|\Delta x(\tau)|> (pix)',
     'FontName','Arial','FontSize',14);
set(h2_y_axis_handles(2),'String','<|\Delta y(\tau)|> (pix)',
     'FontName','Arial','FontSize',14);

h2_title_handles = cell2mat(get(sub_h2,'title'));
set(h2_title_handles,'String',{run_title,['\tau = ' num2str(tau_min) ' sec']},
     'FontName','Arial','FontSize',14);

% Save figure
saveas(h2, 'Stationarity.fig','fig');

% Compute average number of replicates used in computation of mean and
% standard deviation:
avg_reps = round(mean(SSD(:,6)));
fprintf(1,'
	Average # replicates used to compute mean and s.d. = %.0f
',avg_reps);
fprintf(fid,'
	Average # replicates used to compute mean and s.d. = %.0f
',avg_reps);

% Save 'SSD' array in .mat and .txt format:
save 'SSD.mat' SSD;
fid2 = fopen('SSD.txt','wt');
fprintf(fid2,'Exp. t (sec)	<|dx|> (pix)	 dx s.d. (pix)	<|dy|> (pix)	 dy s.d. (pix)	# Replicates
');
for k = 1:num_abs_exp_t
    fprintf(fid2,'%.0f	%f	%f	%f	%f	%.0f
',SSD(k,:));
end
close(fid2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings
');
    fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(1,'
%s completed
',func_name);
fprintf(fid2,';','');
end

% This function plots:
% (1)Histograms of the fractional part of x-position of the particles and
% y-position of the particles (x mod 1, y mod 1)
% (2)Histograms of the absolute displacements of cells between consecutive
% frames (dx and dy)
% These are constructed to determine sample validity consistent with
% Crocker and Hoffman's "Multiple-Particle Tracking and Two-Point
% n/a
% ASSUMPTIONS:
% n/a
% INPUT:
% data = array having following structure:
% col 1 = unique track number ID assigned to each cell
% col 2 = x coordinate (pixels) of centroid
% col 3 = y coordinate (pixels) of centroid
% col 4 = absolute time corresponding to frame in which cell is found
% (sec)
% col 5 = binned time corresponding to frame in which cell is found (sec)
% col 6 = area of cell in pixels
% col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
% if no (i.e. continuation of an existing track.
% CAD = ("consecutive absolute differentials") array containing absolute
% displacement differentials between consecutive frames across all tracks
% in a given 'data' array. Data from frames separated by an interval of
% time greater than the attempted constant imaging rate are excluded.
% col 1 = track ID to which this interval belongs
% col 2 = dx (absolute displacement along x coordinate axis in pixels)
% col 3 = dy (absolute displacement along y coordinate axis in pixels)
% col 4 = elapsed experimental imaging time at end of corresponding
% displacement (sec)
% col 5 = elapsed binned time (sec) between consecutive frames (i.e. tau)
% tau_min = elapsed time interval between most consecutive frames (i.e.
% attempted constant imaging rate) (sec)
% run_title = user specified string description of experimental condition
% fid = file ID of log file to which progress is recorded
%
% OUTPUT:
% n/a
%**************************************************************************

function [] = Histograms_v3(data, CAD, tau_min, run_title, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

%**********************************
% Check for systemic error in centroid position:
% the particles (x mod 1, y mod 1):
% mod = mod(data(:,2),1):
% ymod = mod(data(:,3),1):
% num_pos = length(xmod):
% num_bins = round(sqrt(num_pos));
% Record number of samples and bins used to plot data:
% fprintf(1,'\n\nNumber of positions (samples) = %.0fn\n', num_pos):
% fprintf(1,'\n\nNumber of bins = %.0fn, num_bins):
% fprintf(1,'\n\nDefault computation of bins is sqrt(# samples)\n');
% fprintf(fid,'\n\nNumber of positions (samples) = %.0fn\n', num_pos):
% fprintf(fid,'\n\nNumber of bins = %.0fn, num_bins):
% fprintf(fid,'\n\nDefault computation of bins is sqrt(# samples)\n');
% Start figure:
% h1 = figure;
% for i = 1:2
% if i == 1
% % Load x mod 1 values:
% samples = xmod;
% % Prepare plot:
subplot(1,2,1);
hold all
title(run_title,'FontName','Arial','FontSize',14);
xlabel('x mod 1 (pix)','FontName','Arial','FontSize',14);
ylabel('Counts','FontName','Arial','FontSize',14);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
elseif i == 2
% Load y mod 1 values:
samples = ymod;
subplot(1,2,2);
hold all
title(run_title,'FontName','Arial','FontSize',14);
xlabel('y mod 1 (pix)','FontName','Arial','FontSize',14);
ylabel('Counts','FontName','Arial','FontSize',14);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
end

% Use intrinsic MATLAB 'hist' function to determine number of samples
% ('counts') that fall into a given bin with centers 'bin_loc' using a
% total number of bins equal to 'num_bins':
[counts bin_loc] = hist(samples, num_bins);

% Plot histogram using a bar graph to make bin widths visually equal.
% The scalar 1 allows bars to touch.
bar(bin_loc, counts, 1);

% Figure out y_max boundary so both plots can have the same y axis
% bounds
if i == 1
y_max = max(counts);
elseif i == 2
if max(counts) > y_max
y_max = max(counts);
end
end
clear counts bin_loc
end

% Set y-limits on both subplots:
subplot(1,2,1);
set(gca,'ylim',[0 y_max]);
subplot(1,2,2);
set(gca,'ylim',[0 y_max]);

% Save x mod 1 and y mod 1 figure:
saveas(h1, 'xy_mod_1.fig','fig');

% Now plot distribution of displacements associated with consecutive
% frames:

% Load data and determine total number of consecutive intervals (# samples)
% computed:
dx = CAD(:,2);
dy = CAD(:,3);
num_samples = length(dx);

% Determine number of bins. A common approach is to use sqrt(num_samples)
% number of bins when you suspect the underlying pdf is gaussian. Here the
% total number of samples 'num_samples' is equal to the number of rows in
% 'CAD'.
num_bins = round(sqrt(num_samples));
% Record number of samples and bins used to plot data:
fprintf(1,'\n\nFor dx and dy histograms:\n\nNumber differentials from consec. frames (samples) = %.0f
', num_samples);
fprintf(1,'\nNumber of bins = %0.f
', num_bins);
fprintf(1,'\nDefault computation of bins is sqrt(# samples)\n\nPlots in terms of tau_min = %.0f sec
',tau_min);
fprintf(fid,'\n\nFor dx and dy histograms:
\nNumber differentials from consec. frames (samples) = %.0f
', num_samples);
fprintf(fid,'\nNumber of bins = %0.f
', num_bins);
fprintf(fid,'\nDefault computation of bins is sqrt(# samples)\n\nPlots in terms of tau_min = %.0f sec
',tau_min);

% Find the max and min delta terms across both dx and dy sets of data so we
% can configure identical axes on each subplot:
d_min = zeros(1,2);
d_min(1) = min(dx);
d_min(2) = min(dy);
d_max = zeros(1,2);
d_max(1) = max(dx);
d_max(2) = max(dy);
d_max = max(d_max);
d_min = min(d_min);

% Generate bin_loc vector which is a vector with length = num_bins which
% are linearly spaced bin centers between and including 'd_min' and
% 'd_max':
bin_loc = linspace(d_min,d_max,num_bins);

% Generate 'x_counts' and 'y_counts' vectors to hold frequency data:
x_counts = hist(dx, bin_loc);
y_counts = hist(dy, bin_loc);

% Start figure:
figure;

% Plot 'dx' results:
bar(bin_loc, x_counts, 1);
hold all;

% Plot 'dy' results:
bar(bin_loc, y_counts, 1);
hold all;

% Determine maximum number of counts to set y-lim on both subplots:
y_max = zeros(2,1);
y_max(1) = max(x_counts);
y_max(2) = max(y_counts);
y_max = max(y_max);

% Make abscissa bounds symmetrical about zero:
d_lim = zeros(1,2);
d_lim(1) = abs(d_max);
d_lim(2) = abs(d_min);
d_lim = max(d_lim);

set(sub_h,'ylim',[0 y_max]);
set(sub_h,'xlim',[d_lim*-1 d_lim]);
set(sub_h,'yscale','log');
set(sub_h,'YMinorTick','on');
set(sub_h,'TickDir','out');
set(sub_h,'FontName','Arial');
set(sub_h,'FontSize',14);
x_axis_handles = cell2mat(get(sub_h,'xlabel'));
set(x_axis_handles(1),'String','\Delta x(\tau) (pix)','FontName','Arial','FontSize',14);
```matlab
set(x_axis_handles(2), 'String', '
\Delta y(\tau) (\text{pix})', 'FontName', 'Arial', 'FontSize', 14);
```

```matlab
deltay(\tau) (\text{pix})
```

```matlab
y_axis_handles = cell2mat(get(sub_h,'ylabel'));
```

```matlab
set(y_axis_handles(1), 'String', 'Counts', 'FontName', 'Arial', 'FontSize', 14);
```

```matlab
set(y_axis_handles(2), 'String', 'Counts', 'FontName', 'Arial', 'FontSize', 14);
```

```matlab
title_handles = cell2mat(get(sub_h,'title'));
```

```matlab
set(title_handles, 'String', ['\tau = ' num2str(tau_min) ' \text{sec}'], 'FontName', 'Arial', 'FontSize', 14);
```

```matlab
% Save dx and dy figure:
saveas(h2, 'Histogram.fig','fig');
```

```matlab
% If no warnings generated report so in log file:
if warn == 0
    fprintf(1, '
\text{Function completed without errors/warnings}
');
    fprintf(fid, '
\text{Function completed without errors/warnings}
');
end
```

```matlab
% Update log file that function is completed:
fprintf(1, '
%s completed
', func_name);
fprintf(fid, '
%s completed
', func_name);
end
```

---

**Path_Length_v6.m**

```matlab
1 % Steven J. Henry
2 % 06/09/2011
3 %**************************************************************************
4 % PURPOSE:
5 % This function plots the cummulative path length of each cell over elapsed
6 % experimental imaging time and saves the results.
7 %
8 % ASSUMPTIONS:
9 % n/a
10 % INPUT:
11 % data = array having following structure:
12 %   col 1 = unique track number ID assigned to each cell
13 %   col 2 = x coordinate (pixels) of centroid
14 %   col 3 = y coordinate (pixels) of centroid
15 %   col 4 = absolute time corresponding to frame in which cell is found
16 %   col 5 = binned time corresponding to frame in which cell is found (sec)
17 %   col 6 = area of cell in pixels
18 %   col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
19 % if no (i.e. continuation of an existing track.
20 % pixel_calib = user-supplied microns/pixel
21 % run_title = user specified string description of experimental condition
22 % fid = file ID of log file to which progress is recorded
23 % OUTPUT:
24 % Path_Length = array of cummulative path length data with following
25 % structure:
26 %   col 1 = track ID
27 %   col 2 = cummulative euclidean distance (pixels)
28 %   col 3 = absolute elapsed time (sec)
29 %   col 4 = binned elapsed time (sec)
30 %   col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
31 % if no (i.e. continuation of an existing track.
32 %**************************************************************************
33
34 function [Path_Length] = Path_Length_v6(data, pixel_calib, run_title, fid)
35
36 % Get function name:
37 func_name = mfilename;
38
39 % Update log file that function is running:
40 fprintf(1, '\text{Function running ...}\n', func_name);
41 fprintf(fid, '\text{Function running ...}\n', func_name);
```

---

250
% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Determine number of rows:
[rows] = size(data);
rows = rows(1);
% Reserve memory block for 'Path_Length' array:
Path_Length = zeros(rows,5);
% Transcribe track IDs and track change flags:
Path_Length(:,1) = data(:,1);
Path_Length(:,5) = data(:,7);

% Determine indices of start and stop positions of each unique track in the
% overall 'data' array:
[junk ind_start] = unique(data(:,1),'first');
clear junk
[junk ind_stop] = unique(data(:,1),'last');
clear junk
% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'
	WARNING: # Unique Track IDs start positions ~= # stop positions\n');
    fprintf(fid,'
	WARNING: # Unique Track IDs ~= # Track Change Flags\n');
    warn = 1;
end
% Total number of tracks to be analyzed:
um_tracks = length(ind_start);

% Loop over tracks:
for i = 1:num_tracks
    % Define start and stop position:
    r_start = ind_start(i);
r_stop = ind_stop(i);
    % Loop over all rows between start and stop rows:
    for ii = r_start:r_stop
        % If index is on start row:
        if ii == r_start
            % All elapsed values (accumulated distance, absolute time, and
            % binned time) are zero:
            Path_Length(ii,2:4) = 0;
        else
            % Otherwise if you are on a non-start row:
            x_prior = data(ii-1,2);
y_prior = data(ii-1,3);
t_abs_prior = data(ii-1,4);
t_bin_prior = data(ii-1,5);
            % Load info of cell (track) in previous frame:
            x_now = data(ii,2);
y_now = data(ii,3);
t_abs_now = data(ii,4);
t_bin_now = data(ii,5);
            % Compute euclidean distance between these two frames:
dx = x_now - x_prior;
dy = y_now - y_prior;
L = sqrt(dx^2+dy^2);
% Compute elapsed time between these two frames:
% Log as the cumulative elapsed path length and time travel as
% the elapsed distance and time between these two frames to the
% previous cumulative values. Note: in essence we are computing our
% sums by adding the differential of each
Path_Length(ii,2) = Path_Length(ii-1,2)+L;
Path_Length(ii,3) = Path_Length(ii-1,3)+dt_abs;
Path_Length(ii,4) = Path_Length(ii-1,4)+dt_bin;
end
end
end

% Plot results:
for j = 1:num_tracks
% Define start and stop position:
r_start = ind_start(j);
r_stop = ind_stop(j);

% Create vectors of track data:
PL = Path_Length(r_start:r_stop,2);
t_abs = Path_Length(r_start:r_stop,3);
t_bin = Path_Length(r_start:r_stop,4);

% Convert distances (in pixels) to microns. Note: 'pixel_calib' is supplied in microns/pixel
PL = PL*pixel_calib;

% Convert time vectors to minutes from seconds:
t_abs = t_abs/60;
t_bin = t_bin/60;

% Plot accumulated distance 'PL' vs. elapsed time (absolute)
% Units are microns and minutes
subplot(1,2,1)
hold on
plot(t_abs,PL,'LineStyle','-','Marker','none','LineWidth',1,'Color','b');

% Plot accumulated distance 'PL' vs. elapsed time (binned)
% Units are microns and minutes
subplot(1,2,2)
hold on
plot(t_bin,PL,'LineStyle','-','Marker','none','LineWidth',1,'Color','b');
end
% Set ordinate upperbound as maximum cumulative path length value:
L_max = max(Path_Length(:,2))*pixel_calib; % microns

% Set abscissa upperbound as maximum elapsed experimental imaging time
% value:
t_max = zeros(2,1);
t_max(1) = max(data(:,4)); % sec
% Set ordinate upperbound as maximum cumulative path length value:
L_max = max(Path_Length(:,2))*pixel_calib; % microns
% Set abscissa upperbound as maximum elapsed experimental imaging time
% value:
t_max = zeros(2,1);
t_max(1) = max(data(:,4)); % sec
% Set ordinate upperbound as maximum cumulative path length value:
L_max = max(Path_Length(:,2))*pixel_calib; % microns
% Set abscissa upperbound as maximum elapsed experimental imaging time
% value:
t_max = zeros(2,1);
t_max(1) = max(data(:,4)); % sec
% Set ordinate upperbound as maximum cumulative path length value:
L_max = max(Path_Length(:,2))*pixel_calib; % microns
% Set abscissa upperbound as maximum elapsed experimental imaging time
% value:
t_max = zeros(2,1);
t_max(1) = max(data(:,4)); % sec

% Label axes:
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial','FontSize',14);
axis([0 t_max 0 L_max]);

subplot(1,2,2)
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
axis([0 t_max 0 L_max]);

% Save figure window generated:
saveas(h, 'Path_Length.fig', 'fig');

% Save 'Path_Length' array in .mat and .txt format:
save 'Path_Length.mat' Path_Length
fid2 = fopen('Path_Length.txt','wt');
fprintf(fid2,'ID	PL (pix)	Abs dt (sec)	Bin dt (sec)	TrackChange
');
for k=1:rows
fprintf(fid2,'%.0f	%f	%.0f	%.0f	%.0f
',Path_Length(k,:));
end
fclose(fid2);

% If no warnings generated report so in log file:
if warn == 0
fprintf(1,'
	Function completed without errors/warnings
');
fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(fid,'%n%s completed
',func_name);

end

Mean_Path_Length_v5.m

% Steven J. Henry
% 08/11/2012
%**************************************************************************
% PURPOSE:
% This function plots the mean (ensemble averaged) cumulative path length
% of each cell over elapsed experimental imaging time and saves the
% results.
% INPUT:
% PathLength = array of cumulative path length data with following
% structure:
%    col 1 = track ID
%    col 2 = cumulative euclidean distance (pixels)
%    col 3 = absolute elapsed time (sec)
%    col 4 = binned elapsed time (sec)
%    col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
% if no (i.e. continuation of an existing track.
% pixel_calib = user-supplied microns/pixel
% run_title = user specified string description of experimental condition
% fid = file ID of log file to which progress is recorded
% OUTPUT (saved but not passed to main driver):
% MeanPathLengthTabs = array of mean cell path length with following
function [] = Mean_Path_Length_v5(Path_Length, pixel_calib, run_title, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Identify all elapsed time intervals that have been observed (using both
% absolute and binned time intervals):
t_abs_uniq = unique(Path_Length(:,3));
t_bin_uniq = unique(Path_Length(:,4));

for i = 1:2
    if i == 1 % Work with absolute time values
        t_uniq = t_abs_uniq;
        t_all = Path_Length(:,3);
    elseif i == 2 % Work with binned time values
        t_uniq = t_bin_uniq;
        t_all = Path_Length(:,4);
    end

    num_uniq = length(t_uniq);
    Mean_Path_Length = zeros(num_uniq,4);

    for ii = 1:num_uniq
        dt = t_uniq(ii);
        ind = t_all == dt;
        PL_dt = Path_Length(ind,2);
        PL_mean_dt = mean(PL_dt);
        PL_std_dt = std(PL_dt);
        log mean path length, standard deviation, the number of
% observations used in the mean calculation, and the elapsed time
% interval:
Mean_Path_Length(ii,1) = PL_mean_dt;
Mean_Path_Length(ii,2) = PL_std_dt;
Mean_Path_Length(ii,3) = sum(ind);
Mean_Path_Length(ii,4) = dt;
end
if i == 1
    Mean_Path_Length_tabs = Mean_Path_Length;
clear Mean_Path_Length
elseif i == 2
    Mean_Path_Length_tbin = Mean_Path_Length;
clear Mean_Path_Length
end
end
% Determine indices of start and stop positions of each unique track in the
% 'Path_Length' array:
[junk ind_start] = unique(Path_Length(:,1),'first');
clear junk
[junk ind_stop] = unique(Path_Length(:,1),'last');
clear junk
% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'

% Total number of tracks to be analyzld:
num_tracks = length(ind_start);
% Plot overlay figure:
h = figure;
for j = 1:num_tracks
    % Define start and stop position:
    r_start = ind_start(j);
    r_stop = ind_stop(j);
    % Create vectors of track data:
    PL = Path_Length(r_start:r_stop,2);
    t_abs = Path_Length(r_start:r_stop,3);
    t_bin = Path_Length(r_start:r_stop,4);
    % Convert length (in # pixels) to microns. Note: 'pixel_calib' has
    % units micron/pixel
    PL = PL*pixel_calib;
    % Convert time vectors to minutes from seconds:
    t_abs = t_abs/60;
    t_bin = t_bin/60;
    % Plot path length vs. elapsed time (absolute)
    subplot(1,2,1)
    hold on
    plot(t_abs,PL,'LineStyle','none','Marker','.','LineWidth',1,'Color','b','HandleVisibility','off');
% Set ordinate upperbound as maximum mean cumulative path length value:
201   PL_max = max(Path_Length(:,2))*pixel_calib; % microns
202
203 % Set abscissa upperbound as maximum elapsed experimental imaging time
204 % value:
205   t_max = zeros(2,1);
206   t_max(1) = max(t_abs); % min
207   t_max(2) = max(t_bin); % min
208   t_max = max(t_max); % min
209
210 % Label axes and overlay mean series:
211   subplot(1,2,1)
212   title(run_title,'FontName','Arial','FontSize',16);
213   xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
214   ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
215   set(gca,'FontName','Arial');
216   set(gca,'FontSize',14);
217   h_legend = legend('Mean Path Length','Location','NorthWest');
218   set(h_legend,'FontName','Arial');
219   set(h_legend,'FontSize',12);
220   axis([0 t_max 0 PL_max]);
221
222   subplot(1,2,2)
223   title(run_title,'FontName','Arial','FontSize',16);
224   xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
225   ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
226   set(gca,'FontName','Arial');
227   set(gca,'FontSize',14);
228   h_legend = legend('Mean Path Length','Location','NorthWest');
229   set(h_legend,'FontName','Arial');
230   set(h_legend,'FontSize',12);
231   axis([0 t_max 0 PL_max]);
232
233 % Save figure window generated:
234   saveas(h, 'Mean_Path_Length_Overlay.fig', 'fig');
235 %******************************************
236 % Plot just mean with error bars:
237 % Set ordinate upperbound as maximum accumulated distance value:
238   temp = zeros(1,2);
239   temp(1) = max(Mean_Path_Length_tabs(:,1)+Mean_Path_Length_tabs(:,2));
240   temp(1,2) = max(Mean_Path_Length_tbin(:,1)+Mean_Path_Length_tbin(:,2));
241   PL_max = max(temp)*pixel_calib;
242
243   h2 = figure;
244   subplot(1,2,1)
245   errorbar(t_abs,m_abs,std_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold on
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('<Path Length> ± s.d. (μm)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
axis([0 t_max 0 PL_max]);

subplot(1,2,2)
errorbar(t_bin,m_bin,std_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('<Path Length> ± s.d. (μm)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
axis([0 t_max 0 PL_max]);

% Save figure window generated:
saveas(h2, 'Mean_Path_Length.fig', 'fig');
%***************

% Save 'Mean_Path_Length' arrays in .mat and .txt format:
save 'Mean_Path_Length_tabs.mat' Mean_Path_Length_tabs
fid2_1 = fopen('Mean_Path_Length_tabs.txt','wt');
fprintf(fid2_1,'Mean PL (# pix)	Stdrd Dev (# pix)	# Obs	Abs dt (sec)
');
rows = length(Mean_Path_Length_tabs);
for k = 1:rows
    fprintf(fid2_1,'%f	%f	%.0f	%.0f
',Mean_Path_Length_tabs(k,:));
end
close(fid2_1);

save 'Mean_Path_Length_tbin.mat' Mean_Path_Length_tbin
fid2_2 = fopen('Mean_Path_Length_tbin.txt','wt');
fprintf(fid2_2,'Mean PL (# pix)	Stdrd Dev (# pix)	# Obs	Bin dt (sec)
');
rows = length(Mean_Path_Length_tbin);
for kk = 1:rows
    fprintf(fid2_2,'%f	%f	%.0f	%.0f
',Mean_Path_Length_tbin(kk,:));
end
close(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'%nFunction completed without errors/warnings%n');
    fprintf(fid,'%nFunction completed without errors/warnings%n');
end

% Update log file that function is completed:
fprintf(fid,'%n%s completed',func_name);
fprintf(1,'%n%s completed',func_name);

Area_v4.m

% Steven J. Henry
% 06/09/2011

%**************************************************************************
% PURPOSE:
% This function plots area of each cell over elapsed experimental imaging
% time and saves the results. Since experiments are all 2D migration and
% cells are terminally differentiated, cross sectional area (contact area)
% can be considered a metric of substrate affinity. If area changes with
% time one might correlate this to a change in substrate affinity.
% % ASSUMPTIONS:
% n/a
% INPUT:
data = array having following structure:
col 1 = unique track number ID assigned to each cell
col 2 = x coordinate (pixels) of centroid
col 3 = y coordinate (pixels) of centroid
col 4 = absolute time corresponding to frame in which cell is found
(col 5 = binned time corresponding to frame in which cell is found (sec)
col 6 = area of cell in pixels
col 7 = track change flag. Entry = 1 if start of new track (yes) or 0 if no (i.e. continuation of an existing track.

% pixel_calib = user-supplied microns/pixel
t_max = user-supplied imaging duration in minutes
run_title = user specified string description of experimental condition
fid = file ID of log file to which progress is recorded

% OUTPUT:
Area = array of track area data with following structure:
col 1 = track ID
col 2 = cell area (pixels)
col 3 = absolute elapsed time (sec)
col 4 = binned elapsed time (sec)
col 5 = track change flag. Entry = 1 if start of new track (yes) or 0 if no (i.e. continuation of an existing track.

function [Area] = Area_v4(data, pixel_calib, run_title, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a warning dialog the log file records no errors/warnings generated:
warn = 0;

% Determine number of rows:
[rows] = size(data);
rows = rows(1);

% Reserve memory block for 'Area' array:
Area = zeros(rows,5);

% Transcribe track IDs, areas, and track change flags:
Area(:,1) = data(:,1);
Area(:,2) = data(:,6);
Area(:,5) = data(:,7);

% Determine indices of start and stop positions of each unique track in the overall 'data' array:
[junk ind_start] = unique(data(:,1),'first');
clear junk
[junk ind_stop] = unique(data(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'
\n\nWARNING: # Unique Track IDs start positions ~= # stop positions\n\n\n');
    fprintf(fid,\n\n\nWARNING: # Unique Track IDs start positions ~= # stop positions\n\n\n');
    warn = 1;
end

% Total number of tracks to be analyzed:
um_tracks = length(ind_start);
% Loop over tracks to compute elapsed times. Note: this was previously done
% in 'Path_Length.m' but is duplicated here so that the 'Path_Length'
% and 'Area.m' functions are independent.
for i = 1:num_tracks

    % Define start and stop position:
    r_start = ind_start(i);
    r_stop = ind_stop(i);

    % Loop over all rows between start and stop rows:
    for ii = r_start:r_stop

        % If index is on start row:
        if ii == r_start
            % Elapsed time values are zero:
            Area(ii,3:4) = 0;
        else

            % Load info of cell (track) in previous frame:
            t_abs_prior = data(ii-1,4);
            t_bin_prior = data(ii-1,5);

            % Load info of cell in present frame:
            t_abs_now = data(ii,4);
            t_bin_now = data(ii,5);

            % Compute elapsed time between these two frames:
            dt_abs = t_abs_now - t_abs_prior;
            dt_bin = t_bin_now - t_bin_prior;

            % Log as the cumulative elapsed time as the elapsed time
            % between these two frames to the previous cumulative values.
            % Note: in essence we are computing our sums by adding the
            % differential of each adjacent frame.
            Area(ii,3) = Area(ii-1,3)+dt_abs;
            Area(ii,4) = Area(ii-1,4)+dt_bin;

        end

    end
end

% Determine physical area of each pixel (assume pixel is square with edge
% length 'pixel_calib'). Units of pixel_area are (um^2/pixel)
pixel_area = pixel_calib^2;

% Plot results:
figure;
for j = 1:num_tracks

    % Define start and stop position:
    r_start = ind_start(j);
    r_stop = ind_stop(j);

    % Create vectors of track data:
    A = Area(r_start:r_stop,2);
    t_abs = Area(r_start:r_stop,3);
    t_bin = Area(r_start:r_stop,4);

    % Convert areas (in # pixels) to microns^2. Note: 'pixel_area' has
    % units micron^2/pixel
    A = A*pixel_area;
end
% Convert time vectors to minutes from seconds:
168   t_abs = t_abs/60;
169   t_bin = t_bin/60;
171
172 % Plot area vs. elapsed time (absolute)
173 % Units are microns^2 and minutes
174 subplot(1,2,1)
175 hold on
176 plot(t_abs,A,'LineStyle','-','Color','b','Marker','none','LineWidth',1);
177
178 % Plot area vs. elapsed time (binned)
179 % Units are microns^2 and minutes
180 subplot(1,2,2)
181 hold on
182 plot(t_bin,A,'LineStyle','-','Color','b','Marker','none','LineWidth',1);
183
184 end
185
186 % Set ordinate upperbound as maximum area value:
187   A_max = max(Area(:,2))*pixel_area;
188
189 % Set abscissa upperbound as maximum elapsed experimental imaging time
190 % value:
191   t_max = zeros(2,1);
192   t_max(1) = max(data(:,4)); % sec
193   t_max(2) = max(data(:,5)); % sec
194   t_max = max(t_max)/60; % min
195
196 % Label axes:
197   subplot(1,2,1)
198   title(run_title,'FontName','Arial','FontSize',16);
199   xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
200   ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
201   set(gca,'FontName','Arial');
202   set(gca,'FontSize',14);
203   axis([0 t_max 0 A_max]);
204
205   subplot(1,2,2)
206   title(run_title,'FontName','Arial','FontSize',16);
207   xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
208   ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
209   set(gca,'FontName','Arial');
210   set(gca,'FontSize',14);
211   axis([0 t_max 0 A_max]);
212
213 % Save 'Area' array in .mat and .txt format:
214   save 'Area.mat'  Area;
215   fid2 = fopen('Area.txt','wt');
216   fprintf(fid2,'ID	Area (# pix)	Abs dt (sec)	Bin dt (sec)	TrackChange
');
217   for k = 1:rows
218       fprintf(fid2,'%.0f	%f	%.0f	%.0f	%.0f
',Area(k,:));
219   end
220   fclose(fid2);
221
222 % Save figure window generated:
223   saveas(h, 'Area.fig', 'fig');
224
225 % If no warnings generated report so in log file:
226   if warn == 0
227       fprintf(fid, '\nFunction completed without errors/warnings\n');
228       fprintf(fid, '\nFunction completed without errors/warnings\n');
229   end
230
231 % Update log file that function is completed:
232   fprintf(fid, '\n%s completed\n',func_name);
233   fprintf(fid, '\n%s completed\n',func_name);
234
235 end
Mean_Area_v5.m

function [] = Mean_Area_v5(Area, pixel_calib, run_title, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Identify all elapsed time intervals that have been observed (using both
% absolute and binned time intervals):
t_abs_uniq = unique(Area(:,3));
t_bin_uniq = unique(Area(:,4));

for i = 1:2
    if i == 1 % Work with absolute time values
        t_uniq = t_abs_uniq;
        t_all = Area(:,3);
    elseif i == 2 % Work with binned time values
        t_uniq = t_bin_uniq;
        t_all = Area(:,4);
    end
end

% Determine number of unique time values:
num_uniq = length(t_uniq);
% Reserve memory for temporary 'Mean_Array'
Mean_Area = zeros(num_uniq,4);

% Loop over all unique elapsed time values:
for ii = 1:num_uniq
    % Time interval
    dt = t_uniq(ii);
    % Find indices in 't_all' that map to positions where entry is
    % equal to 'dt'
    ind = t_all == dt;
    % Because 'Area' and 't_all' have the same row
    % structure/organization use the indices above to extract the
    % corresponding area values observed at the given 'dt':
    A_dt = Area(ind,2);
    % Compute mean area for given 'dt'
    A_mean_dt = mean(A_dt);
    % Compute standard deviation for given 'dt'
    A_std_dt = std(A_dt);
    % Log mean area, standard deviation, the number of observations
    % used in the mean calculation, and the elapsed time interval:
    Mean_Area(ii,1) = A_mean_dt;
    Mean_Area(ii,2) = A_std_dt;
    Mean_Area(ii,3) = sum(ind);
    Mean_Area(ii,4) = dt;
end
if i == 1
    Mean_Area_tabs = Mean_Area;
    clear Mean_Area
elseif i == 2
    Mean_Area_tbin = Mean_Area;
    clear Mean_Area
end

% Determine indices of start and stop positions of each unique track in the
% 'Area' array:
[junk ind_start] = unique(Area(:,1),'first');
clear junk
[junk ind_stop] = unique(Area(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'WARNING: # Unique Track IDs start positions ~= # stop positions\n');
    fprintf(fid,'WARNING: # Unique Track IDs start positions ~= # stop positions\n');
    warn = 1;
    keyboard
end

% Total number of tracks to be analyzed:
num_tracks = length(ind_start);

% Determine physical area of each pixel (assume pixel is square with edge
% length 'pixel_calib'). Units of pixel_area are (um^2/pixel)
pixel_area = pixel_calib^2;

% Plot overlay figure
h = figure;
for j = 1:num_tracks

% Define start and stop position:
r_start = ind_start(j);
r_stop = ind_stop(j);

% Create vectors of track data:
A = Area(r_start:r_stop,2);
t_abs = Area(r_start:r_stop,3);
t_bin = Area(r_start:r_stop,4);

% Convert areas (in # pixels) to microns^2. Note: 'pixel_area' has
% units micron^2/pixel
A = A*pixel_area;

% Convert time vectors to minutes from seconds:
t_abs = t_abs/60;
t_bin = t_bin/60;

% Plot area vs. elapsed time (absolute)
% Units are microns^2 and minutes
subplot(1,2,1)
hold on
plot(t_abs,A,'LineStyle','none','Marker','.','LineWidth',1,'Color','b','HandleVisibility','off');

% Plot area vs. elapsed time (binned)
% Units are microns^2 and minutes
subplot(1,2,2)
hold on
plot(t_bin,A,'LineStyle','none','Marker','.','LineWidth',1,'Color','b','HandleVisibility','off');

end

% Convert pixels to microns and sec to min:
t_abs = Mean_Area_tabs(:,4)/60;
m_abs = Mean_Area_tabs(:,1)*pixel_area;
std_abs = Mean_Area_tabs(:,2)*pixel_area;
t_bin = Mean_Area_tbin(:,4)/60;
m_bin = Mean_Area_tbin(:,1)*pixel_area;
std_bin = Mean_Area_tbin(:,2)*pixel_area;

% Set ordinate upperbound as maximum area value:
A_max = max(Area(:,2))*pixel_area;

% Set abscissa upperbound as maximum elapsed experimental imaging time
max = max(t_abs); % min

% Label axes and overlay mean series:
subplot(1,2,1)
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('Cell Area (\mu m^2)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
h_legend = legend('Mean Area','Location','NorthEast');
set(h_legend,'FontName','Arial');
set(h_legend,'FontSize',12);
axis([0 t_max 0 A_max]);

subplot(1,2,2)
plot(t_bin,m_bin,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('Cell Area (\mu m^2)','FontName','Arial','FontSize',16);
set(gca,'FontName','Arial','FontSize',14);
set(gca,'Location','NorthEast');
legend('Mean Area','Location','NorthEast');
set(h_legend,'FontName','Arial','FontSize',12);
axis([0 t_max 0 A_max]);

% Save figure window generated:
saveas(h, 'Mean_Area_Overlay.fig', 'fig');

%***************

% Plot just mean with error bars:

% Set ordinate upperbound as maximum accumulated distance value:
temp = zeros(1,2);
temp(1,1) = max(Mean_Area_tabs(:,1)+Mean_Area_tabs(:,2));
temp(1,1) = max(Mean_Area_tbin(:,1)+Mean_Area_tbin(:,2));
A_max = max(temp)*pixel_area;
h2 = figure;
subplot(1,2,1);
errorbar(t_abs,m_abs,std_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('<Cell Area> \pm s.d. (\mu m^2)','FontName','Arial','FontSize',16);
set(gca,'FontSize',14);
axis([0 t_max 0 A_max]);

subplot(1,2,2);
errorbar(t_bin,m_bin,std_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
title(run_title,'FontName','Arial','FontSize',16);
xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
ylabel('<Cell Area> \pm s.d. (\mu m^2)','FontName','Arial','FontSize',16);
set(gca,'FontSize',14);
axis([0 t_max 0 A_max]);

% Save 'Mean_Area' arrays in .mat and .txt format:
save 'Mean_Area_tabs.mat' Mean_Area_tabs;
 fid2_1 = fopen('Mean_Area_tabs.txt','wt');
 fprintf(fid2_1,'Mean Area (# pix)	Stdrd Dev (# pix)	# Obs	Abs dt (sec)
');
 rows = length(Mean_Area_tabs);
 for k = 1:rows
  fprintf(fid2_1,'%f	%f	%.0f	%.0f
',Mean_Area_tabs(k,:));
 end
close(fid2_1);
save 'Mean_Area_tbin.mat' Mean_Area_tbin;
 fid2_2 = fopen('Mean_Area_tbin.txt','wt');
 fprintf(fid2_2,'Mean Area (# pix)	Std Dev (# pix)	ObstAbs dt (sec)
');
 rows = length(Mean_Area_tbin);
for kk = 1:rows
    fprintf(fid2_2,'%f	%f	%.0f	%.0f
',Mean_Area_tbin(kk,:));
end
close(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings\n');
    fprintf(fid,'
	Function completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(1,'
%s completed\n',func_name);
fprintf(fid,'
%s completed\n',func_name);
end

function [data_filt] = Filter_Exp_Data_v3(data, exp_t_max, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,\’%n\%s running \%\n’,func_name);
fprintf(fid,\’%n\%s running \%\n’,func_name);

% Turn warning flag ‘warn’ off. If ‘warn’ is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Convert 'exp_t_max' to sec from min:
exp_t_max = exp_t_max*60;

% Identify binned time values that exceed exp_t_max value in seconds:
del_ind = data(:,5) > exp_t_max;

% Eliminate these rows:
data_filt = data;
data_filt(del_ind,:) = [];

% Dimension check:
num_del_theory = sum(del_ind);
num_del_practice = size(data,1)-size(data_filt,1);
if num_del_practice ~= num_del_theory
    fprintf(1,'
	WARNING: # deletions from data ~= # deletions predicted
');
    fprintf(1,'
	WARNING: # deletions from data ~= # deletions predicted
');
    warn = 1;
    keyboard
end

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings
');
    fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(fid,'Function completed without errors/warnings
');

Differentials_v5.m

1   % Steven J. Henry
2   % 08/11/2012
45   %**************************************************************************
46   % PURPOSE:
47   % This function computes squared differentials in displacement and time for
48   % all intervals **USING A MOVING ORIGIN** along a given cell's track.
49   %
50   % ASSUMPTIONS:
51   % n/a
52   %
53   % INPUT:
54   % data = array having following structure:
55   %   col 1 = unique track number ID assigned to each cell
56   %   col 2 = x coordinate (pixels) of centroid
57   %   col 3 = y coordinate (pixels) of centroid
58   %   col 4 = absolute time corresponding to frame in which cell is found
59   % (sec)
60   %   col 5 = binned time corresponding to frame in which cell is found (sec)
61   %   col 6 = area of cell in pixels
62   %   col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
63   % if no (i.e. continuation of an existing track.
64   % fid = file ID of log file to which progress is recorded
65   %
66   % OUTPUT:
67   % SD = array containing squared displacements and corresponding times
68   %   col 1 = track ID to which this interval belongs
69   %   col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
70   %   col 4 = interval binned time (sec)
71   %   col 5 = dx (pix)
function [SD] = Differentials_v5(data, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Determine indices of start and stop positions of each unique track in the
% overall 'data' array:
[junk ind_start] = unique(data(:,1),'first');
clear junk
[junk ind_stop] = unique(data(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'
	WARNING: # Unique Track IDs start positions ~= # stop positions
');
    fprintf(fid,'
	WARNING: # Unique Track IDs ~= # Track Change Flags
');
    warn = 1;
end

% Total number of tracks to be analyzed:
num_tracks = length(ind_start);
SD = [];

% Reserve 'num_intervals_tot' to hold total number of intervals computed:
um_intervals_tot = 0;

% Loop over tracks:
for i = 1:num_tracks
    % Define start and stop position:
    r_start = ind_start(i);
    r_stop = ind_stop(i);

    % Determine the number of intervals that will be observed when using a
    % moving origin from the first row of this track to the second-to-last
    % row of the track.
    N = r_stop-r_start+1;
    num_intervals = 0;
    for j = 1:N
        num_intervals = (j-1) + num_intervals;
    end
    track_SD = zeros(num_intervals,6);
    num_intervals_tot = num_intervals_tot + num_intervals;

    print_row = 1;

    % Loop over all rows between start and second-to-last stop rows:
    for orig_row = r_start:r_stop-1
        % Load origin info:
        track_ID = data(orig_row,1);
        x_orig = data(orig_row,2);
        y_orig = data(orig_row,3);
        t_abs_orig = data(orig_row,4);
        t_bin_orig = data(orig_row,5);

        ...
% Loop over all advance rows ahead of origin between 'orig_row' + 1
% and last row ('r_stop'):
for adv_row = orig_row + 1:r_stop

% Load advance row info:
x_adv = data(adv_row,2);
y_adv = data(adv_row,3);
t_abs_adv = data(adv_row,4);
t_bin_adv = data(adv_row,5);

% Compute differentials:
dx = (x_adv - x_orig);
dx2 = dx^2;
dy = (y_adv - y_orig);
dy2 = dy^2;
dt_abs = t_abs_adv - t_abs_orig;
dt_bin = t_bin_adv - t_bin_orig;

% Compute squared displacement:
d2 = dx2 + dy2;

% Log values:
track_SD(print_row,1) = track_ID;
track_SD(print_row,2) = d2;
track_SD(print_row,3) = dt_abs;
track_SD(print_row,4) = dt_bin;
track_SD(print_row,5) = dx;
track_SD(print_row,6) = dy;

% Advance print_row:
print_row = print_row+1;
end
end

% After all intervals for 'track_ID' are computed make sure the length
% of 'track_SD' is = 'num_intervals':
[track_SD_r track_SD_c] = size(track_SD);
if track_SD_r~=num_intervals
    fprintf(1,'\n	WARNING: # intervals computed ~= expected #\n');
    fprintf(1,'\tWarning generated for track = %.0f\n',track_ID);
    fprintf(fid,'\n	WARNING: # intervals computed ~= expected #\n');
    fprintf(fid,'\tWarning generated for track = %.0f\n',track_ID);
    warn = 1;
    keyboard
end

% Concatenate 'track_SD' with existing 'SD'
if isempty(SD) == 1
    SD = track_SD;
    clear track_SD
else
    SD_old = SD;
    [SD_old_r SD_old_c] = size(SD_old);
    clear SD
    SD = zeros(SD_old_r+track_SD_r,SD_old_c);
    SD(1:SD_old_r,1:track_SD_c) = SD_old;
    SD(SD_old_r+1:SD_old_r+track_SD_r,1:track_SD_c) = track_SD;
    [SD_new_r SD_new_c] = size(SD);
end
end
% Dimensionality check. There should be six columns after the
% concatenations are finished:
if SD_new_c ~= 6
    fprintf(1,'
	WARNING: # cols = %.0f not 6 as expected\n',SD_new_c);
    fprintf(fid,'\n	Warning generated for track = %.0f
',track_ID);
    warn = 1;
    keyboard
end

% Dimensionality check. The total number of rows in 'SD' should be equal to
% the sum of each track's theoretical intervals:
if SD_new_r ~= num_intervals_tot;
    fprintf(1,'
	WARNING: # rows (intervals) = %.0f not %.0f as expected
',SD_new_r,num_intervals_tot);
    fprintf(fid,'
	WARNING: # rows (intervals) = %.0f not %.0f as expected
',SD_new_r,num_intervals_tot);
    warn = 1;
    keyboard
end

% Check for instances in which a squared displacement was measured over an
% elapsed time of zero:
dt_zero_inds = SD(:,3)==0 | SD(:,4)==0;
num_dt_zero_inds = sum(dt_zero_inds);
if num_dt_zero_inds > 0
    dt_nz_inds = SD(:,3)~=0 & SD(:,4)~=0;
    SD = SD(dt_nz_inds,:);
    % Dimensionality check.
    SD_dt_nz_r = size(SD,1);
    if SD_dt_nz_r ~= SD_new_r - num_dt_zero_inds
        fprintf(1,'
	WARNING: # zero tau deletions made (%.0f) ~= # zero tau deletions expected (%.0f)\n',SD_new_r-
SD_dt_nz_r,num_dt_zero_inds);
        fprintf(fid,'
	WARNING: # zero tau deletions made (%.0f) ~= # zero tau deletions expected
(%.0f)\n',SD_new_r-SD_dt_nz_r,num_dt_zero_inds);
        warn = 1;
        keyboard
    end
end

% Save 'SD' array in .mat and .txt format:
save 'SD.mat' SD;
fid2 = fopen('SD.txt','wt');
fprintf(fid2,'Track ID	d^2 (pix^2)	Abs dt (sec)	Bin dt (sec)	dx (pix)	dy (pix)\n');
final_num_rows = size(SD,1);
for k = 1:final_num_rows
    fprintf(fid2,'%.0f	%f	%.0f	%.0f	%.0f	%.0f\n',SD(k,:));
end
fclose(fid2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,\n	Function completed without errors/warnings\n');
    fprintf(fid,\n	Function completed without errors/warnings\n');
    end
end

% Update log file that function is completed:
fprintf(fid,\n%s completed\n',func_name);

Neff_v1.m

% Steven J. Henry
% 06/10/2011
%**************************************************************************
% PURPOSE: This function computes Neff value for computation of standard error of the variance associated with MSD values. Neff is the number of **INDEPENDENT** squared displacement observations associated with a given tau interval. % The quantity is defined by Crocker and Hoffman in "Multiple-Particle Tracking and Two-Point Microrheology in Cells", Methods in Cell Biology, 2007, Vol 83, P141. % The essential points to consider are: % (1) To increase the number of squared displacement observations associated with a given tau we previously invoked a "moving origin" strategy in "Differentials.m". Doing so means we introduced correlations in the data which we must account for in the computation of the MSD error bars. As a result of this moving origin strategy we cannot simply set Neff equal to the total number of squared displacement observations for a given tau which is a number larger than the number of actual independent measurements. Thus we would artificially and erroneously decrease the magnitude of our error bars. % (2) The Crocker & Hoffman text defines Neff ~ Ncell*T/tau. However this equation is for deal data in which one it is possible to track all Ncells for the duration of the imaging experiment. In our data Ncells is a number that generally decreases with time and furthermore the duration of each track of the set of tracks comprising Ncells are not of equal length. % In this code Neff is computed to account for both points (1) and (2) above.

% ASSUMPTIONS: n/a

% INPUT:
- data = array having following structure:
  - col 1 = unique track number ID assigned to each cell
  - col 2 = x coordinate (pixels) of centroid
  - col 3 = y coordinate (pixels) of centroid
  - col 4 = absolute time corresponding to frame in which cell is found (sec)
  - col 5 = binned time corresponding to frame in which cell is found (sec)
  - col 6 = area of cell in pixels
  - col 7 = track change flag. Entry = 1 if start of new track (yes) or 0 if no (i.e. continuation of an existing track).

- SD = array containing squared displacement differentials and linear time differentials:
  - col 1 = track ID to which this interval belongs
  - col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
  - col 3 = interval absolute time (sec)
  - col 4 = interval binned time (sec)
  - col 5 = dx (pix)
  - col 6 = dy (pix)

- fid = file ID of log file to which progress is recorded

% OUTPUT:
- Indep_Obs_tabs = ("Independent Observations") array containing Neff values corresponding to unbinned tau intervals:
  - col 1 = tau values using absolute time intervals (sec)
  - col 2 = Neff values associated with absolute tau intervals
- Indep_Obs_tbin = ("Independent Observations") array containing Neff values corresponding to binned tau intervals:
  - col 1 = tau values using binned time intervals (sec)
  - col 2 = Neff values associated with binned tau intervals

function [Indep_Obs_tabs, Indep_Obs_tbin] = Neff_v1(data, SD, fid)
% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Determine indices of start and stop positions of each unique track in the
% overall 'data' array:
[junk ind_start] = unique(data(:,1),'first');
clear junk
[junk ind_stop] = unique(data(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'
	WARNING: # Unique Track IDs start positions ~= # stop positions
');
    fprintf(fid,'
	WARNING: # Unique Track IDs ~= # Track Change Flags
');
    warn = 1;
end

% Total number of tracks to be analyzed:
um_tracks = length(ind_start);

% Identify all elapsed time intervals that have been observed (using both
% absolute and binned time intervals):
tau_abs = unique(SD(:,3));
tau_abs = sort(tau_abs,'ascend');
num_tau_abs = length(tau_abs);
tau_bin = unique(SD(:,4));
tau_bin = sort(tau_bin,'ascend');
num_tau_bin = length(tau_bin);

% Generate vectors that will hold the Neff counts corresponding to each tau
% value. Initially all entries are zero but as each track is processed
% sequentially the Neff value is updated (running sum).
Indep_Obs_tabs = zeros(num_tau_abs,2);
Indep_Obs_tabs(:,1) = tau_abs;
Indep_Obs_tbin = zeros(num_tau_bin,2);
Indep_Obs_tbin(:,1) = tau_bin;

% Loop over tracks:
for i = 1:num_tracks
    % Define start and stop row positions:
    r_start = ind_start(i);
    r_stop = ind_stop(i);

    % Get total time 'T' track 'i' has been imaged for:
    T_abs = data(r_stop,4) - data(r_start,4); % sec
    T_bin = data(r_stop,5) - data(r_start,5); % sec

    % Compute number of independent (i.e. non-overlapping) observations
    % possible for this track with respect to each of the possible tau
    % values. The result of this operation is a vector. Note we round down
    % to make sure we only count whole intervals:
    Neff_abs = floor(T_abs./tau_abs);
    Neff_bin = floor(T_bin./tau_bin);

    % Update output arrays:
Indep_Obs_tabs(:,2) = Indep_Obs_tabs(:,2) + Neff_abs;
Indep_Obs_tbin(:,2) = Indep_Obs_tbin(:,2) + Neff_bin;
end
% Save 'Indep_Obs' arrays in .mat and .txt format:
save 'Indep_Obs_tabs.mat' Indep_Obs_tabs;
fid2 = fopen(['Indep_Obs_tabs.txt'],'wt');
fprintf(fid2,'tau (abs dt) (sec)	Neff (#)
');
for k = 1:num_tau_abs
    fprintf(fid2,'%.0f	%.0f
',Indep_Obs_tabs(k,:));
end
close(fid2);

save 'Indep_Obs_tbin.mat' Indep_Obs_tbin;
fid3 = fopen(['Indep_Obs_tbin.txt'],'wt');
fprintf(fid3,'tau (bin dt) (sec)	Neff (#)
');
for k = 1:num_tau_bin
    fprintf(fid3,'%.0f	%.0f
',Indep_Obs_tbin(k,:));
end
close(fid3);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings
');
    fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(1,'
%s completed
',func_name);
fprintf(fid,'
%s completed
',func_name);
end

%**************************************************************************
% PURPOSE:
% This function computes mean (time and ensemble averaged) squared
% displacements of all cells as a function of lag time (tau) in terms of
% both absolute and binned time intervals. Because the squared
% displacements were previously computed using a moving origin strategy
% (see "Differentials.m" code) the elapsed time values are lag times or
% "taus". These time intervals were either in terms of absolute time
% or binned time differences. The binning strategy is to help
% improve the number of samples per mean computation.
%
% REMARKS:
% Assuming the underlying errors in measurement are Gaussian then the SEV
% is:
%   SEV = 2*MSD(tau)/sqrt(Neff)
% where Neff = number of **INDEPENDENT** observations of displacements
% associated with a given imaging time interval tau.
%
% The SEV form was obtained from discussion with John Crocker on 01/31/2011
% and reference to Crocker and Hoffman's "Multiple-Particle Tracking and
%
% An extremely practical reference regarding error bar construction and the
% source of "inferential" vs. "descriptive" definitions above can be found
% in Cumming's "Error bars in experimental biology," The Journal of Cell
%
% INPUT:
% SD = array containing squared displacement differentials and linear time
% differentials ("taus");

Mean_Differentials_v6.m
273
274 % col 1 = track ID to which this interval belongs
275 % col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
276 % Note: d^2 values are not rounded to the nearest whole pixel
277 % col 3 = tau absolute time (sec)
278 % col 4 = tau binned time (sec)
279 % col 5 = dx (pix)
280 % col 6 = dy (pix)
281 % Indep_Obs_tabs = ("Independent Observations") array containing Neff
282 % values corresponding to unbinned tau intervals:
283 % col 1 = tau values using absolute time intervals (sec)
284 % col 2 = Neff values associated with absolute taus
285 % Indep_Obs_tbin = ("Independent Observations") array containing Neff
286 % values corresponding to binned tau intervals:
287 % col 1 = tau values using binned time intervals (sec)
288 % col 2 = Neff values associated with binned taus
289 % fid = file ID of log file to which progress is recorded
290 % OUTPUT:
291 % MSD_tabs = array containing mean squared displacements and corresponding
292 % lag time taus in terms of absolute differences with the following
293 % structure:
294 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
295 % col 2 = obsolete 'NaN'
296 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
297 % col 4 = Neff (number of independent observations)
298 % col 5 = tau absolute elapsed time (sec)
299 % col 6 = VAR(dx) (pix^2)
300 % col 7 = VAR(dy) (pix^2)
301 % MSD_tbin = array containing mean squared displacements and corresponding
302 % lag time taus in terms of binned differences with the following
303 % structure:
304 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
305 % col 2 = obsolete 'NaN'
306 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
307 % col 4 = Neff (number of independent observations)
308 % col 5 = tau binned elapsed time (sec)
309 % col 6 = VAR(dx) (pix^2)
310 % col 7 = VAR(dy) (pix^2)
311 %**************************************************************************
312 function [MSD_tabs, MSD_tbin] = Mean_Differentials_v6(SD, Indep_Obs_tabs, Indep_Obs_tbin, fid)
313 % Get function name:
314 func_name = mfilename;
315 % Update log file that function is running:
316 fprintf(1,'
%s running ...
',func_name);
317 fprintf(fid,'
%s running ...
',func_name);
318 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
319 % warning dialog the log file records no errors/warnings generated:
320 warn = 0;
321 % Identify all elapsed time intervals that have been observed (using both
322 % absolute and binned time intervals):
323 t_abs_uniq = unique(SD(:,3));
324 t_abs_uniq = sort(t_abs_uniq,'ascend');
325 t_bin_uniq = unique(SD(:,4));
326 t_bin_uniq = sort(t_bin_uniq,'ascend');
327 % Check for concisistency with tau values previously computed in "Neff.m"
328 % function:
329 if t_abs_uniq ~= Indep_Obs_tabs(:,1)
330 fprintf(fid,\"\nWarning: Absolute tau values in \%s not same as those passed by \"Neff.m\"\n\",func_name);
331 end
332 if t_bin_uniq ~= Indep_Obs_tbin(:,1)
333 fprintf(fid,\"\nWarning: Binned tau values in \%s not same as those passed by \"Neff.m\"\n\",func_name);
334 end
335
273
fprintf(fid,"\n\tWARNING: Binned tau values in \%s not same as those passed by \"Neff.m\"\n",func_name);

end

for i = 1:2
    if i == 1 % Work with lag time taus in absolute differences (sec)
        t_uniq = t_abs_uniq;
        t_all = SD(:,3);
        indep = Indep_Obs_tabs(:,2);
    elseif i == 2 % Work with lag time taus in binned differences (sec)
        t_uniq = t_bin_uniq;
        t_all = SD(:,4);
        indep = Indep_Obs_tbin(:,2);
    end

    % Determine number of unique time values:
    num_uniq = length(t_uniq);

    % Reserve memory for temporary 'MSD'
    MSD = zeros(num_uniq,7);

    % Loop over all unique elapsed time values:
    for ii = 1:num_uniq
        % Time interval
        tau = t_uniq(ii);

        % Find indices in 't_all' that map to positions where entry is
        % equal to 'tau'
        ind = t_all==tau;

        % Because 'SD' and 't_all' have the same row structure/organization
        % use the indices above to extract the corresponding square
        % displacement values observed at the given 'tau' both within and
        % across all tracks:
        SD_tau = SD(ind,2);

        % Compute mean squared displacement for given 'tau'
        varx = var(SD(ind,5));
        vary = var(SD(ind,6));
        covxy = cov(SD(ind,5),SD(ind,6));
        mean_tau = varx+vary+2*covxy;

        % Compute standard error of the variance for given 'tau'. Recall:
        % SEV = 2*MSD(tau)/sqrt(Neff)
        % where Neff is called from Indep_Obs array constructed previously
        % in "Neff.m" function.
        Neff = indep(ii);
        sev_tau = 2*mean_tau/sqrt(Neff);

        % Log MSD(tau), SD, SEV, the number of observations (Ncell) used in
        % the mean calculation, and tau:
        MSD(ii,1) = mean_tau; % pixels^2
        MSD(ii,2) = NaN;
        MSD(ii,3) = sev_tau; % pixels^2
        MSD(ii,4) = Neff; % integer value
        MSD(ii,5) = tau; % sec
        MSD(ii,6) = varx; % pix^2
        MSD(ii,7) = vary; % pix^2

    end

    if i == 1
        MSD_tabs = MSD;
        clear MSD
    elseif i == 2
        MSD_tbin = MSD;
clear MSD

% Save 'MSD' arrays in .mat and .txt format:
save 'MSD_tabs.mat' MSD_tabs
fid2_1 = fopen('MSD_tabs.txt','wt');
fprintf(fid2_1,'MSD(tau) (pix^2)	NaN	Abs tau (sec)	VAR(dx) (pix^2)	VAR(dy) (pix^2)
');
rows = size(MSD_tabs,1);
for k = 1:rows
    fprintf(fid2_1,'%f	%f	%f
',MSD_tabs(k,:));
end
fclose(fid2_1);

save 'MSD_tbin.mat' MSD_tbin
fid2_2 = fopen('MSD_tbin.txt','wt');
fprintf(fid2_2,'MSD(tau) (pix^2)	NaN	Abs tau (sec)	VAR(dx) (pix^2)	VAR(dy) (pix^2)
');
rows = size(MSD_tbin,1);
for k = 1:rows
    fprintf(fid2_2,'%f	%f	%f
',MSD_tbin(k,:));
end
fclose(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings
');
    fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(1,'
%s completed
',func_name);
fprintf(fid,'
%s completed
',func_name);

end

MSD_Epsilon_Subtract_v3.m

% Steven J. Henry
% 08/11/2012
%**************************************************************************
% PURPOSE:
% The following function subtracts a user-supplied epsilon value uniformly
% from all MSD data points. The epsilon value is a quantified measure of
% noise in the data for the corresponding experimental condition derived
% from 2D MSD values.
% REMARKS:
% We can subtract a constant value from the array of means without having
% to recompute the standard deviation or standard error of the variance.
% Standard deviation is invariant under location. An abbreviated proof
% follows:
% Let X = {x1, x2, x3...xN}
% E(X) = (1/N)*sum(xi,i=1..N)
% E(X-c) = (1/N)*sum(xi-c,i=1..N) = (1/N)*sum(xi,i=1..N) - (1/N)*N*c
% E(X-c) = (1/N)*sum(xi,i=1..N) - c
% E(X-c) = E(X) - c
% VAR(X) = (1/N)*sum((xi-E(X))^2,i=1..N)
% VAR(X) = E(X^2) - E(X)^2
% VAR(X-c) = E((X-c)^2) = E(X-c)^2 - E(X) - c
% Recall E(X) = E(X-c) = E(X-c)^2
% Recall E(constant) = constant
% VAR(X-c) = E((X-c)^2) - 2E(X)(c) + c^2 - (E(X)^2 + 2E(X)c + c^2)
% VAR(X-c) = E((X-c)^2) - 2E(X)c + c^2 - (E(X)^2 + 2E(X)c + c^2)
% VAR(X-c) = E((X-c)^2) - E(X)c^2
% VAR(X-c) = VAR(X)
% Since std = sqrt(VAR)
% std(X-c) = std(X)

% ASSUMPTIONS:
% n/a

% INPUT:
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau absolute elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% MSD_tbin = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau binned elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% epsilon = user specified noise constant (pix)
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% MSD_tabs_epsilon = same structure as input array except with epsilon
% subtracted from all MSD values
% MSD_tbin_epsilon = same structure as input array except with epsilon
% subtracted from all MSD values

function [MSD_tabs_epsilon MSD_tbin_epsilon] = MSD_Epsilon_Subtract_v3(MSD_tabs, MSD_tbin, epsilon, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Subtract 4*epsilon^2 from all MSD values in column 1:
MSD_tabs_epsilon = MSD_tabs;
MSD_tabs_epsilon(:,1) = MSD_tabs_epsilon(:,1)-4*epsilon^2;
MSD_tbin_epsilon = MSD_tbin;
MSD_tbin_epsilon(:,1) = MSD_tbin_epsilon(:,1)-4*epsilon^2;

% Save epsilon corrected 'MSD' arrays in .mat and .txt format:
save 'MSD_tabs_epsilon.mat' MSD_tabs_epsilon
fid2_1 = fopen('MSD_tabs_epsilon.txt','wt');
fprintf(fid2_1,'MSD-4*epsilon^2 (pix^2)	NaN	s.e.v. (pix^2)	Neff (#)	Abs tau (sec)	VAR(dx) (pix^2)	VAR(dy) (pix^2)
');
rows = length(MSD_tabs_epsilon);
for k = 1:rows
    fprintf(fid2_1,'%f	%f	%f	%.0f	%.0f	%.0f	%.0f
',MSD_tabs_epsilon(k,:));
end
fclose(fid2_1);

%**************************************************************************
save 'MSD_tbin_epsilon.mat' MSD_tbin_epsilon
fid2_2 = fopen('MSD_tbin_epsilon.txt','wt');
fprintf(fid2_2,'MSD-4*epsilon^2 (pix^2)	NaN	s.e.v. (pix^2)	Neff (#)	Bin tau (sec)	VAR(dx) (pix^2)	VAR(dy) (pix^2)\n');
rows = length(MSD_tbin_epsilon);
for k = 1:rows
    fprintf(fid2_2,'%f	%f	%f	%.0f	%.0f	%.0f
',MSD_tbin_epsilon(k,:));
end
fclose(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'
	Function completed without errors/warnings
');
    fprintf(fid,'
	Function completed without errors/warnings
');
end

% Update log file that function is completed:
fprintf(1,'
%s completed
',func_name);
fprintf(fid,'
%s completed
',func_name);

end

Plot_Mean_Differentials_v5.m

% Steven J. Henry
% 08/11/2012
%**************************************************************************
% PURPOSE:
% This function plots mean squared displacements of all cells as a function
% of lag time (tau) in terms of both absolute and binned time intervals.
% The elapsed time values are lag times or "taus". These time intervals
% were either in terms of absolute time differences or binned time
% differences. The binning strategy is to help improve the number of
% samples per mean computation.
%
% REMARKS:
% Placing all plotting associated with the MSD arrays into a separate
% function allows easier plotting manipulation without having to
% reconstruct the MSD arrays each time.
%
% ASSUMPTIONS:
%
% INPUT:
% SD = array containing squared displacement differentials and linear time
%   col 1 = track ID to which this interval belongs
%   col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
%   col 3 = tau absolute time (sec)
%   col 4 = tau binned time (sec)
%   col 5 = dx (pix)
%   col 6 = dy (pix)
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
% col 1 = MSD(tau) = VAR(dr) (pixels^2)
% col 2 = obsolete "NaN"
% col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
% col 4 = Neff (number of independent observations)
% col 5 = tau absolute elapsed time (sec)
% col 6 = VAR(dx) (pix^2)
% col 7 = VAR(dy) (pix^2)
% MSD_tbin = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
% col 1 = MSD(tau) = VAR(dr) (pixels^2)
% col 2 = obsolete 'NaN'
% col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
% col 4 = Neff (number of independent observations)
% col 5 = tau binned elapsed time (sec)
% col 6 = VAR(dx) (pix^2)
% col 7 = VAR(dy) (pix^2)
% pixel_calib = user-supplied microns/pixel
% t_max = maximum experimental imaging time for which empirical data is
% being used to compute MSD values.
% run_title = user specified string description of experimental condition
% epsilon_flag = 0 if MSD data being filtered is not corrected for
% random noise or 1 if MSD data being filtered is corrected for random
% noise
% fid = file ID of log file to which progress is recorded

function [] = Plot_Mean_Differentials_v5(SD, MSD_tabs, MSD_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

%******************************
% Begin plot that overlays MSD values on SD of each track:
%******************************

% Determine indices of start and stop positions of each unique track in the
% 'SD' array:
[junk ind_start] = unique(SD(:,1),'first');
clear junk
[junk ind_stop] = unique(SD(:,1),'last');
clear junk

% Make sure lengths of start and stop vectors are same:
if length(ind_start) ~= length(ind_stop)
    fprintf(1,'\n\nWARNING: # Unique Track IDs start positions ~= # stop positions\n');
    fprintf(fid,'\n\nWARNING: # Unique Track IDs start positions ~= # stop positions\n');
    warn = 1;
end

% Total number of tracks to be analyzed:
num_tracks = length(ind_start);

for j = 1:num_tracks
    % Define start and stop position:
    r_start = ind_start(j);
    r_stop = ind_stop(j);

    % Create vectors of track data:
d2 = SD(r_start:r_stop,2);
t_abs = SD(r_start:r_stop,3);
t_bin = SD(r_start:r_stop,4);
% Convert squared displacement (in pixels^2) to microns^2. Note:
% 'pixel_calib' has units micron/pixel:
136 \[ d_2 = d_2 \times \text{pixel\_calib}^2; \]
137
% Convert time vectors to minutes from seconds:
139 \[ t_{\text{abs}} = t_{\text{abs}}/60; \text{\% min} \]
140 \[ t_{\text{bin}} = t_{\text{bin}}/60; \text{\% min} \]
141
% Plot squared displacement vs. elapsed time (absolute)
143 \% Units are microns^2 and minutes
144 h1\_sub(1) = subplot(1,2,1);
145 hold on
146 plot([t_{\text{abs}},d_2],'LineStyle','none','Marker','.','LineWidth',1,'Color','b','HandleVisibility','off');
147
% Plot squared displacement vs. elapsed time (binned)
149 \% Units are microns^2 and minutes
151 h1\_sub(2) = subplot(1,2,2);
152 hold on
153 plot([t_{\text{bin}},d_2],'LineStyle','none','Marker','.','LineWidth',1,'Color','b','HandleVisibility','off');
154
end
156
% Set ordinate upperbound as maximum squared displacement value:
158 \[ d_{2\_\text{max\_h1}} = \max(SD(:,2))\times\text{pixel\_calib}^2; \text{\% microns^2} \]
159
% Prepare the mean data for plotting:
161 t_{\text{abs}} = MSD\_tabs(:,5)/60; \text{\% min}
162 m_{\text{abs}} = MSD\_tabs(:,1)\times\text{pixel\_calib}^2; \text{\% microns^2}
163 sev_{\text{abs}} = MSD\_tabs(:,3)\times\text{pixel\_calib}^2; \text{\% microns^2}
164 t_{\text{bin}} = MSD\_tbin(:,5)/60; \text{\% min}
166 m_{\text{bin}} = MSD\_tbin(:,1)\times\text{pixel\_calib}^2; \text{\% microns^2}
167 sev_{\text{bin}} = MSD\_tbin(:,3)\times\text{pixel\_calib}^2; \text{\% microns^2}
168
% Overlay mean series:
170 subplot(h1\_sub(1));
171 plot([t_{\text{abs}},m_{\text{abs}}],'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
172 h1\_legend\_handle = legend('<r^2>','Location','NorthWest');
173 set(h1\_legend\_handle,'FontName','Arial','FontSize',12);
174 subplot(h1\_sub(2));
176 plot([t_{\text{bin}},m_{\text{bin}}],'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
177 h1\_legend\_handle = legend('<r^2>','Location','NorthWest');
178 set(h1\_legend\_handle,'FontName','Arial','FontSize',12);
179
% Set axes properties:
181 set(h1\_sub,'ylim',[0 d_{2\_\text{max\_h1}}]);
182 set(h1\_sub,'xlim',[0 t_{\text{max}}]);
183 set(h1\_sub,'FontName','Arial');
184 set(h1\_sub,'FontSize',14);
185 h1\_x\_axis\_handles = cell2mat(get(h1\_sub,'xlabel'));
186 set(h1\_x\_axis\_handles(1),'String',\'Absolute \tau (min)\','FontName','Arial','FontSize',16);
187 set(h1\_x\_axis\_handles(2),'String',\'Binned \tau (min)\','FontName','Arial','FontSize',16);
188
190 h1\_y\_axis\_handles = cell2mat(get(h1\_sub,'ylabel'));
191 if epsilon\_flag == 0
192 ord\_label = '<br^2> (\mum^2);'
193 else
194 ord\_label = '<br^2> - 4\epsilon^2 (\mum^2);'
195 end
196 set(h1\_y\_axis\_handles,'String',ord\_label,'FontName','Arial','FontSize',16);
% Set ordinate upper bound as maximum mean squared displacement value:
    temp = zeros(1,2);
    temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
    temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
    d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2

% Plot data:
    h4 = figure;
    h4_sub(1) = subplot(1,2,1);
    errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
    hold on
    h4_sub(2) = subplot(1,2,2);
    errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
    hold on

% Set axes properties:
    set(h4_sub,'ylim',[0 d2_max_h4_h5]);
    set(h4_sub,'xlim',[0 t_max]);
    set(h4_sub,'FontName','Arial');
    set(h4_sub,'FontSize',14);
    h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
    set(h4_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
    set(h4_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
    h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
    if epsilon_flag == 0
        ord_label = '<r^2> \pm s.e.v. (\mum^2)';
    elseif epsilon_flag == 1
        ord_label = '<r^2> - 4*\epsilon^2 \pm s.e.v. (\mum^2)';
    end
    set(h4_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
    h4_title_handles = cell2mat(get(h4_sub,'title'));
    set(h4_title_handles,'String',run_title,'FontName','Arial','FontSize',16);
    % Save figure window generated:
    saveas(h4, 'MSD_sev.fig', 'fig');

%***************************************

% On log-log axes negative values result in output warnings to user. To
% avoid this filter for msd - s.d. (lower bounds) that result in negative
% values.
L_sev_abs = sev_abs;
for i = 1:length(L_sev_abs)
    if m_abs(i)-L_sev_abs(i) < 0
        L_sev_abs(i) = 0;
    end
end
L_sev_bin = sev_bin;
for i = 1:length(L_sev_bin)
    if m_bin(i) - L_sev_bin(i) < 0
        L_sev_bin(i) = 0;
    end
end

% Plot data:
h5 = figure;
h5_sub(1) = subplot(1,2,1);
errorbar(t_abs,m_abs,L_sev_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
h5_sub(2) = subplot(1,2,2);
errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on

% Set axes properties:
set(h5_sub,'ylim',[1 d2_max_h4_h5]);
set(h5_sub,'xlim',[1 t_max]);
set(h5_sub,'yscale','log');
set(h5_sub,'xscale','log');
set(h5_sub,'YMinorTick','on');
set(h5_sub,'XMinorTick','on');
set(h5_sub,'FontName','Arial');
set(h5_sub,'FontSize',14);

h5_x_axis_handles = cell2mat(get(h5_sub,'xlabel'));
set(h5_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
set(h5_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);

h5_y_axis_handles = cell2mat(get(h5_sub,'ylabel'));
if epsilon_flag == 0
    ord_label = '<r^2> \pm s.e.v. (\mu^m^2)';
elseif epsilon_flag == 1
    ord_label = '<r^2> - 4*\epsilon^2 \pm s.e.v. (\mu^m^2)';
end
set(h5_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);

h5_title_handles = cell2mat(get(h5_sub,'title'));
set(h5_title_handles,'String',run_title,'FontName','Arial','FontSize',16);

% Save figure window generated:
saveas(h5, 'MSD_sev_loglog.fig', 'fig');

%*******************************
% Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data

% aspect ratio to [1 1 1].

% Plot data:
h6 = figure;
h6_plot = axes;
hold on
errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on

% Set axes properties:
if d2_max_h4_h5 >= t_max
    axis_lim = d2_max_h4_h5;
else
    axis_lim = t_max;
end

errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);

% Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data
% aspect ratio to [1 1 1].

% Plot data:
h6 = figure;
h6_plot = axes;
hold on
errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on

% Set axes properties:
if d2_max_h4_h5 >= t_max
    axis_lim = d2_max_h4_h5;
else
    axis_lim = t_max;
end
set(h6_plot,’ylim’,[1 axis_lim]);
set(h6_plot,’xlim’,[1 axis_lim]);
set(h6_plot,’yscale’,’log’);
set(h6_plot,’xscale’,’log’);
set(h6_plot,’DataAspectRatio’,[1 1 1]);
set(h6_plot,’YMinorTick’,’on’);
set(h6_plot,’XMinorTick’,’on’);
set(h6_plot,’FontName’,’Arial’);
set(h6_plot,’FontSize’,14);
set(h6_plot,’box’,’on’);

h6_plot_x_axis_handle = xlabel(’\tau (min)’);
set(h6_plot_x_axis_handle,’FontName’,’Arial’,’FontSize’,16);

if epsilon_flag == 0
    ord_label = ’<r^2> \pm s.e.v. (\mu m^2)’;
elseif epsilon_flag == 1
    ord_label = ’<r^2> - 4*\epsilon^2 \pm s.e.v. (\mu m^2)’;
end

h6_plot_y_axis_handle = ylabel(ord_label);
set(h6_plot_y_axis_handle,’FontName’,’Arial’,’FontSize’,16);

h6_plot_title_handle = title(run_title);
set(h6_plot_title_handle,’FontName’,’Arial’,’FontSize’,16);

% Save figure window generated:
saveas(h6, ’MSD_sev_loglog_final.fig’, ’fig’);
%******************
% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,’n	Function completed without errors/warnings
’);
    fprintf(fid,’n	Function completed without errors/warnings
’);
end
% Update log file that function is completed:
fprintf(1,’n%s completed
’,func_name);
fprintf(fid,’n%s completed
’,func_name);
end

% Steven J. Henry
% 08/11/2012
%**************************************************************************
% PURPOSE:
% This function filters a supplied array of mean squared displacement and
% elapsed time values (taus) and produces an array with the same structure
% but only containing data corresponding to tau values between and
% including user specified bounds.
% REMARKS:
% n/a
% ASSUMPTIONS:
% n/a
% INPUT:
% MSD_tabs_full = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
% col 1 = MSD(tau) = VAR(dr) (pixels^2)
% col 2 = obsolete ’NaN’
% col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
% col 4 = Neff (number of independent observations)
% col 5 = tau absolute elapsed time (sec)

Filter_Mean_Differentials_v4.m
% col 6 = VAR(dx) (pix^2)
% col 7 = VAR(dy) (pix^2)
% MSD_tbin_full = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
% col 1 = MSD(tau) = VAR(dr) (pixels^2)
% col 2 = obsolete 'NaN'
% col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
% col 4 = Neff (number of independent observations)
% col 5 = tau binned elapsed time (sec)
% col 6 = VAR(dx) (pix^2)
% col 7 = VAR(dy) (pix^2)
% fit_tau_min = user supplied minimum tau interval in (min)
% fit_tau_max = user-supplied maximum tau interval in (min)
% epsilon_flag = 0 if MSD data being filtered is not corrected for
% random noise or 1 if MSD data being filtered is corrected for random
% noise
% iid = file ID of log file to which progress is recorded
% OUTPUT:
% MSD_tabs_part = array containing mean squared displacements and
% corresponding lag time taus for a given experimental condition for those
% taus that are less than or equal to the user-specified 'tau_max'. Taus
% are in terms of absolute differences. The array has the same structure as
% before.
% MSD_tbin_part = array containing mean squared displacements and
% corresponding lag time taus for a given experimental condition for those
% taus that are less than or equal to the user-specified 'tau_max'. Taus
% are in terms of binned differences. The array has the same structure as
% before.
%**************************************************************************

function [MSD_tabs_part MSD_tbin_part] = Filter_Mean_Differentials_v4(MSD_tabs_full, MSD_tbin_full, fit_tau_min, fit_tau_max, epsilon_flag, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,'%
%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Convert tau boundaries to sec from min:
fit_tau_min = fit_tau_min*60; % sec
fit_tau_max = fit_tau_max*60; % sec

% Identify row position of all elapsed time intervals that pass the filter
% 'tau_max' (i.e. that are less than or equal to 'tau_max'):
lib_pass_ind = fit_tau_min<=MSD_tabs_full(:,5) & MSD_tabs_full(:,5)<=fit_tau_max;

% Double check manipulations:
if any(MSD_tabs_part(:,5) < fit_tau_min) == 1
    fprintf(fid,'%

**WARNING: tau entry in "MSD_tabs_part" is less than "fit_tau_min" after filtering.
')
    war = 1;
    keyboard
end
if any(MSD_tabs_part(:,5) > fit_tau_max) == 1
    fprintf(fid,'%

**WARNING: tau entry in "MSD_tabs_part" is greater than "fit_tau_max" after filtering.
')
    war = 1;
    keyboard
end

MSD_tabs_part = MSD_tabs_full(t_abs_pass_ind,:); MSD_tbin_part = MSD_tbin_full(t_bin_pass_ind,:);
warn = 1;
keyboard
end

if any(MSD_tbin_part(:,5) < fit_tau_min) == 1
    fprintf(1,'\n\tWARNING: tau entry in "MSD_tbin_part" is less than "fit_tau_min" after filtering.\n')
    fprintf(fid,'\n\tWARNING: tau entry in "MSD_tbin_part" is less than "fit_tau_min" after filtering.\n')
    warn = 1;
    keyboard
end

if any(MSD_tbin_part(:,5) > fit_tau_max) == 1
    fprintf(1,'\n\tWARNING: tau entry in "MSD_tbin_part" is greater than "fit_tau_max" after filter.\n')
    fprintf(fid,'\n\tWARNING: tau entry in "MSD_tbin_part" is greater than "fit_tau_max" after filter.\n')
    warn = 1;
    keyboard
end

if length(t_abs_pass_ind)==length(MSD_tabs_part(:,1))
    fprintf(1,'\n\tWARNING: length(t_abs_pass_ind) ~= length(MSD_tabs_part)\n')
    fprintf(fid,'\n\tWARNING: length(t_abs_pass_ind) ~= length(MSD_tabs_part)\n')
    warn = 1;
    keyboard
end

if length(t_bin_pass_ind)==length(MSD_tbin_part(:,1))
    fprintf(1,'\n\tWARNING: length(t_bin_pass_ind) ~= length(MSD_tbin_part)\n')
    fprintf(fid,'\n\tWARNING: length(t_bin_pass_ind) ~= length(MSD_tbin_part)\n')
    warn = 1;
    keyboard
end

% Save filtered 'MSD' arrays in .mat and .txt format:
if epsilon_flag == 0
    MSD_tabs_header = 'MSD(tau) (pix^2)/ts.d. (pix^2)/ts.e.v. (pix^2)/tNeff (#)/tAbs tau (sec)/tVAR(dx) (pix^2)/tVAR(dy) (pix^2)\n';
    MSD_tbin_header = 'MSD(tau) (pix^2)/ts.d. (pix^2)/ts.e.v. (pix^2)/tNeff (#)/tBin tau (sec)/tVAR(dx) (pix^2)/tVAR(dy) (pix^2)\n';
    MSD_tabs_part = MSD_tabs_part';
    MSD_tbin_part = MSD_tbin_part';
    MSD_tabs_part = MSD_tabs_part';
    MSD_tbin_part = MSD_tbin_part';
else epsilon_flag == 1
    MSD_tabs_header = 'MSD(tau)-4*epsilon^2 (pix^2)/ts.d. (pix^2)/ts.e.v. (pix^2)/tNeff (#)/tAbs tau (sec)/tVAR(dx) (pix^2)/tVAR(dy) (pix^2)\n';
    MSD_tbin_header = 'MSD(tau)-4*epsilon^2 (pix^2)/ts.d. (pix^2)/ts.e.v. (pix^2)/tNeff (#)/tBin tau (sec)/tVAR(dx) (pix^2)/tVAR(dy) (pix^2)\n';
    MSD_tabs_part = MSD_tabs_part';
    MSD_tbin_part = MSD_tbin_part';
    MSD_tabs_part = MSD_tabs_part';
    MSD_tbin_part = MSD_tbin_part';
end

save 'MSD_tabs_filtered.mat' MSD_tabs_part
fid2_1 = fopen('MSD_tabs_filtered.txt','wt');
fprintf(fid2_1,MSD_tabs_header);
rows = length(MSD_tabs_part);
for k = 1:rows
    fprintf(fid2_1,'%f %f %f %d %d %d %d\n',MSD_tabs_part(k,:));
end
fclose(fid2_1);

save 'MSD_tbin_filtered.mat' MSD_tbin_part
fid2_2 = fopen('MSD_tbin_filtered.txt','wt');
fprintf(fid2_2,MSD_tbin_header);
rows = length(MSD_tbin_part);
for k = 1:rows
    fprintf(fid2_2,'%f %f %f %d %d %d %d\n',MSD_tbin_part(k,:));
end
fclose(fid2_2);

if warn == 0
    fprintf(1,'\n\tFunction completed without errors/warnings\n');
    fprintf(fid,'\n\tFunction completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(1,'%s completed\n',func_name);
```matlab
fprintf(fid,'\n%s completed\n',func_name);
end

function [fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v11(MSD_tabs, MSD_tbin, pixel_calib, fid)
```

```matlab
% The following function calculates root-mean-square speed 'S' and
% persistence time 'P' from a nonlinear curve fitting algorithm performed
% on \( <d^2> \) vs. time interval data using the Lauffenburger persistent random
% walk model (6-35a) on p.312 of "Receptors: Models for binding,
% trafficking, and signaling" 1993 Oxford University Press.
```

% PURPOSE:
% The following function calculates root-mean-square speed 'S' and
% persistence time 'P' from a nonlinear curve fitting algorithm performed
% on \( <d^2> \) vs. time interval data using the Lauffenburger persistent random
% walk model (6-35a) on p.312 of "Receptors: Models for binding,
% trafficking, and signaling" 1993 Oxford University Press.

% ASSUMPTIONS:
% Model is appropriate for mode of migration being analyzed.

% INPUT:
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau absolute elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% MSD_tbin = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau binned elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% pixel_calib = user-supplied microns/pixel
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% fit_BRW_tabs = (BRW = biased random walk) array with length =
% length(MSD_tabs) containing fit data generated using best-fit S and P
% parameters having structure:
%   col 1 = fit mean squared displacement (pixels^2)
%   col 2 = tau absolute elapsed time (sec)
% fit_BRW_tbin = (BRW = biased random walk) array with length =
% length(MSD_tabs) containing fit data generated using best-fit S and P
% parameters having structure:
%   col 1 = fit mean squared displacement (pixels^2)
%   col 2 = tau binned elapsed time (sec)
% Sout = best fit speed value (um/min)
% Pout = best fit persistence value (min)
% muout = random motility coefficient from best-fit values =
% 0.5*Sout^2*Pout (um^2/min)

%**************************************************************************
```
% Update log file that function is running:
fprintf(1,'n%gs running ...
',func_name);
fprintf(fid,'n%gs running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

for i = 1:2
  MSD = MSD_tabs;
else i == 2
  MSD = MSD_tbin;
end

% Generate an initial guess for the root mean squared speed in
% pixels/sec. Use as the guess the instantaneous speed corresponding to
% the <d^2> value that has the most number of samples ("Neff")
% have the most confidence in):
[max_Neff, ind_max_Neff] = max(MSD(:,4));
d2 = MSD(ind_max_Neff,1); % pix^2
tau = MSD(ind_max_Neff,5); % sec
d = sqrt(d2); % pix
S0 = d/tau; % pixel/sec

% Generate an initial guess for persistence time in sec from the
% long-time approximation that:
% P = <d^2>/(2*S^2*tau) using S = S0
[max_tau, ind_max_tau] = max(MSD(:,5));
long_d2 = MSD(ind_max_tau,1); % pix^2
long_tau = MSD(ind_max_tau,5); % sec
if long_tau ~= max_tau
    fprintf(1,'nWARNING: index reported for "max_tau" does not correspond to "max_tau" in MSD array/n');
    fprintf(fid,'nWARNING: index reported for "max_tau" does not correspond to "max_tau" in MSD array/n');
end
P0 = long_d2/(2*S0^2*long_tau); % sec
if P0 < 0
    P0 = tau; % sec
    fprintf(1,'nWARNING: P0 < 0 so set P0 = %s (sec)/n',num2str(P0));
    fprintf(fid,'nWARNING: P0 < 0 so set P0 = %s (sec)/n',num2str(P0));
end
if P0 > long_tau
    P0 = long_tau; % sec
    fprintf(1,'nWARNING: P0 > max tau so set P0 = %s (sec)/n',num2str(P0));
    fprintf(fid,'nWARNING: P0 > max tau so set P0 = %s (sec)/n',num2str(P0));
end

%******************
% Use 'lsqcurvefit' function to determine S and P. Create a row vector
% to hold the initial guesses at the fit parameters S and P:
para0 = [S0,P0];

% Define empty matrices of lower and upper bounds. Utilization of
% Levenberg_Marquardt algorithm does not accomodate boundary constraints
% on solver (see discussion below). Thus we pass empty boundaries to
% the solver algorithm.
lb = [];
ub = [];

% Define 'options'. Note: the Levenberg_Marquardt algorithm was
% selected for two reasons. First it was the default algorithm being
% used by a grandfathered code from Brendon Ricart in 2010. Second
% discussion of the algorithm in the Help section of the Optimization Toolbox titled "Least Squares (Model Fitting)" turned up that this algorithm is not the most efficient but is robust especially when the solution has a nonzero residual which I anticipate to be the case for my empirical data:

% "The poorer efficiency is partly because the Gauss-Newton method is generally more effective when the residual is zero at the solution. However, such information is not always available beforehand, and the increased robustness of the Levenberg-Marquardt method compensates for its occasional poorer efficiency."

options = optimset('MaxFunEvals', 2000, 'Algorithm','levenberg-marquardt');

% Generate an "anonymous" function that contains the Biased Random Walk model. Use as the weights vector the Neff values corresponding to each data point.
weights = MSD(:,4);

% weights = ones(size(MSD(:,4))); % use this for an unweighted fit

BRW_fun = @(para,t) weights.*((2^2*para(1)^2*para(2)*t-para(2)^2*2^2*(1-exp(-t./para(2)))));

[para, resnorm, residual, exitflag] = lsqcurvefit(BRW_fun, para0, MSD(:,5), MSD(:,1).*weights, lb, ub, options);

% Fit root mean squared speed (pixels/sec) and persistence time (sec):
S = para(1);
P = para(2);

% Send S and P out of function with physical units:
Sout = S*pixel_calib*60;
Pout = P/60;

% Compute random motility coefficient:
mu = 0.5*(S)^2*P;
muout = 0.5*(Sout)^2*Pout;

% Reserve memory for an array that will hold the theoretical MSD values using returned S and P values and corresponding tau:
rows = length(MSD(:,1));
F = zeros(rows,2);

for j = 1:rows
    tau = MSD(j,5);
    F(j,1) = 2*S^2*(P*tau-P^2*(1-exp(-tau/P)));
    F(j,2) = tau;
end

if i == 1
    fit_tabs = F;
    tau_type = 'ABSOLUTE';
elseif i == 2
    fit_tbin = F;
    tau_type = 'BINNED';
end

% Update log file with progress:
fprintf(1,'
	Processing %s tau MSD data:
',tau_type);
fprintf(1,'	*****
');
fprintf(1,'	S0 guess supplied = %0.4f pixels/sec = %0.4f um/min
',S0,S0*pixel_calib*60);
fprintf(1,'	P0 guess supplied = %0.4f sec = %0.4f min
',P0,P0/60);
fprintf(1,'	*****
');
fprintf(1,'	squared 2-norm of the residual (resnorm) at solution = %.4E
',resnorm);
fprintf(1,'	exit flag = %.0f
',exitflag);
fprintf(1,'	*****
');
fprintf(1,'	S weighted fit returned = %0.4f pixels/sec = %0.4f um/min
',S,Sout);
fprintf(1,'	P weighted fit returned = %0.4f sec = %0.4f min
',P,Pout);
fprintf(1,'	*****
');
fprintf(1,'	Random motility coefficient (mu) = %0.4f pixels^2/sec = %0.4f um^2/min
',mu,muout);
fprintf(1,'	*****
');
fprintf(fid,'
	Processing %s tau MSD data:
',tau_type);
fprintf(fid,'%*****n');
fprintf(fid,'%S0 guess supplied = %0.4f pixels/sec = %0.4f um/min\n',S0,S0*pixel_calib*60);
fprintf(fid,'%P0 guess supplied = %0.4f sec = %0.4f min\n',P0,P0/60);
fprintf(fid,'%*****n');
fprintf(fid,'%squared 2-norm of the residual (resnorm) at solution = %.4E\n',resnorm);
fprintf(fid,'%exit flag = %.0f\n',exitflag);
fprintf(fid,'%*****n');
fprintf(fid,'%S weighted fit returned = %0.4f pixels/sec = %0.4f um/min\n',S,Sout);
fprintf(fid,'%P weighted fit returned = %0.4f sec = %0.4f min\n',P,Pout);
fprintf(fid,'%*****n');
fprintf(fid,'%Random motility coefficient (mu) = %0.4f pixels^2/sec = %0.4f um^2/min\n',mu,muout);
fprintf(fid,'%*****n');
end

% Ensure number of data entries in 'fit' arrays is consistent with MSD:
if length(fit_tabs(:,1))~=length(MSD_tabs(:,1))
fprintf(fid1,'%WARNING: length(fit_tabs) ~= length(MSD_tabs)\n');
warn = 1;
end
if length(fit_tbin(:,1))~=length(MSD_tbin(:,1))
fprintf(fid1,'%WARNING: length(fit_tbin) ~= length(MSD_tbin)\n');
warn = 1;
end

% Rename fit_tabs and fit_tbin to note that they are from the biased random
% walk model (BRW):
fit_BRW_tabs = fit_tabs;
fit_BRW_tbin = fit_tbin;

% Save 'fit' arrays in .mat and .txt format:
save 'fit_BRW_tabs.mat' fit_BRW_tabs
fid2_1 = fopen('fit_BRW_tabs.txt','wt');
fprintf(fid2_1,'Weighted Fit MSD (pix^2)	Abs. tau (sec)\n');
rows = length(fit_BRW_tabs);
for k = 1:rows
fprintf(fid2_1,'%f	%.0f\n',fit_BRW_tabs(k,:));
end
close(fid2_1);
save 'fit_BRW_tbin.mat' fit_BRW_tbin
fid2_2 = fopen('fit_BRW_tbin.txt','wt');
fprintf(fid2_2,'Weighted Fit MSD (pix^2)	Bin. tau (sec)\n');
rows = length(fit_BRW_tbin);
for k = 1:rows
fprintf(fid2_2,'%f	%.0f\n',fit_BRW_tbin(k,:));
end
close(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
fprintf(fid1,'%nlFunction completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(fid1,'%nl%s completed\n',func_name);
fprintf(fid1,'%n%','%s completed\n',func_name);
end

Power_Law_v4.m
% Steven J. Henry
% 11/03/2011
% PURPOSE:
% The following function applies a power-law fit to MSD vs. tau data. It
% solves for the parameters 'A' and 'alpha' where: MSD = A*tau^alpha
% %
% REMARKS:
% We can determine A and alpha using MATLAB's intrinsic 'lscov' routine
% which fits a linear function in the least squares sense. The goal is to
% identify constants 'A' and 'alpha' that fit empirical MSD vs. tau data
% such that:
% MSD = A*tau^alpha
% %
% Using properties of logs:
% log10(MSD) = log10(A*tau^alpha)
% log10(MSD) = log10(A)+log10(tau^alpha)
% log10(MSD) = log10(A)+alpha*log10(tau)
% rearranging:
% log10(MSD) = alpha*log10(tau) + log(A)
% which has the familiar form:
% Y = mX + b
% where Y = log10(MSD), m = alpha, X = log10(tau), b = log10(A) => A = 10^b
% %
% ASSUMPTIONS:
% Model is appropriate for mode of motility.
% %
% INPUT:
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
% col 1 = MSD(tau) = VAR(dr) (pixels^2)
% col 2 = obsolete 'NaN'
% col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
% col 4 = Neff (number of independent observations)
% col 5 = tau absolute elapsed time (sec)
% col 6 = VAR(dx) (pix^2)
% col 7 = VAR(dy) (pix^2)
% pixel_calib = user-supplied microns/pixel
% fid = file ID of log file to which progress is recorded
% %
% OUTPUT:
% fit_PL_tabs = (PL = power law) array with length = length(MSD_tabs)
% containing fit data generated using best-fit 'A' and 'alpha' parameters
% having structure:
% col 1 = fit mean squared displacement (pixels^2)
% col 2 = tau absolute elapsed time (sec)
% fit_PL_tbin = (PL = power law) array with length = length(MSD_tbin)
% containing fit data generated using best-fit 'A' and 'alpha' parameters
% having structure:
% col 1 = fit mean squared displacement (pixels^2)
% col 2 = tau binned elapsed time (sec)
% Aout = best fit coefficient value (um^2/min^alpha)
% alphaout = best fit power value (unitless)
%**************************************************************************

function [fit_PL_tabs fit_PL_tbin Aout alphaout] = Power_Law_v4(MSD_tabs, MSD_tbin, pixel_calib, fid)
% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'\n%s running ...\n',func_name);
fprintf(fid,'\n%s running ...\n',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

for i = 1:2
  if i == 1
    MSD = MSD_tabs;
  elseif i == 2
    MSD = MSD_tbin;
  end

% Take logs of data:
Y = log10(MSD(:,1));
X = log10(MSD(:,5));

%***********
% Perform weighted fitting:
%***********
% Define a weight vector 'w' that is the # independent observations
% used to generate that data point (Neff)
w = MSD(:,4);  % use for unweighted fit
% Use of 'lscov' requires considering the matrix form of the system of
% equations. We want to think in terms of a system Cp = Y.
% C = coefficient matrix. Col 1 = vector X. Col 2 = ones vector.
% p = parameter vector. p(1) = m. p(2) = b.
C = [X ones(size(X))];

% stdx, mse, and S are returned for possible use with polyval in future
[p] = lscov(C,Y,w);

% versions
m = p(1);  % slope
b = p(2);  % intercept

% Solve the weighted parameters A and alpha you actually care about:
alpha = m;  % unitless
alphaout = alpha;
A = 10^b;  % units of pix^2/sec^alpha

% Unit conversion:
Aunit = A*(pixel_calib^2)*(60^alpha);
Aout = Aunit;

%***********
% Reserve memory for an array that will hold the theoretical MSD values
% using returned parameters:
rows = length(MSD(:,1));
F = zeros(rows,3);

for j = 1:rows
  tau = MSD(j,5);
  F(j,1) = A*tau*alpha;
  F(j,2) = tau;
end

if i == 1
  fit_tabs = F;
  tau_type = 'ABSOLUTE';
elseif i == 2
  fit_tbin = F;
tau_type = 'Binned';
end

% Update log file with progress:
fprintf(1,'\n\nProcessing %s tau MSD data:\n\n',tau_type);
fprintf(1,'\n*****
');
fprintf(1,'WEIGHTED Fit Parameters:\n');
fprintf(1,'\tA = %0.4f (pix^2/sec^alpha)\n',A);
fprintf(1,'\tA = %.04f (um^2/min^alpha)\n',Aunit);
fprintf(1,'\talpha = %0.4f (unitless)\n',alpha);
fprintf(1,'*****
');
fprintf(fid,'\n\nProcessing %s tau MSD data:\n\n',tau_type);
fprintf(fid,'\n*****
');
fprintf(fid,'WEIGHTED Fit Parameters:\n');
fprintf(fid,'\tA = %0.4f (pix^2/sec^alpha)\n',A);
fprintf(fid,'\tA = %.04f (um^2/min^alpha)\n',Aunit);
fprintf(fid,'\talpha = %0.4f (unitless)\n',alpha);
fprintf(fid,'*****
');

% Ensure number of data entries in 'fit' arrays is consistent with MSD:
if length(fit_tabs(:,1))~=length(MSD_tabs(:,1))
    fprintf(1,'\n\nWARNING: length(fit_tabs) \neq length(MSD_tabs)\n')
    fprintf(fid,'\n\nWARNING: length(fit_tabs) \neq length(MSD_tabs)\n')
    warn = 1;
end
if length(fit_tbin(:,1))~=length(MSD_tbin(:,1))
    fprintf(fid,'\n\nWARNING: length(fit_tbin) \neq length(MSD_tbin)\n')
    warn = 1;
end

% Rename fit_tabs and fit_tbin to denote that they are from the Power Law model (PL):
fit_PL_tabs = fit_tabs;
fit_PL_tbin = fit_tbin;

% Save 'fit' arrays in .mat and .txt format:
save 'fit_PL_tabs.mat' fit_PL_tabs
fid2_1 = fopen('fit_PL_tabs.txt','wt');
fprintf(fid2_1,'\n\n\nWeighted Fit MSD (pix^2)\tAbs. tau (sec)\n');
rows = length(fit_PL_tabs);
for k = 1:rows
    fprintf(fid2_1,'%f\t%.0f\n',fit_PL_tabs(k,:));
end
fclose(fid2_1);
save 'fit_PL_tbin.mat' fit_PL_tbin
fid2_2 = fopen('fit_PL_tbin.txt','wt');
fprintf(fid2_2,'\n\n\nWeighted Fit MSD (pix^2)\tBin. tau (sec)\n');
rows = length(fit_PL_tbin);
for k = 1:rows
    fprintf(fid2_2,'%f\t%.0f\n',fit_PL_tbin(k,:));
end
fclose(fid2_2);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'\n\nFunction completed without errors/warnings\n');
    fprintf(fid,'\n\nFunction completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(1,'\n\n%s completed\n',func_name);
fprintf(fid,'\n\n%s completed\n',func_name);
% PURPOSE:
% This function overlays the theoretical fit data from 'SandP.m' on top
% of the empirical MSD data.

% REMARKS:
% n/a

% ASSUMPTIONS:
% n/a

% INPUT:
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau absolute elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% MSD_tbin = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau binned elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% fit_tabs = array with length = length(MSD_tabs) containing fit data
% generated using best-fit S and P parameters having structure:
%   col 1 = fit mean squared displacement (pixels^2)
%   col 2 = tau absolute elapsed time (sec)
% fit_tbin = array with length = length(MSD_tabs) containing fit data
% generated using best-fit S and P parameters having structure:
%   col 1 = fit mean squared displacement (pixels^2)
%   col 2 = tau binned elapsed time (sec)
% pixel_calib = user-supplied microns/pixel
% t_max = user-supplied imaging duration (min)
% run_title = user specified string description of experimental condition
% epsilon_flag = 0 if MSD data being filtered is not corrected for random
% noise or 1 if MSD data being filtered is corrected for random
% noise
% fid = file ID of log file to which progress is recorded

% OUTPUT:

function [] = Plot_SandP_Fit_v6(MSD_tabs, MSD_tbin, fit_tabs, fit_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,'n%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Prepare the data for plotting:
t_abs = MSD_tabs(:,5)/60; % min
m_abs = MSD_tabs(:,1)*pixel_calib^2; % microns^2
sev_abs = MSD_tabs(:,3)*pixel_calib^2; % microns^2
f_t_abs = fit_tabs(:,2)/60; % min
f_m_abs = fit_tabs(:,1)*pixel_calib^2; % microns^2

% Prepare the data for binned plots:
t_bin = MSD_tbin(:,5)/60; % min
m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
f_t_bin = fit_tbin(:,2)/60; % min
f_m_bin = fit_tbin(:,1)*pixel_calib^2; % microns^2

%******************************
% Plot mean +/- s.e.v. (linear-linear axes)
% Set ordinate upperbound as maximum mean squared displacement value:
temp = zeros(1,2);
temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2

% Plot data:
h4 = figure;
h4_sub(1) = subplot(1,2,1);
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_abs,f_m_abs,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h4_legend_handle,'FontName','Arial','FontSize',12);

h4_sub(2) = subplot(1,2,2);
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h4_legend_handle,'FontName','Arial','FontSize',12);

% Set axes properties:
set(h4_sub,'ylim',[0 d2_max_h4_h5]);
set(h4_sub,'xlim',[0 t_max]);
set(h4_sub,'FontName','Arial');
set(h4_sub,'FontSize',14);

h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
set(h4_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
set(h4_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);

h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
if epsilon_flag == 0
    ord_label = '<r^2> \pm s.e.v. (\mum^2)';
else
    ord_label = '<r^2> - 4*\epsilon^2 \pm s.e.v. (\mum^2)';
end
set(h4_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);

h4_title_handles = cell2mat(get(h4_sub,'title'));
set(h4_title_handles,'String',run_title;'Biased Random Walk Model','FontName','Arial','FontSize',16);
% Save figure window generated:
saveas(h4, 'Fit_BRW_sev.fig', 'fig');
%*******************************

% Plot mean +/- s.e.v. (log-log axes)
% On log-log axes negative values result in output warnings to user. To
% avoid this filter for msd - s.d. (lower bounds) that result in negative
% values.
L_sev_abs = sev_abs;
for i = 1:length(L_sev_abs)
    if m_abs(i)-L_sev_abs(i) < 0
        L_sev_abs(i) = 0;
    end
end
L_sev_bin = sev_bin;
for i = 1:length(L_sev_bin)
    if m_bin(i)-L_sev_bin(i) < 0
        L_sev_bin(i) = 0;
    end
end

% Plot data:
h5 = figure;
h5_sub(1) = subplot(1,2,1);
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_abs,f_m_abs,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h5_legend_handle,'FontName','Arial','FontSize',12);

h5_sub(2) = subplot(1,2,2);
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h5_legend_handle,'FontName','Arial','FontSize',12);

% Set axes properties:
set(h5_sub,'ylim',[1 d2_max_h4_h5]);
set(h5_sub,'xlim',[1 t_max]);
set(h5_sub,'yscale','log');
set(h5_sub,'xscale','log');
set(h5_sub,'YMinorTick','on');
set(h5_sub,'XMinorTick','on');
set(h5_sub,'FontName','Arial');
set(h5_sub,'FontSize',14);

h5_x_axis_handles = cell2mat(get(h5_sub,'xlabel'));
set(h5_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
set(h5_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);

h5_y_axis_handles = cell2mat(get(h5_sub,'ylabel'));
if epsilon_flag == 0
    ord_label = '<r^2> \pm s.e.v. (\mum^2)';
elseif epsilon_flag == 1
    ord_label = '<r^2> - 4*\epsilon^2 \pm s.e.v. (\mum^2)';
end
set(h5_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
% Save figure window generated:
saveas(h5, 'Fit_BRW_sev_loglog.fig', 'fig');

% Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data
% aspect ratio to [1 1 1].

% Plot data:
h6 = figure;
h6_plot = axes;

errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandelVisibility','on');

h6_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h6_legend_handle,'FontName','Arial','FontSize',12);

% Set axes properties:
if d2_max_h4_h5 >= t_max
    axis_lim = d2_max_h4_h5;
else
    axis_lim = t_max;
end

set(h6_plot,'ylim', [1 axis_lim]);
set(h6_plot,'xlim', [1 axis_lim]);
set(h6_plot,'yscale','log');
set(h6_plot,'xscale','log');
set(h6_plot,'DataAspectRatio',[1 1 1]);
set(h6_plot,'YMinorTick','on');
set(h6_plot,'XMinorTick','on');
set(h6_plot,'FontName','Arial');
set(h6_plot,'FontSize',14);
set(h6_plot,'box','on');

h6_plot_x_axis_handle = xlabel('	au (min)');
set(h6_plot_x_axis_handle,'FontName','Arial','FontSize',16);

if epsilon_flag == 0
    ord_label = '<r^2> \pm s.e.v. (\mum^2)';
elseif epsilon_flag == 1
    ord_label = '<r^2> - 4*\epsilon^2 \pm s.e.v. (\mum^2)';
end

h6_plot_y_axis_handle = ylabel(ord_label);
set(h6_plot_y_axis_handle,'FontName','Arial','FontSize',16);

h6_plot_title_handle = title({run_title;'Biased Random Walk Model'});
set(h6_plot_title_handle,'FontName','Arial','FontSize',16);

% Save figure window generated:
saveas(h6, 'Fit_BRW_sev_loglog_final.fig', 'fig');

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1,'\n
Function completed without errors/warnings\n');
    fprintf(fid,'\n
Function completed without errors/warnings\n');
end

% Update log file that function is completed:
% Steven J. Henry
% 07/22/2011

%**************************************************************************
% PURPOSE:
% This function overlays the theoretical fit data from 'Power_Law.m' on top
% of the empirical MSD data.
% %
% REMARKS:
% %
% ASSUMPTIONS:
% %

% INPUT:
% MSD_tabs = array containing mean squared displacements and corresponding
% lag time taus in terms of absolute differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau absolute elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% MSD_tbin = array containing mean squared displacements and corresponding
% lag time taus in terms of binned differences with the following
% structure:
%   col 1 = MSD(tau) = VAR(dr) (pixels^2)
%   col 2 = obsolete 'NaN'
%   col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
%   col 4 = Neff (number of independent observations)
%   col 5 = tau binned elapsed time (sec)
%   col 6 = VAR(dx) (pix^2)
%   col 7 = VAR(dy) (pix^2)
% fit_tabs = array with length = length(MSD_tabs) containing fit data
% fit_tbin = array with length = length(MSD_tabs) containing fit data
% generated using best-fit A and alpha parameters:
%   col 1 = fit mean squared displacement (pixels^2)
%   col 2 = tau absolute elapsed time (sec)
% t_max = user-supplied imaging duration (min)
% epsilon_flag = 0 if MSD data being filtered is not corrected for random
% noise or 1 if MSD data being filtered is corrected for random
% noise
% fid = file ID of log file to which progress is recorded

function [] = Plot_Power_Law_Fit_v4(MSD_tabs, MSD_tbin, fit Tabs, fit_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Prepare the data for plotting:
t_abs = MSD_tabs(:,5)/60; % min
m_abs = MSD_tabs(:,1)*pixel_calib^2; % microns^2
sev_abs = MSD_tabs(:,3)*pixel_calib^2; % microns^2
f_t_abs = fit_tabs(:,2)/60; % min
f_m_abs_uw = fit_tabs(:,1)*pixel_calib^2; % microns^2, unweighted fit

% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;

% Prepare the data for plotting:
t_bin = MSD_tbin(:,5)/60; % min
m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
f_t_bin = fit_tbin(:,2)/60; % min
f_m_bin_uw = fit_tbin(:,1)*pixel_calib^2; % microns^2, unweighted fit

%******************************
% Plot mean +/- s.e.v. (linear-linear axes)
% Set ordinate upperbound as maximum mean squared displacement value:
temp = zeros(1,2);
temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2

% Plot data:
h4 = figure;
h4_sub(1) = subplot(1,2,1);
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_abs,f_m_abs_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h4_legend_handle,'FontName','Arial','FontSize',12);

h4_sub(2) = subplot(1,2,2);
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f_t_bin,f_m_bin_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h4_legend_handle,'FontName','Arial','FontSize',12)

% Set axes properties:
set(h4_sub,'ylim',[0 d2_max_h4_h5]);
set(h4_sub,'xlim',[0 t_max]);
set(h4_sub,'FontSize','14');

h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
set(h4_x_axis_handles(1),'String','Absolute \&tau (min)','FontName','Arial','FontSize',16);
set(h4_x_axis_handles(2),'String','Binned \&tau (min)','FontName','Arial','FontSize',16);

h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
set(h4_y_axis_handles(1),'String','<r^2> \pm s.e.v. (\mu m^2)');</r^2> - 4*\epsilon^2 \pm s.e.v. (\mu m^2);
set(h4\_y\_axis\_handles,'String','ord\_label','FontName','Arial','FontSize',16);

h4\_title\_handles = cell2mat(get(h4\_sub,'title'));
set(h4\_title\_handles,'String',[run\_title;'Power Law Model'],'FontName','Arial','FontSize',16);

% Save figure window generated:
saveas(h4, 'Fit\_PL\_sev.fig', 'fig');

%*******************************
%*******************************
% Plot mean +/- s.e.v. (log-log axes)
% On log-log axes negative values result in output warnings to user. To
% avoid this filter for msd - s.d. (lower bounds) that result in negative
% values.

L\_sev\_abs = sev\_abs;
for i = 1:length(L\_sev\_abs)
    if m\_abs(i)-L\_sev\_abs(i) < 0
        L\_sev\_abs(i) = 0;
    end
end

L\_sev\_bin = sev\_bin;
for i = 1:length(L\_sev\_bin)
    if m\_bin(i)-L\_sev\_bin(i) < 0
        L\_sev\_bin(i) = 0;
    end
end

% Plot data:
figure;
h5\_sub(1) = subplot(1,2,1);
errorbar(t\_abs,m\_abs,sev\_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f\_t\_abs,f\_m\_abs\_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on);
h5\_legend\_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h5\_legend\_handle,'FontName','Arial','FontSize',12)

h5\_sub(2) = subplot(1,2,2);
errorbar(t\_bin,m\_bin,sev\_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
hold on
plot(f\_t\_bin,f\_m\_bin\_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
h5\_legend\_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h5\_legend\_handle,'FontName','Arial','FontSize',12)

% Set axes properties:
set(h5\_sub,'ylim',[1 d2\_max\_h4\_h5]);
set(h5\_sub,'xlim',[1 t\_max]);
set(h5\_sub,'yscale','log');
set(h5\_sub,'xscale','log');
set(h5\_sub,'YMinorTick','on');
set(h5\_sub,'XMinorTick','on');
set(h5\_sub,'FontName','Arial');
set(h5\_sub,'FontSize',14);

h5\_x\_axis\_handles = cell2mat(get(h5\_sub,'xlabel'));
set(h5\_x\_axis\_handles(1),'String','Absolute \tau (min)'),'FontName','Arial','FontSize',16); set(h5\_x\_axis\_handles(2),'String','Binned \tau (min)'),'FontName','Arial','FontSize',16);

if epsilon\_flag == 0
ord_label = '<r^2> \pm s.e.v. (\mu m^2)';

elseif epsilon_flag == 1
  ord_label = '<r^2> - 4*epsilon^2 \pm s.e.v. (\mu m^2)';
end

set(h5_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);

h5_title_handles = cell2mat(get(h5_sub,'title'));
set(h5_title_handles,'String',{run_title;'Power Law Model'},'FontName','Arial','FontSize',16);

% Save figure window generated:
saveas(h5, 'Fit_PL_sev_loglog.fig', 'fig');

%*******************************
%*******************************
% Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data
% aspect ratio to [1 1 1].

% Plot data:
h6 = figure;

errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);

hold on

plot(f_t_bin,f_m_bin_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');

h6_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
set(h6_legend_handle,'FontName','Arial','FontSize',12);

% Set axes properties:
if d2_max_h4_h5 >= t_max
  axis_lim = d2_max_h4_h5;
else
  axis_lim = t_max;
end

set(h6_plot,'ylim', [1 axis_lim]);
set(h6_plot,'xlim', [1 axis_lim]);
set(h6_plot,'yscale','log');
set(h6_plot,'xscale','log');
set(h6_plot,'DataAspectRatio',[1 1 1]);
set(h6_plot,'YMinorTick','on');
set(h6_plot,'XMinorTick','on');
set(h6_plot,'FontName','Arial');
set(h6_plot,'FontSize',14);
set(h6_plot,'box','on');

h6_plot_x_axis_handle = xlabel('	au (min)');
set(h6_plot_x_axis_handle,'FontName','Arial','FontSize',16);

if epsilon_flag == 0
  ord_label = '<r^2> \pm s.e.v. (\mu m^2)';
else
  ord_label = '<r^2> - 4*epsilon^2 \pm s.e.v. (\mu m^2)';
end
h6_plot_y_axis_handle = ylabel(ord_label);
set(h6_plot_y_axis_handle,'FontName','Arial','FontSize',16);

h6_plot_title_handle = title({run_title;'Power Law Model'});
set(h6_plot_title_handle,'FontName','Arial','FontSize',16);

% Save figure window generated:
saveas(h6, 'Fit_PL_sev_loglog_final.fig', 'fig');

%******************
If no warnings generated report so in log file:
if warn == 0
fprintf(1,'\n	Function completed without errors/warnings\n');
fprintf(fid,'\n	Function completed without errors/warnings\n');
end

% Update log file that function is completed:
fprintf(1,'\n%s completed\n',func_name);
fprintf(fid,'\n%s completed\n',func_name);
end

Van_Hove_Analysis_v3.m

1 % Steven J. Henry
2 % 10/16/2013
17 %**************************************************************************
18 % PURPOSE:
19 % This program computes all the displacements observed for a population of
20 % moving objects. Displacements are those between centroid observations of
21 % a given object's trajectory, not between objects. All objects contribute
22 % displacements to the ensemble (population) of displacements observed. The
23 % displacements computed ***have a sign*** and ***are not squared***. The
24 % displacements are overlapping such that all displacements corresponding
25 % to a given centroid's trajectory are not statistically independent.
26 %
27 % ASSUMPTIONS:
28 % 'data' array only possesses data that had previously been utilized in
29 % 'Mean_Displacements.m'. As such this array has been processed via
30 % 'Post_IJ_Manual_Track.m' and possibly 'Filter_Exp_Data.m' such that only
31 % those centroid positions invoked in the previous computation of the MSD
32 % curve sit in the 'data' array presently.
33 %
34 % INPUT:
35 % pixel_calib = user specified objective calibration (um/pixel)
36 % data = an array containing all pertinent tracking information for each
37 % cell in the given experimental condition having the following structure
38 % (as a result of 'Post_IJ_Manual_Track.m'):
39 % col 1 = unique track number ID assigned to each cell
40 % col 2 = x coordinate (pixels) of centroid
41 % col 3 = y coordinate (pixels) of centroid
42 % col 4 = absolute time corresponding to frame in which cell is found
43 % (sec)
44 % col 5 = binned time corresponding to frame in which cell is found (sec)
45 % col 6 = area of cell in pixels
46 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
47 % if no (i.e. continuation of an existing track.
48 %
49 % OUTPUT:
50 % MATLAB figures (.fig)
51 % Van_Hove_dx.fig = van hove analysis of displacements in x direction
52 % Van_Hove_dy.fig = van hove analysis of displacements in y direction
53 % Variance.fig = variance as a function of tau
54 % Kurtosis.fig = kurtosis as a function of tau
55 %**************************************************************************
function [] = Van_Hove_Analysis_v3(pixel_calib, data, MSD_tbin, epsilon_flag, run_title, fid)
%
% Get function name:
func_name = mfilename;
%
% Update log file that function is running:
fprintf(1,'\n%s running ...
',func_name);
fprintf(fid,'\n%s running ...
',func_name);
%
% Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
% warning dialog the log file records no errors/warnings generated:
warn = 0;
%
% Retrieve number of tracks:
uniq_IDs = unique(data(:,1));
num_tracks = length(uniq_IDs);
%
% Reserve the variable name "all_disp" for the column array that will
% log all displacements computed from each track:
all_disp = [];
%
% Loop over each track
for i = 1:num_tracks
%
% Grap the track number
track = uniq_IDs(i);
%
% Find corresponding row numbers for this track's entry in 'data'
%
% array:
track_ind = data(:,1)==track;
%
% Isolate track's x,y centroid positions
track_xy = data(track_ind,2:3);
%
% Isolate bin-time values corresponding to x,y centroid positions
track_t = data(track_ind,5);
%
% What is the total number of centroid observations for this track?
num_obs = length(track_t);
%
% Determine the total number of displacements to be computed using a
% moving origin strategy. Example: if you have 5 observations of an
% object's centroid the total number of intervals you will compute is
% (5-1)+(5-2)+(5-3)+(5-4) = 4+3+2+1 = 10. More generally for N total
% observations of an object's centroid you will have summation(i = 1, i
% = N-1) intervals. Rather than having to evaluate "num_obs-1" with
% every iteration in the following for loop we can move the "-1" into
% the actual computation itself.
num_intervals = 0;
for ii = 2:num_obs
    num_intervals = (ii-1) + num_intervals;
end
%
% if isempty(all_disp)==1
all_disp = zeros(num_intervals,3);
print_row = 1;
else
    all_disp_old = all_disp;
    num_intervals_tot = size(all_disp_old,1);
clear all_disp
    all_disp = zeros(num_intervals_tot+num_intervals,3);
    all_disp(1:num_intervals_tot,:) = all_disp_old;
clear all_disp_old
    print_row = num_intervals_tot+1;
end
for k = 1:num_obs-1
x_orig = track_xy(k,1);
y_orig = track_xy(k,2);
t_orig = track_t(k);

for kk = k+1:num_obs

    % Load advance row
    x_adv = track_xy(kk,1);
y_adv = track_xy(kk,2);
t_adv = track_t(kk);

    % Compute displacements:
dx = x_adv - x_orig;
dy = y_adv - y_orig;
dt = t_adv - t_orig;

    % Log displacements:
    all_disp(print_row,1) = dx;
    all_disp(print_row,2) = dy;
    all_disp(print_row,3) = dt;

    % Advance print row:
    print_row = print_row+1;
end
end
end

% Save 'all_disp' array in .mat and .txt format:
save('all_disp.mat','all_disp');
fid2_1 = fopen('all_disp.txt','wt');
fprintf(fid2_1,'dx (pix)	dy (pix)	dt (sec)
');
rows = size(all_disp,1);
for k = 1:rows
    fprintf(fid2_1,'%f	%f	%f
',all_disp(k,:));
end
fclose(fid2_1);

% **************************
% Begin plotting: (Note this could be rolled into a separate function but
% the following suits the current needs satisfactorily.)
% Convert to physical units:
all_disp(:,1:2) = all_disp(:,1:2)*pixel_calib;
all_disp(:,3) = all_disp(:,3)/60;

taus = unique(all_disp(:,3));
tot_taus = length(taus);

% The following are arbitrary but for the majority of cases identified so
% far (11/03/2011) data is only analyzed through 30 min and imaging is
% conducted at 1 min/frame:
m = 2;
n = 3;
tau_plot = [1 5 10 15 20 25]; %min

% Define figure handles
hist_fig_x = figure;
hist_fig_y = figure;
var_fig = figure;
kurt_fig = figure;

% Reserve memory:
V = zeros(tot_taus,4);
K = zeros(tot_taus,3);

% Generate counter:
counts = 1;

for i = 1:tot_taus
    tau = taus(i);
    ind = all_disp(:,3)==tau;
    x_list = all_disp(ind,1);
    V(i,1) = var(x_list);
    K(i,1) = kurtosis(x_list);
    y_list = all_disp(ind,2);
    V(i,2) = var(y_list);
    K(i,2) = kurtosis(y_list);

    % delR = delX ihat + delY jhat
    % VAR(delR) = VAR(delX)+VAR(delY)+2*COV(delX,delY);
    % We anticipate delX and delY are independent and so COV = 0
    xycov = cov(x_list,y_list);
    % xycov is a 2X2 matrix with:
    % xycov(1,1) = VAR(x_list)
    % xycov(2,2) = VAR(y_list)
    % xycov(1,2) = xycov(2,1) = COV(x_list,y_list);
    V(i,3) = V(i,1)+V(i,2)+2*xycov(1,2);
    V(i,4) = tau;
    K(i,3) = tau;

    % Round to the nearest tenth minute and evaluate to see if it resides
    % in the list of tau_plot values for which a histogram should be
    % constructed, but only do this if you haven't already plotted m*n
    % histograms (the max for this figure).
    if any(round(tau*10)/10==tau_plot)==1 && counts < m*n;
        figure(hist_fig_x)
        subplot(m,n,counts);
        nbins = round(sqrt(length(x_list)));
        [freq bin_loc] = hist(x_list,nbins);
        bar(bin_loc,freq,1);
        x_hist_fig_title = title(['\tau = ' num2str(round(tau)) ' min']);
        set(x_hist_fig_title,'FontName','Arial','FontSize',16);
        x_hist_fig_xlabel = xlabel('\Deltax (\mum)');
        set(x_hist_fig_xlabel,'FontName','Arial','FontSize',16);
        x_hist_fig_ylabel = ylabel('Counts');
        set(gca,'yscale','log');
        set(gca,'FontName');

        figure(hist_fig_y)
        subplot(m,n,counts);
        [freq bin_loc] = hist(y_list,nbins);
        bar(bin_loc,freq,1);
        y_hist_fig_title = title(['\tau = ' num2str(round(tau)) ' min']);
        set(y_hist_fig_title,'FontName','Arial','FontSize',16);
        y_hist_fig_xlabel = xlabel('\Deltay (\mum)');
        set(y_hist_fig_xlabel,'FontName','Arial','FontSize',16);
        y_hist_fig_ylabel = ylabel('Counts');
        set(gca,'yscale','log');
        set(gca,'FontName');
    end
end
set(gca,'FontSize',14);
counts = counts+1;
end
end

% Save Van Hove figures generated:
saveas(hist_fig_x, 'Van_Hove_dx.fig', 'fig');
saveas(hist_fig_y, 'Van_Hove_dy.fig', 'fig');

% Re-plot MSD+/- s.e.v. data for easy visual comparison with variance
% method here:
t_bin = MSD_tbin(:,5)/60; % min
m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
% On log-log axes negative values result in output warnings to user. To
% avoid this filter for msd - s.d. (lower bounds) that result in negative
% values.
L_sev_bin = sev_bin;
for i = 1:length(L_sev_bin)
    if m_bin(i)-L_sev_bin(i) < 0
        L_sev_bin(i) = 0;
    end
end
if epsilon_flag == 0
    legend_label = '<\Deltar^2> \pm s.e.v. (\mum^2)';
elseif epsilon_flag == 1
    legend_label = '<\Deltar^2> - 4*\epsilon^2 \pm s.e.v. (\mum^2)';
end

% Plot variance figure:
figure(var_fig);
errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
hold on
plot(V(:,4),V(:,1),'LineStyle','none','Marker','^','MarkerEdgeColor','k','MarkerFaceColor','b','MarkerSize',5,'HandleVisibility','on');
plot(V(:,4),V(:,2),'LineStyle','none','Marker','s','MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',5,'HandleVisibility','on');
plot(V(:,4),V(:,3),'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
var_fig_xlabel = xlabel('\tau (min)');
set(var_fig_xlabel,'FontName','Arial','FontSize',16);
var_fig_ylabel = ylabel('Squared Displacement (\mum^2)');
set(var_fig_ylabel,'FontName','Arial','FontSize',16);
set(gca,'yscale','log');
set(gca,'xscale','log');
set(gca,'YMinorTick','on');
set(gca,'XMinorTick','on');
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
var_fig_legend = legend(legend_label,'\sigma^2(\Deltax)','\sigma^2(\Deltay)','\sigma^2(\Deltax)+\sigma^2(\Deltay)');
set(var_fig_legend,'FontName','Arial','FontSize',12,'Location','NorthWest');
var_fig_title = title(run_title);
set(var_fig_title,'FontName','Arial','FontSize',16);

% Save variance figure window generated:
saveas(var_fig, 'Variance.fig', 'fig');

% Save 'V' array in .mat and .txt format:
save('Variance.mat','V');
fid2_1 = fopen('Variance.txt','wt');
fprintf(fid2_1,'var(dx) (um^2) var(dy) (um^2) \ tvar(dr) (um^2) \ ttau (min)\n');
rows = size(V,1);
for k = 1:rows
fprintf(fid2_1,'%f,%f,%f,%f
',V(k,:));
end
fclose(fid2_1);

figure(kurt_fig);
plot(K(:,3),K(:,1),'LineStyle','none','Marker','^','MarkerEdgeColor','k','MarkerFaceColor','b','MarkerSize',5,'HandleVisibility','on');
hold on
plot(K(:,3),K(:,2),'LineStyle','none','Marker','s','MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',5,'HandleVisibility','on');
kurt_fig_xlabel = xlabel('\tau (min)');
set(kurt_fig_xlabel,'FontName','Arial','FontSize',16);
kurt_fig_ylabel = ylabel('Kurtosis (unitless)');
set(kurt_fig_ylabel,'FontName','Arial','FontSize',16);
set(gca,'FontName','Arial');
set(gca,'FontSize',14);
set(gca,'box','on');
kurt_fig_legend = legend('\Delta x','\Delta y');
set(kurt_fig_legend,'FontName','Arial','FontSize',12);
kurt_fig_title = title(run_title);
set(kurt_fig_title,'FontName','Arial','FontSize',16);

% Save kurtosis figure window generated:
saveas(kurt_fig, 'Kurtosis.fig', 'fig');

% Save 'kurt' array in .mat and .txt format:
save('Kurtosis.mat','K');
fid2_1 = fopen('Kurtosis.txt','wt');
fprintf(fid2_1,'%f,%f,%f
',K(k,:));
fclose(fid2_1);

% If no warnings generated report so in log file:
if warn == 0
    fprintf(1, '\n Function completed without errors/warnings\n');
    fprintf(fid, '\n Function completed without errors/warnings\n');
end
% Update log file that function is completed:
fprintf1(1, '\n%s completed\n', func_name);
fprintf1(fid1('n%s completed\n', func_name);
end

Tidy_Up_v1.m

Tidy_Up_v1.m

% Steven J. Henry
% 05/01/2011
%**************************************************************************
% PURPOSE:
The following function places all files with .fig, .mat, and .txt extensions into folders called "figs", "mats", and "txts" respectively.
% Only the master log file is left outside these folders for easy navigation.
% REMARKS:
% n/a
function [] = Tidy_Up_v1(logfile)

% Make a folder called 'txts':
mkdir('txts');
% Define the path to that folder:
txt_path = [pwd '\txts'];
% Move all .txt files in the current folder to destination 'txts'
movefile('*.txt',txt_path);
% Change the directory to the 'txts' folder
cd(txt_path);
% Move the master log file out of the 'txts' files and back up one
directory:
movefile(logfile,'#')
% Change directory to original position:
cd('#')
% Make a folder called 'figs'
mkdir('figs');
% Define the path to that folder:
fig_path = [pwd '\figs'];
% Move all .fig files in the current folder to destination 'figs'
movefile('*.fig',fig_path);
% Make a folder called 'mats'
mkdir('mats');
% Define the path to that folder:
mat_path = [pwd '\mats'];
% Move all .mat files in the current folder to destination 'mats'
movefile('*.mat',mat_path);
end
Appendix B

Custom MATLAB Code for Analysis of Neutrophil Spreading

Introduction

The purpose of this appendix is to provide the reader with more detail regarding the data analysis workflow employed to measure neutrophil spreading statistics. Broadly speaking the workflow consisted of capturing timelapse images of neutrophil spreading on mPADs, identifying the posts in each image and computing geometric centroids, linking centroids into trajectories, positioning post trajectories relative to their resting lattice (undeflected) positions, and computing ensemble statistics. The appended code is original and custom built to interface with Maria Kilfoil’s MATLAB version (1) of John Crocker and David Grier’s particle tracking routines originally developed in the IDL programming language (2). When a Kilfoil function is called in the following code I direct the reader to consult her well-annotated user manual online (3). I only reproduce my custom MATLAB codes which were used to interface with the Kilfoil scripts and to perform post hoc ensemble averaging and statistical analysis.

A significant functionality of my code is to allow reconstruction of the resting mPADs lattice conformation from an image of the deflected lattice via a two dimensional Fourier filtering method (Fig. B.1). This functionality allows the user to work with data sets in which cell spreading and post engagement commenced prior to the start of data acquisition and therefore requires that a post trajectory be positioned relative to its resting lattice position which was not the trajectory position at the onset of imaging. The
Figure B.1 Resting lattice bitmap reconstruction. This is the workflow associated with the reconstruction of a resting lattice bitmap from a perturbed bitmap. It consists of a two dimensional Fourier transform on the perturbed lattice bitmap followed by filtering on the histogram of complex moduli to retain the signal from the resting lattice. When a suitable threshold is empirically determined the inverse two dimensional Fourier transform is computed. Feature finding is performed to centroid the resting lattice positions and extraneous noise is excluded. Finally a nearest neighbours search is executed to position perturbed post positions relative to their resting lattice locations.
positioning of post trajectories relative to the resting lattice position is required to compute the deflection strain and therefore back calculate the force required to generate the deflection.

**Methodology**

1. Prepare data according to the nomenclature required by Kilfoil. A parent directory should contain a sequence of .tif images in a folder “fov#”. Images are labeled “fov##_####.tif”. For example “fov9” would refer to a folder corresponding to field of view 9 and “fov9_0037.tif” would be the filename of frame 37 at location 9. A time vector called “time” must be the same length as the number of images in the “fov#” folder and reside in the same parent directory as the “fov#” folder. The time vector is has a filename “fov##_times.mat”. Multiple “fov#” folders and “fov##_times.mat” vectors can reside within the same directory and can be processed simultaneously.

2. In MATLAB, run Kilfoil’s “mpretrack_init.m” to initialize the appropriate tracking parameters for the experimental data set such as particle shape, intensity, and size.

3. Using the tracking parameters established from “mpretrack_init.m” run Kilfoil’s “mpretrack.m”.

4. The output of “mpretrack.m” is a folder in the parent directory called “Feature_finding” containing a file called “MT_##_Feat_Size_#.mat”. For example “MT_9_Feat_Size_3.mat” would be the data array of all features of radius 3 pixels located in “fov9” images. The data array is the input of “fancytrack.m”.

5. In MATLAB, run Kilfoil’s “fancytrack.m”.

6. The output of “fancytrack.m” is a folder in the parent directory called “Bead_tracking” which contains a subfolder called “res_files.” This subfolder
contains a matrix called “res_fov#.mat” which holds the particle connectivities (i.e. trajectories) in successive frames.

7. In MATLAB, run “Post_Analysis_Driver_v9_thesis.m”

a. You will need to specify the following parameters:

   i. Line 134: “exp_date” – 8 digit date of experiment (yyyyymmdd)

   ii. Line 137: “exp_donor” – donor ID string

   iii. Line 140: “exp_cond” – string description of experimental condition

   iv. Line 143: “fovn” – numeric field of view number

   v. Line 146: “time_int” – time interval between frames in seconds

   vi. Line 149: “max_num_frames” – maximum number of frames for fovn

   vii. Line 152: “microntopix” – micron to pixel conversion in pixels/μm

   viii. Line 155: “kspring” – post spring constant in pN/nm

   ix. Line 158: “basepath” – string path to parent directory containing fovn data

   x. Line 166: “dilate” – numeric integer in pixels by which objects will be dialted to assist with visualizing post lattices

b. “Post_Analysis_Driver_v9_thesis.m” calls the following subroutines:

   i. “SelectBackgroundPosts_v3.m” – You will be asked if you want to dedrift the data. You should always elect to dedrift or errors will be thrown because subsequent functions require the drift model. Select “New Model” and navigate to an image of the contractile cell. Draw a polygon around posts not beneath the cell. Double click when the polygon is closed to continue processing. The function will output a figure of the drift model in microns.

   ii. “dedrifting_and_conversions_v3.m” (SJH adaptation of Kilfoil code)
1. “pixtomicro.m” (Kilfoil code) - The Kilfoil code does not receive the “microntopix” variable so you must adjust the hardcoded value manually.

2. “drift_loop_makedriftfrombkgr.m” (SJH adaptation of Kilfoil code)
   a. From_8_columns_to_4 (Kilfoil code)
   b. Motion.m (Kilfoil code)

3. “drift_loop_dedriftalldata.m” (SJH adaptation of Kilfoil code)
   a. “dedrift.m (Kilfoil code)
   b. “putting_in_missing_frames.m” (Kilfoil code)

   iii. “conversion_no_dd.m” (Kilfoil code)

   iv. “getting_individual_beads.m” (Kilfoil code)

   v. “Time_Avg_Centroids_v4.m” – This function will output plots of the time average centroid positions of each trajectory superimposed on the trajectories themselves. You should inspect these plots to see if trajectories must be merged.

   vi. “DedriftImageForOverlays_v1.m” – Navigate to a .tif image that will be used for subsequent superimposition of trajectories and post IDs.

   vii. “OverlayPostIDs_v5.m”

   viii. “Merge_Trajectories_v1.m” – If the plots from “Time_Avg_Centroids_v4.m” reveal trajectories that require merging specify which tracks should be joined. Repeat as necessary.

   ix. “PlotPostIDs_v1.m”

   x. “SetSuperResolutionRefinement_v1.m” – Set refinement value, this is an integer multiple by which you will increase the number of pixels.
xi. “Construct_Bitmap_v4.m” -

xii. “Find_Rotation_Angle_v1.m” – Specify the rotation angle or use the interactive feature to determine the angle.

xiii. “Apply_Rotation_v3.m”

xiv. “Crop_Bitmap_v2.m” – A user interface requires you to draw a rectangular ROI that achieves periodic boundary conditions. Double click inside ROI when finished to proceed with analysis.

xv. “FilterForRestingLattice_v6.m” – An iterative process is run to determine a threshold on the histogram of complex moduli from a 2D Fourier transform of the perturbed lattice bitmap. The program pauses and allows you to probe the threshold value (“thres”) and manually change this value. You can assess the result of changing this value by executing lines 188-190, 204, 208-211, in the command window. The goal is to achieve a threshold setting (usually a value that retains the upper flat tail of the complex moduli histogram) so that the inverse 2D Fourier transform returns a resting lattice bitmap with bright and distinct features. When finished enter “return” in the command window.

xvi. “UndoCrop_v1.m”


1. “mpretrack_init.m” (Kilfoil code)

xviii. “TrajectoriesRelativeToRestingLattice_v9.m”

xix. “TrajectoriesInCellReferenceFrame_v5.m”- Double click on approximate geometric centroid of cell.
xx. “IdentifyEngagedPosts_v4.m” – Draw a polygonal ROI around the diffuse cloud of high variance data points, excluding the tight cloud of low variance data points.

xxi. “ReviseCellRefTrajectories_v1.m”

xxii. “GeoSortEngagedPosts_v2.m” – Set the nearest neighbor distance (“<knn>”) in pixels to distinguish perimeter posts from core posts. This is the value that differentiates the first plateau of nearest neighbor distances from the second plateau.

xxiii. “RepopulatePostIDLLists_v2.m”

xxiv. “PlotCellRefTrajectories_v7.m”

xxv. “PlotMetricsVsRadialDist_v7.m”

xxvi. “Plot_fvst_Strips_v1.m”

xxvii. “IndividualPostAutoCorrelation_v2.m”

xxviii. “Tidy_Up_v2.m”

8. The output from running “Post_Analysis_Driver_v9_thesis.m” is a “Post_Analysis_Driver_v9_thesis_fovn” folder. The numeric prefix to the folder is the ISO 8601 dateform (“yyyymmddTHHMMSS”) on which the analysis was performed. The two subfolders of particular interest include “fig” which contains copies of all figures generated during the run and “lsx” which contains an Excel worksheet (MeanMetrics.xlsx) that summarizes the ensemble statistics computed during the run. A session workspace “SessionWorkspace.mat” is also saved which retains all the variables and arrays generated during the run.
Areas for Code Improvement

One area for improvement in the existing code is the localization of post trajectories relative to their resting lattice positions. Presently a simple nearest neighbors approach is employed. While this is satisfactory for small initial displacements relative to the resting lattice position when displacements are large the nearest neighbors approach leads to misidentification. A stronger strategy is to perform a seek analogous to an energy minimization scheme permuting all combinations of posts and resting lattice positions until the sum of the squared distances is minimized. This is the approach utilized in Crocker and Grier’s particle tracking algorithm as well as Kilfoil’s version of their routines. At a high level this improvement would involve a simple substitution of my “TrajectoriesRelativeToRestingLattice_v9.m” with Kilfoil’s “fancytrack.m” with the necessary nomenclature and variable changes being satisfied.

An additional area for improvement is to move away from fixed data arrays and towards an object oriented programming scheme. In the former case great care must be taken to maintain array dimensionality and organization especially when passing arrays between functions. In the later case data is assigned properties and these properties can be used to retrieve data or aspects of data without as much overhead.

Code

Note: missing lines are version history annotation, removed for space considerations.

Post_Analysis_Driver_v9_thesis.m

1    % Steven J. Henry
2    % 02/17/2015
58    %**************************************************************************
60    % PURPOSE:
61    % This driver calls a sequence of functions to dedrift post trajectories,
62    % locate post trajectories relative to their ideal resting lattice
63    % position, parses engaged from non-engaged posts, and computes associated
64    % statistics.
% ASSUMPTIONS:
% n/a

% INPUT:
% exp_date = 8 digit number date of experiment (yyyyymmdd)
% exp_donor = donor ID string ('DXX')
% exp_cond = string describing experimental condition (e.g. 'Control')
% fovn = field of view number
% time_int = time interval between frames in seconds
% max_num_frames = maximum number of frames for fovn
% microntopix = microntopix conversion in pixels/um
% kspring = post spring constant in pN/um
% basepath = path to parent directory containing fovn data
% dilate = radius in pixels by which a single "on" pixel should be dilated
% to assisted with visual inspection of lattice

% OUTPUT:
% A host of intermediate and final ensemble analysis metrics and figures.

% DRIVER/FUNCTION MAP:

%**************************************************************************
clc;
clear all;
close all;
tic;

%**** HARD CODED PARAMETERS START ***************************************
% Set experiment data in YYYYMMDD format:
exp_date = #;

% Set experimental donor in 'DXX' format:
exp_donor = 'DXX';

% Set experimental condition:
exp_cond = 'EnterConditionHere';

% Set fovn:
fovn = #;

% Set time between frames:
time_int = #; % (s/frame)

% Set maximum number of frames
max_num_frames = #;

% Have user specify the microntopix conversion:
microntopix = #; % pixels/um

% Set post spring constant
kspring = #; % pN/nm

% Set basepath to experimental folder
basepath = uigetdir(...
    'EnterPathToDataHere','...
    'Where does the data reside?');
basepath = [basepath ''];

% Radius (pix) by which a single "on" pixel should be dilated using a
% 'disk' structuring element to assist with visual inspection of lattice
% manipulations
dilate = #;

%**** HARD CODED PARAMETERS END *******************************************

% Have user specify where to save data
save_here = uigetdir(basepath,'Where should analysis be saved?');

% Determine date and time. Create a string in "dateform" "30" (ISO 8601)
% which has the format 'yyyyymmddTHHMSS'.
dstr = datestr(now, 30);

% Create a folder to hold results of analysis:
func_name = mfilename;
results_folder_name = [dstr '_' func_name];
results_folder_path = [save_here '\results_folder_name' '_'fovn ...
    num2str(fovn)];
mkdir(results_folder_path);

% Set directory to analysis folder:
results_folder_path = results_folder_path;

% Start a log file. Save in new directory:
logfile = [results_folder_name '__Log.txt'];
fid = fopen(logfile,'wt');

% Update log file that function is running:
fprintf(fid,'Analysis commenced: %s
',dstr);

fprintf(fid,'
Results folder path:
%s
',results_folder_path);

fprintf(fid,'
%s running ...
',func_name);
% Record hard-coded values:
fprintf(fid,'\nHard-coded values for this run:\n');
fprintf(fid,'Hard-coded values for this run:\n');
fprintf(fid,'\n\texp_date = %s \n',num2str(exp_date));
fprintf(fid,'\texp_donor = %s \n',num2str(exp_donor));
fprintf(fid,'\texp_cond = %s \n',num2str(exp_cond));
fprintf(fid,'\tmicrontopix = %s um/pix\n',num2str(microntopix));
fprintf(fid,'\mtime_int = %s s/frame\n',num2str(time_int));
fprintf(fid,'\tkspring = %s pN/nm\n',num2str(kspring));
fprintf(fid,'\tmax_num_frames = %s \n',num2str(max_num_frames));
fprintf(fid,'\tdilate = %s pix (radius)\n',num2str(dilate));
fprintf(fid,'\n\nend record hard-coded values
');

% Do you want to dedrift the data?
% If you elect not to dedrift you will produce errors later in functions
% that depend upon the output of dedrifting, namely the 'driftmodel_um' and
% 'Idedrift' variables.
dedrift_prechoice = menu('Dedrift the data?','Yes','No');
if dedrift_prechoice == 1
    % Dedrift data. It's critical that smoo = 0 or correlation will be
    % introduced in the data.
    dedrift_choice = menu('Dedrift with:','Existing model','New model');
    if dedrift_choice == 1
        [drift_name,drift_path] = uigetfile([basepath '*.mat'],...
            'Select existing drift model in microns');
        load([drift_path drift_name]);
        [bkgr_name, bkgr_path] = uigetfile([basepath '*.mat'],...
            'Select existing list of background post IDs');
        load([bkgr_path bkgr_name]);
        [driftmodel_um] = dedrifting_and_conversions_v3(...
            basepath,fovn,driftmodel_um);
    elseif dedrift_choice == 2
        [bkgr_post_ids] = SelectBackgroundPosts_v3(...
            basepath,fovn,results_folder_path,fid);
        [driftmodel_um] = dedrifting_and_conversions_v3(basepath,fovn,[]);
    end
    save([results_folder_path filesep 'driftmodel_um.mat'],...
        'driftmodel_um');
    save([results_folder_path filesep 'bkgr_post_ids.mat'],...
        'bkgr_post_ids');
else dedrift_prechoice == 2
    conversions_no_dd( basepath, fovn);
end
% Construct individual bead trajectories
getting_individual_beads(basepath,fovn);

% Move 'ddposum_files' folder within results_folder_path:
source = [basepath 'Bead_tracking' filesep 'ddposum_files'];
destination = [results_folder_path filesep 'ddposum_files'];
movefile(source,destination);

% Copy 'individual_beads' and rename to 'beads'
source = [results_folder_path filesep 'ddposum_files' filesep ...
    'individual_beads'];
bead_path = [results_folder_path filesep 'beads'];
copyfile(source,bead_path);

% Select bead.mat files to process and compute time-average centroids
{TAC,FOC} = Time_Avg_Centroids_v4(bead_path,microntopix,...
    results_folder_path,fid);

% Select an image that will be used for subsequent overlays and to
determine the cell geometric centroid. This image will require
"dedrifting" so future overlays properly align.
[Idedrift,raw_dim] = DedriftImageForOverlays_v1(...
    basepath,fovn,microntopix,driftmodel_um,results_folder_path,fid);

% Output an image with a post-ID overlay:
cat = 'allposts';
color = [0,0,0.8];
OverlayPostIDs_v5(bead_path,Idedrift,cat,color,microntopix,...
    results_folder_path,fid);

% Ask user if he wants to merge trajectories belonging to the same post:
merge_choice = menu('Merge trajectories?','Yes','No');

if merge_choice == 1

    mkdir(bead_path,'premerge_beads');
copyfile(bead_path,[bead_path filesep 'premerge_beads']);
extflag = 1;
while exitflag ~= 2

        Merge_Trajectories_v1(bead_path,fid);

        PlotPostIDs_v1(bead_path,microntopix,results_folder_path,fid);

        exitflag = menu('Merge again?','Merge again','Finished merging');

end

end

% Select bead.mat files to process and compute time-average centroids
{TAC,FOC} = Time_Avg_Centroids_v4(bead_path,microntopix,...
    results_folder_path,fid);

% Output an image with a post-ID overlay:
cat = 'allposts_aftermerge';
OverlayPostIDs_v5(bead_path,Idedrift,cat,color,microntopix,...
    results_folder_path,fid);

end

close('all');

% Have user specify how much to super-sample the existing bitmap:
[refine] = SetSuperResolutionRefinement_v1(results_folder_path,fid);

% Construct a binary bitmap using time average centroids.

% Plot bitmap:
fig_handle_Ap_dilated = figure;
imshow(imdilate(Ap,strel('disk',dilate)),'DisplayRange',[0]);
title_string = [num2str(dilate) ' X dilated bitmap of time averaged centroids positioned to ' num2str(1/refine) ' pix'];
title(title_string,'FontSize',20,'Interpreter','None');
saveas(fig_handle_Ap_dilated, [results_folder_path '\Bitmap_Perturbed_Dilated.fig'], 'fig');
save([results_folder_path filesep 'Bitmap_Perturbed.mat'],'Ap');
save([results_folder_path filesep 'Bitmap_Perturbed_LUT.mat'],'Ap_LUT');

% Rotate binary bitmap containing time average centroids so one row is (approximately) parallel with vertical axis.
rotate_choice = menu('Rotate by:','Entering rotation','Finding rotation');
if rotate_choice == 1
    theta = input('Set theta (rotation angle and sign) = ');
elseif rotate_choice == 2
    [theta] = Find_Rotation_Angle_v1(Ap,results_folder_path,fid);
end
save([results_folder_path filesep 'theta.mat'],'theta');

[PXr,PYr] = Apply_Rotation_v3(TAC(:,2),TAC(:,3),theta,...
    results_folder_path,fid,raw_dim);
    PXr,PYr),raw_dim,refine,fid);

% Plot bitmap:
fig_handle_Ap_rotated_dilated = figure;
imshow(imdilate(Ap_rotated,strel('disk',dilate)),'DisplayRange',[0]);
title_string = [num2str(dilate) ' X dilated bitmap of rotated time averaged centroids positioned to ' num2str(1/refine) ' pix'];
title(title_string,'FontSize',20,'Interpreter','None');
saveas(fig_handle_Ap_rotated_dilated, [results_folder_path '\Bitmap_Perturbed_Rotated_Dilated.fig'], 'fig');
save([results_folder_path filesep 'Bitmap_Perturbed_Rotated.mat'],'Ap_rotated');
save([results_folder_path filesep 'Bitmap_Perturbed_Rotated_LUT.mat'],'Ap_rotated_LUT');

% Crop rotated bitmap so boundaries are (approximately) periodic:
    results_folder_path,fid);

% Filter for resting lattice
pts = 5;
method = 'auto';
[Ar] = FilterForRestingLattice_v6(Ap_cropped,pts,method,...
    results_folder_path,fid);

% Undo crop:
[Ar_uncropped] = UndoCrop_v1(Ar,pad,dilate,results_folder_path,fid);

% Save Ar_uncropped in format necessary for Kilfoil feature finding:
mkdir([results_folder_path '\fov0']);
imwrite(Ar_uncropped,[results_folder_path '\fov0\fov0_0000.tif']);
time = 0;
save([results_folder_path '\fov0\fov0_times.mat'],'time');

% Perform initialize.m to set necessary parameters for Kilfoil feature
% finding:
[M2,MT,KilfoilRestingLatticeParameters] = KilfoilInitialize_v3(0,0,...
    results_folder_path,fid);
dim = size(Ar_uncropped);
[RX,RY] = Apply_Rotation_v3(MT(:,1),MT(:,2),theta,...
    results_folder_path,fid,dim);
% Step down resolution:
RX = RX/refine;
RY = RY/refine;

[Ar,Ar_LUT] = Construct_Bitmap_v4(horzcat((1:size(MT(:,1),1))',RX,RY),...
raw_dim,refine,fid);

% Plot bitmap:
fig_handle_Ar_dilated = figure;
imshow(imdilate(Ar,strel('disk',dilate)),'DisplayRange',[]);
title_string = ['X dilated bitmap of resting lattice centroids positioned to ','
num2str(1/refine),' pix'];
title(title_string,'FontSize',20,'Interpreter','None');

%Save figure and array:
saveas(fig_handle_Ar_dilated, [results_folder_path ...
'Bitmap_Resting_Dilated.fig'], 'fig');
save([results_folder_path filesep 'Bitmap_Resting.mat'], 'Ar');
save([results_folder_path filesep 'Bitmap_Resting_LUT.mat'], 'Ar_LUT');

imwrite(imdilate(Ap,strel('disk',dilate)),'Ap_tavg_dil.tif');
imwrite(imdilate(Apo,strel('disk',dilate)),'Ap_frame1_dil.tif');
imwrite(imdilate(Ar,strel('disk',dilate)),'Ar_dil.tif');
imwrite(Ap,'Ap_tavg.tif');
imwrite(Apo,'Ap_frame1.tif');
imwrite(Ar,'Ar.tif');

close('all');

% Update bead.mat files so all trajectories are relative to their resting
% lattice position:
[shift_Ap,sbead_path] = TrajectoriesRelativeToRestingLattice_v9(...
Ar,Apo_LUT,microntopix,refine,bead_path,results_folder_path,fid);

% Translate trajectories into the cell reference frame
[manual_centroid,rsbead_manualcentroid_path]...
= TrajectoriesInCellReferenceFrame_v5(sbead_path,microntopix,...
Idedrift,results_folder_path,fid);

% Sort on post trajectories to distinguish cell-engaged from non-engaged
% posts:
[engaged_post_IDs,ens_manualcentroid_rsbead_path]...
= IdentifyEngagedPosts_v4(bead_path,rsbead_manualcentroid_path,...
microntopix,Idedrift,results_folder_path,fid);

% Pause to make manual adjustments?
pause_choice = menu('Make manual changes?','Yes','No');
if pause_choice == 1
keyboard;
end

% Revise the cell reference frame trajectories using the true geometric
% centroid of the ensemble of engaged posts:
[centroid,rsbead_path,ens_rsbead_path] = ReviseCellRefTrajectories_v1(...
    sbead_path,engaged_post_IDs,microntopix,results_folder_path,fid);

% Perform geometric sort of engaged posts to bin those that reside at the
cell edge vs. those that reside at the core:
[core_post_IDs, core_rsbead_path, perim_post_IDs, perim_rsbead_path]... = GeoSortEngagedPosts_v2(sbead_path,rsbead_path,microntopix,...
    engaged_post_IDs,Idedrift,results_folder_path,fid);

% Pause to make manual adjustments?
pause_choice = menu('Make manual changes?','Yes','No');
if pause_choice == 1
    keyboard;
    % If manual adjustments have been made you need to repopulate the list
    % of post_IDS belonging to each category
    [engaged_post_IDs, core_post_IDs, perim_post_IDs]... = RepopulatePostIDLists_v2(ens_rsbead_path,core_rsbead_path,...
        perim_rsbead_path,results_folder_path,fid);
end

cat = 'ens';
color = [0,0,0.8];
OverlayPostIDs_v5(bead_path,Idedrift,cat,color,microntopix,...
    results_folder_path,fid,engaged_post_IDs);

transition_time = input(
    ['Set ' cat ...
        ' time (s) after Fmax to consider system "steady state" = ']);
fprintf(1,...
    'nFor %s category user said steady state is %s (s) post peakin',
    cat,num2str(round(transition_time)));

[ens_mpara,ens_sdpara,ens_cpara,ens_separa,...
    ens_mperp,ens_sdperp,ens_cperp,ens_seperp,...
    ens_Fmax,ens_Fss]... = PlotCellRefTrajectories_v7(ens_rsbead_path,time_int,...
    kspring,cat,color,results_folder_path,fid,transition_time);

[ens_radial_metrics] = PlotMetricsVsRadialDist_v7(...
    exp_date,exp_donor,exp_cond,fovn,...
    sbead_path,centroid,microntopix,...
    transition_time,cat,color,results_folder_path,fid,...
    ens_Fmax,ens_Fss,ens_t,ens_mpara_F);

Plot_fvst_Strips_v1(rsbead_path,ens_radial_metrics,kspring,...
    time_int,cat,color,results_folder_path,fid);

IndividualPostAutocorrelation_v2(rsbead_path,engaged_post_IDs,...
    max_num_frames,time_int,cat,color,results_folder_path,fid);

cat = 'core';
color = [0.17, 0.51, 0.34];
OverlayPostIDs_v5(bead_path,Idedrift,cat,color,microntopix,...
    results_folder_path,fid,core_post_IDs);

choice = menu("Set new transition time?",'Yes',...
    ['Use current value (' num2str(transition_time) 's)']);
if choice == 1
    transition_time = input(
        ['Set ' cat ...
            ' time (s) after Fmax to consider system "steady state" = ']);
end

fprintf(1,...
"nFor %s category user said steady state is %s (s) post peakin'....
cat num2str(round(transition_time));
fprintf(fid,...."nFor %s category user said steady state is %s (s) post peakin'....
cat,num2str(round(transition_time));
    [core_mpara,core_sdpara,core_cpara,core_separa,...
core_mperp,core_sdperp,core_cperp,core_seperp,...
core_mpara_F,core_sdpara_F,core_cpara_F,core_separa_F,...
core_mperp_F,core_sdperp_F,core_cperp_F,core_seperp_F,...
core_t,core_Fmax,core_Fss] = PlotCellRefTrajectories_v7(core_rsbead_path,time_int,...
spring,cat,color,results_folder_path,fid,transition_time);

    [core_radial_metrics] = PlotMetricsVsRadialDist_v7(...
e_date,exp_donor,exp_cond,fovn,...
sbead_path,centroid,microntopix,...
transition_time,cat,color,results_folder_path,fid,...
core_Fmax,core_Fss,core_t,core_mpara_F);
    Plot_fvst_Strips_v1(rsbead_path,core_radial_metrics,kspring,...
time_int,cat,color,results_folder_path,fid);

    IndividualPostAutocorrelation_v2(rsbead_path,core_post_IDs,...
max_num_frames,time_int,cat,color,results_folder_path,fid);
    Plot_fvst_Strips_v1(rsbead_path,core_post_IDs,kspring,...
time_int,cat,color,results_folder_path,fid);
    IndividualPostAutocorrelation_v2(rsbead_path,core_post_IDs,...
max_num_frames,time_int,cat,color,results_folder_path,fid);

    % Start Perim analysis
    cat = 'perim';
    color = 'r';
    OverlayPostIDs_v5(bead_path,Idedrift,cat,color,microntopix,...
results_folder_path,fid,perim_post_IDs);
    choice = menu('Set new transition time?','Yes',...
'Use current value (num2str(transition_time) s)');
    if choice == 1
        transition_time = input(['Set ' cat...
        ' time (s) after Fmax to consider system "steady state" = ']);
    end
    fprintf(fid,...."nFor %s category user said steady state is %s (s) post peakin'....
cat,num2str(round(transition_time));
    [perim_mpara,perim_sdpara,perim_cpara,perim_separa,...
perim_mperp,perim_sdperp,perim_cperp,perim_seperp,...
perim_mpara_F,perim_sdpara_F,perim_cpara_F,perim_separa_F,...
perim_mperp_F,perim_sdperp_F,perim_cperp_F,perim_seperp_F,...
perim_t,perim_Fmax,perim_Fss] = PlotCellRefTrajectories_v7(perim_rsbead_path,time_int,...
spring,cat,color,results_folder_path,fid,transition_time);
    Plot_fvst_Strips_v1(rsbead_path,perim_radial_metrics,kspring,...
time_int,cat,color,results_folder_path,fid);
    IndividualPostAutocorrelation_v2(rsbead_path,perim_post_IDs,...
max_num_frames,time_int,cat,color,results_folder_path,fid);
    % Update log file that function is completed:
elapsed = toc/60; % minutes, default is seconds
% Sort all files generated into folders of .fig, .mats, and .txt files
% leaving the master log file residing outside the three folders:
cd(results_folder_path);
Tidy_Up_v2(logfile);
save([results_folder_path filesep 'SessionWorkspace.mat']);

%**************************************************************************
% PURPOSE:
% This function establishes the set of posts in a FOV that are known
% background posts for use in constructing a drift model.
% %
% % ASSUMPTIONS:
% % n/a
% %
% % INPUT:
% basepath = path to parent directory containing fovn data
% fovn = field of view number
% save_path = path to folder holding analysis results
% fid = log file identifier
%%
% % OUTPUT:
% bkgr_pos_ids = a vector of post IDs designated as "background" post
% because they reside within user's drawn polygon ROI.
% %
% % DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)
% %**************************************************************************

function [bkgr_post_ids] = SelectBackgroundPosts_v3(basepath,fovn,...
        save_path,fid)

    % Get function name:
    func_name = mfilename;

    % Update log file that function is running:
    fprintf(1,'\n%\ns running ...
',func_name);
    fprintf(fid,'\n%\ns running ...
',func_name);

    % Open a frame and select region not containing cell
    [img_FileName,img_PathName] = uigetfile([basepath '*.*'],...
        'Select .tif image to outline background posts','MultiSelect','off');

    % Outline region containing background posts, excluding cell(s)
    imshg([img_PathName img_FileName]);
    title('Outline region containing background posts, excluding cell(s)');

    % Outline region containing background posts, excluding cell(s)n';
    h = impoly;
    verts = wait(h);
% Make a copy of the figure for superimposing trajectories:
h1=gcf;
fig_handle_impoly2 = figure;
objects=allchild(h1);
copyobj(get(h1,'children'),fig_handle_impoly2);
colormap(gray);

% Open res file that corresponds to this iamge:
% Open a frame and select region containing cell
[res_FileName,res_PathName] = uigetfile([basepath '*.mat'],
    'Select res.mat file corresponding to image.','MultiSelect','off');
load([res_PathName,res_FileName]);

% Identify the bead ID #s in res.mat that belong to these background posts
% and create a bkgr_res.mat file:
post_list = unique(res(:,8)); %#ok<NODEF>
num_posts = length(post_list);
bkgr_keep_list = false(size(res,1),1);
for i = 1:num_posts
    fprintf(fid,'
    Processing post %s of %s
',
        num2str(i),num2str(num_posts));
    end
post = post_list(i);
post_id = res(:,8)==post;
x = res(post_id,1);
y = res(post_id,2);
IN = inpolygon(x(1),y(1),verts(:,1),verts(:,2));
if IN == 1
    bkgr_keep_list(post_id) = true;
end
end

bkgr_post_ids = unique(res_bkgr(:,8));

if exist('fig_handle_impoly','var')==1
    saveas(fig_handle_impoly,[save_path filesep img_FileName(1:end-4) ...
    '_user_impoly.fig'])
end

end

% Update log file that function is completed:
fprintf(fid,'%s completed
',func_name);
end

%**************************************************************************
% PURPOSE:
% This is a modification of Kilfoil's *dedrifting_and_conversions.m* to
% establish the drift model (center of mass motion) of the ensemble of
% background (non-cell engaged) posts from all post trajectories.
%
% ASSUMPTIONS:
% n/a
% INPUT:
% basepath = path to parent directory containing fovn data

dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil code)
% take = field of view number
% save_path = path to folder holding analysis results
% driftmodel_um = received as an empty vector []

% OUTPUT:
% dr_um = drift model in microns of center of mass motion of ensemble of
% particles (posts)

% DRIVER/FUNCTION MAP:
% 0     dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil)
% 1     pixtomicro.m (Kilfoil)
% 1     drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil)
% 1     drift_loop_dedriftalldata (SJH adaptation of Kilfoil)
%**************************************************************************

function[dr_um] = dedrifting_and_conversions_v3(basepath,take,...
    driftmodel_um)

    pathout = ([basepath 'Bead_tracking\ddposum_files\']);
    [status,message,messageid] = mkdir( pathout );

    if isempty(driftmodel_um)==1
        % Load res_bkgr.mat
        load( [basepath 'Bead_Tracking\res_files\res_fov' num2str(take) ...
            '_bkgr.mat'] );

        % Convert to microns from pixels
        res_bkgr_um = pixtomicro(res_bkgr);

        % Create drift vectors in microns
        dr_um = drift_loop_makedriftfrombkgr(res_bkgr_um);

        print('-dpng','-r150', [pathout 'dedrift_run' num2str(take) '.png'])
    else
        dr_um = driftmodel_um;
    end

    % Load res.mat
    load( [basepath 'Bead_Tracking\res_files\res_fov' num2str(take) '.mat'] );

    % Convert to microns from pixels
    res_um = pixtomicro(res);

    % Apply drift model to all posts in res
    ddposum = drift_loop_dedriftalldata(res_um,dr_um);

    % Save dedrift data
    save( [pathout 'ddposum_run' num2str(take) '.mat'], 'ddposum' );
end

function[dr_bkgr] = drift_loop_makedriftfrombkgr(res_bkgr_um)

    bkg = mean(res_bkgr_um); %create a drift vector.
    bkg = bkg(1,1,1); % extract single value from dimensions.

    % Apply drift to res
    ddposum = drift_loop_dedriftalldata(res_bkgr_um,bkg);
end

function[ddposum] = drift_loop_dedriftalldata(res,dr_bkgr)

    df = zeros (size(res,1),size(res,2),size(res,3));

    % correct to microns
    res = pixtomicro(res);

    % apply drift to all posts
    for k = 1:size(res,3)
        df(:,:,k) = res(:,:,k) - dr_bkgr;
    end

    % save
    ddposum = df;
end

%**************************************************************************

% dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil code)
% Steven J. Henry
% 02/17/2015
%**************************************************************************

% PURPOSE:
% This is a modification of Kilfoil's "drift_loop.m" to establish drift
% model of background (non-cell engaged) posts only.
% % ASSUMPTIONS:
% n/a
% % INPUT:
% res_bkgr = res file from track.m that only contains user-specified
% background posts. Filter was done prior to passing to this function.
% OUTPUT:
% dr_um = drift model in microns of center of mass motion of ensemble of
% particles (posts)
%
% DRIVER/FUNCTION MAP:
% 0         drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil)
% 1         from_8_columns_to_4.m (Kilfoil)
% 1         motion.m (Kilfoil)
%**************************************************************************

function[dr_um] = drift_loop_makedriftfrombkgr(res_bkgr)
% This part makes a matrix of 4 columns that is more convenient.
poss = from_8_columns_to_4(res_bkgr);

% Finds the center of mass motion for each frame.
drift=motion(poss,[1 0],2);

function
%**************************************************************************

figure
plot(t, dr)
xlabel('frame','fontsize',16)
ylabel('drift (blue x, green y) (mum)','fontsize',16)
set(gca,'fontsize',16,...
    'TickLength',[0.025 0.06]);

% Pass out a variable called dr_um = drift in microns SJH 03/06/2014
dr_um = dr;

% Steven J. Henry
% 02/17/2015
%**************************************************************************

% PURPOSE:
% This is a modification of Kilfoil's "drift_loop.m" to perform subtraction
% of drift model established from background (non-cell engaged) posts to all
% posts.
%**************************************************************************

function[rdf2] = drift_loop_dedriftalldata(res,dr)
poss = from_8_columns_to_4(res);

%**************************************************************************

function[rdf2] = drift_loop_dedriftalldata(res,dr)
% This function asks the user to select bead/particle/post trajectories
% .mat files and computes the time average centroid of each post in lab
% reference frame.
% ASSUMPTIONS:
% INPUT:
% basepath = string pointing to location of experimental data and analysis
% microntopix = objective calibration for conversion of microns to pixels
% save_path = string pointing to folder containing analysis results
% fid = file ID of log file to which progress is recorded
% OUTPUT:
% TAC = three column array with total number of rows equal to the number of
% user specified trajectory .mat files
% col 1 = post number (an integer value)
% col 2 = time average x-coordinate of centroid (pix)
% col 3 = time average y-coordinate of centroid (pix)
% FOC = three column array with total number of rows equal to the number of
% user specified trajectory .mat files
% col 1 = post number (an integer value)
% col 2 = frame 1 x-coordinate of centroid (pix)
% col 3 = frame 1 y-coordinate of centroid (pix)
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

function [TAC,FOC] = Time_Avg_Centroids_v4(bead_path,microntopix,...
    save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%n%s running ...
',func_name);
fprintf(fid,'%n%s running ...
',func_name);

% Retrieve all contents that reside inside 'bead_path' folder:
contents = dir(bead_path);

% size returns number of rows of a matrix:
num_items = size(contents,1);

% cell returns a cell array:
FileName = cell(num_items,1);

% NaN stands for Not a Number:
ID = NaN(num_items,1);

% false is returned when keep_ind is not true:
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 10 & strcmp(item_name(1:5),'bead_')
        FileName(i) = item_name;
        ID(i) = str2double(item_name(6:end-4));
        keep_ind(i) = true;
    end
end
FileName = FileName(keep_ind);
ID = ID(keep_ind);

% How many post files did user select?
num_posts = length(FileName);

% Reserve memory
TAC = zeros(num_posts,3);
FOC = TAC;

% Images in Matlab have a reversed y-axis. The origin (0,0) is located in
% the NorthWest corner, not the SouthWest corner. As such if we plot the
% y-values directly the plot will be upside down. Note: This is only the
% case when you're plotting on empty axes. When superimposing trajectories
% onto an image the coordinate axes are correct and therefore the mapping
% of the trajectories onto them is fine.

fig_handle_trajectories = figure;
labels  = cell(num_posts,1);
for i = 1:num_posts
	next if i == 1 || rem(i,50)==0 || i==num_posts
text fprintf(1,'	Evaluating post trajectory %s of %s
...',
num2str(i),num2str(num_posts));
end
end

load([bead_path filesep FileName{i}]);

x = bsec(:,1)*microntopix ; %#ok<NODEF>
y = bsec(:,2)*microntopix;

plot(x,y,'color',[0.17, 0.51, 0.34]);
hold all;
if i == 1
axis('ij');
xlabel('x coordinate (pix)');
ylabel('y coordinate (pix)');
end
labels{i} = ['post' num2str(ID(i))];

TAC(i,1) = ID(i);
TAC(i,2) = mean(x);
TAC(i,3) = mean(y);

FOC(i,1) = ID(i);
FOC(i,2) = x(1);
FOC(i,3) = y(1);

% Make a copy of the first figure for superimposing means:
fig_handle_trajectories2 = figure;
copyobj(get(fig_handle_trajectories,'children'),fig_handle_trajectories2);
figure(fig_handle_trajectories2);
plot(TAC(:,2),TAC(:,3),'LineStyle','none','Marker','+',
'MarkerFaceColor','r','MarkerEdgeColor','r','MarkerSize',4);

% Add legends, for some reason if this is done before the means are
% superimposed you can't superimpose the means:
figure(fig_handle_trajectories)
legend(labels);
legend('hide');
figure(fig_handle_trajectories2)
legend(labels);
legend('hide');

% Save figures:
fprintf(1,'\tSaving figures
');
fprintf(fid,'\tSaving figures
');
saveas(fig_handle_trajectories, [save_path ... 
\`Post_Trajectories.fig'], 'fig');
saveas(fig_handle_trajectories2, [save_path ... 
\`Post_Trajectories_Centroids.fig'], 'fig');

% Save 'TAC' array in .mat and .txt form:
fprintf(1,'\tSaving variable(s)
');
fprintf(fid,'\tSaving variable(s)
');
save 'TimeAveragedCentroids.mat' TAC;
save 'FrameOneCentroids.mat' FOC;
fid2 = fopen([save_path '\Time_Avg_Centroids.txt'], 'wt');
fid3 = fopen([save_path '\Frame_One_Centroids.txt'], 'wt');
fprintf(fid2,'Post# \tx(t) (pix) \ty(t) (pix)
');
fprintf(fid3,'Post# x(0) (pix) y(0) (pix)
');
rows = size(TAC,1);
for k = 1:rows
    fprintf(fid2,'%f %f %f
',TAC(k,:));
    fprintf(fid3,'%f %f %f
',FOC(k,:));
end
fclose(fid2);
fclose(fid3);

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Idedrift,raw_dim] = DedriftImageForOverlays_v1(basepath,fovn,...
microntopix,driftmodel_um,save_path,fid)

% Get function name:
func_name = mfilename;
%**************************************************************************
fprintf(fid,'\n%s running ...
',func_name);

[ImName,ImPath] = uigetfile([basepath '/*.tif'],'Select .tif file in raw data folder to overlay post IDs and set centroid','MultiSelect','off');
I = imread([ImPath ImName]);

% Determine the frame number of this image:
frame_str = ImName(5+length(num2str(fovn)):end-4);
frame = str2double(frame_str);
Idedrift = I;
raw_dim = size(I);

% If user selected a frame other than the first, retrieve the associated
% drift from the model:
if frame>1
    dx_um = driftmodel_um(frame-1,1);
dy_um = driftmodel_um(frame-1,2);
    % convert to pix
    dx_pix = dx_um*microntopix;
dy_pix = dy_um*microntopix;
% round to nearest whole pixel value:
    dx_pix = round(dx_pix);
dy_pix = round(dy_pix);
% Subtract off the drift but preserve the original Image size so add on an
% equivalent zero margin:
    [r,c] = size(I);
        if dx_pix > 0
            Idedrift = horzcat(Idedrift(:,dx_pix+1:end),zeros(r,dx_pix));
        elseif dx_pix < 0
            Idedrift = horzcat(zeros(r,abs(dx_pix)),Idedrift(:,1:end-abs(dx_pix)));
        end
        if dy_pix > 0
            Idedrift = vertcat(Idedrift(dy_pix+1:end,:),zeros(dy_pix,c));
        elseif dy_pix < 0
            Idedrift = vertcat(zeros(abs(dy_pix),c),Idedrift(1:end-abs(dy_pix),:));
        end

end

% scalefactor = 3;
% I = imresize(I,scalefactor);
% Idedrift = imresize(Idedrift,scalefactor); % make I scalefactorX as big
fig_handle_Idedrift = figure;
subplot(1,2,1);
imshow(I);
hold all;
axis('ij');
title(ImName,'Interpreter','none','FontSize',14);
subplot(1,2,2);
imshow(Idedrift);
hold all;
axis('ij');
title('Dedrifted','FontSize',14);

fprintf(1,'	 Saving variable(s) and figure(s)\n');
fprintf(fid,'	 Saving variable(s) and figure(s)\n');

save([save_path filesep ImName '_dedrifted.mat'],'Idedrift');
save([save_path filesep 'EmpiricalDataDimensions'],'raw_dim');
saveas(fig_handle_Idedrift, [save_path filesep ImName '_dedrifted.fig'],...
    'fig');
end
% Steven J. Henry
% 09/20/2014
% PURPOSE: This function overlays Post IDs as text on an image at the
% post positions in the image.
% %
% ASSUMPTIONS:
% % n/a
% % INPUT:
% bead_path = path to location of bead.mat data
% I = image matrix
% cat = string denoting category
% color = text color
% microntopix = objective calibration for conversion of mircons to pixels
% save_path = path to location of stored analysis
% fid = file ID of log file to which progress is recorded
% subsetIDs = post IDs to plot
% %
% OUTPUT:
% % n/a
% % DRIVER/FUNCTION MAP:
% % n/a (calls no subroutines)
% %**************************************************************************

function [] = OverlayPostIDs_v5(bead_path,I,cat,color,microntopix,...
    save_path,fid,subsetIDs)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Retrieve all contents that reside inside 'bead_path' folder:
contents = dir(bead_path);

num_items = size(contents,1);
FileName = cell(num_items,1);
ID = NaN(num_items,1);
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 10 && strcmp(item_name(1:5),'bead_')
        FileName{i} = item_name;
        ID(i) = str2double(item_name(6:end-4));
        keep_ind(i) = true;
    end
end
FileName = FileName(keep_ind);
ID = ID(keep_ind);

if nargin < 8
    subsetIDs = ID; %#ok<NASGU>
else
    % Further filter 'FileName' to include only those ID's that are present
    % in the user-supplied 'subsetIDs' vector:
    num_files = length(FileName);
    keep_ind2 = false(num_files,1);
    for j = 1:num_files
        insubset_test = ID(j)==subsetIDs;
    end
end
if sum(insubset_test)>0
    keep_ind2(j)=true;
end
FileName = FileName(keep_ind2);
ID = ID(keep_ind2);
end

% How many post files did user select?
num_posts = length(FileName);

for i = 1:num_posts
    if i == 1
        fig_handle_overlay = figure;
        imshow(I,'DisplayRange',[]);
        axis('ij');
        hold all;
    end
    load([bead_path filesep FileName{i}]);
    xpix(i) = bsec(1,1)*microntopix; %pix
    ypix(i) = bsec(1,2)*microntopix; %pix
    plot(xpix(i),ypix(i),'LineStyle','none','LineWidth',1,'Marker','o',
         'MarkerEdgeColor','k','MarkerFaceColor',color,'MarkerSize',15);
    hold all;
    %   text(xpix(i)+2,ypix(i)+2,num2str(ID(i)),'Color','w','FontSize',
    %   8,'Background',color,'Margin',1);
end
xlabel('x coord (\mum)','FontSize',14);
ylabel('y coord (\mum)','FontSize',14);

% A figure with text only, no image underneath:
fig_handle_textonly = figure;
plot(xpix,ypix,'LineStyle','none','Marker','+','MarkerEdgeColor','r',
     'MarkerFaceColor','r','MarkerSize',6);
hold all;
axis('ij');
axis('square');

for i = 1:num_posts
    text(xpix(i),ypix(i),num2str(ID(i)),'Color','k','FontSize',8);
end
xlabel('x coord (\mum)','FontSize',14);
ylabel('y coord (\mum)','FontSize',14);

% Save figure:
fprintf(1,'	Saving variable(s) and figure(s)
');
fprintf(fid,'	Saving variable(s) and figure(s)
');

saveas(fig_handle_overlay,
       [save_path filesep 'PostID_Overlay_' cat ...
        '.fig']);
saveas(fig_handle_textonly,
       [save_path filesep 'PostID_TextOnly_' cat ...
        '.fig']);

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
function [] = Merge_Trajectories_v1(bead_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Initialize an exitflag to off:
exitflag = 0;

% Whiel exitflag is off:
while exitflag ~= 1

% Retrieve input from user. 'v' is either a numeric vector or the
% string 'DONE'
v = input(['Supply vector of bead.mat IDs (e.g. [1 2 3]) to merge; -1 when done
']);

% If 'v' is -1
if v==-1
    % User is finished entering vectors of bead IDs:
    fprintf(1,'%sYou have finished entering bead IDs to merge\n');
    fprintf(fid,'%sYou have finished entering bead IDs to merge\n');
    exitflag = 1;

else
    % Does the cell array to hold user-supplied vectors exist?
    if exist('m','var')==1
        % If so concatenate the existing array with a new cell to hold
        % the new vector:
        m_new = vertcat(m,cell(1,1));
        clear('m');
        m_new(end) = v;
        m = m_new;
    else
        % Otherwise this is the first time the user supplied a vector
        m = cell(1,1);
        m(1) = v;
    end

end
end
% User supplied vector
fprintf(1,'User supplied vector of IDs to merge: %s
',num2str(v));
fprintf(fid,'User supplied vector of IDs to merge: %s
',num2str(v));
end
end

% How many posts are being handled?
num_posts = size(m,1);

% Loop over the posts
for i = 1:num_posts
  % Extract the user-supplied vector
  v = m{i};
  % How many merges are going to occur?
  num_merges = length(v);
  % Loop over the merges to take place:
  for j = 1:num_merges
    % Load the bead.mat file, the variable name is 'bsec'
    bead_name = ['bead_' num2str(v(j)) '.mat'];
    load([bead_path filesep bead_name]);
    if j == 1
      % If this is the first time write bsec to a master array:
      bsec_master = bsec;
      clear('bsec');
    else
      % Otherwise append this bsec to the existing master array:
      bsec_master = vertcat(bsec_master,bsec); %#ok<AGROW>
      clear('bsec');
    end
  end
  % Once concatenated sort data by frame# and reassign ID at all frame
  % numbers with the ID in the first frame:
  [~,sort_index] = sort(bsec_master(:,3),'ascend');
  bsec_master_sorted = bsec_master(sort_index,:);
  id = bsec_master_sorted(1,4);
  bsec_master_sorted(:,4) = id;
  bsec = bsec_master_sorted;
  clear('bsec_master','bsec_master_sorted');
end

% Search for duplicate occurrances of a frame # in the concatenated
% bsec. This was found to occur in a few cases and implies that the
% merged trajectories were considered distinct objects for some
% portion of their tracking (usually 1-2 frames at most). My
% hypothesis is that this occurs when a post is highly likely from a
% highly deflected such that some of the sidewall becomes visible in
% addition to the tip.
% Retrieve the list of frame #s. This should be a sorted list.
frames = bsec(:,3);
% If the number of unique entries in 'frames' is less than the length
% of 'frames', there are duplicates.
if length(unique(frames))<length(frames)
  % Compute the consecutive differences (differences between adjacent
  % rows). This vector is length(frames)-1 long.
  dframes = diff(frames);
  % A difference of zero implies that the adjacent entries have the
  % same value.
  dup_rows = find(dframes==0);
if isempty(dup_rows)==0
    num_dups = length(dup_rows);
    % Setup a deletion index vector. This will hold ones
    % corresponding to rows that need to be deleted from 'bsec'.
    del_rows = false(size(bsec,1),1);
    for k = 1:num_dups
        % Row 5 of bsec contains the length of the trajectory at
        % that frame #. We want to retain the row corresponding to
        % the previous trajectory.
        % For example imagine two trajectories that belong to the
        % same object. Trajectory 1 is dropped at frame 1400:
        % bsec(1400,3) = 1400, frame #
        % bsec(1400,4) = 1, ID
        % bsec(1400,5) = 1400, length of trajectory at that frame
        % Trajectory 2 is started at frame 1400:
        % bsec(1400,3) = 1400, frame #
        % bsec(1400,4) = 2, ID
        % bsec(1400,5) = 1, length of trajectory at that frame
        % The following logic would retain frame 1400 belonging to
        % trajectory 1 and eliminate frame 1400 belonging to
        % trajectory 2
        a = bsec(dup_rows(k),5);
        b = bsec(dup_rows(k)+1,5);
        if a>b % keep row containing a, discard row containing b
            del_rows(dup_rows(k)+1)=true;
        else
            del_rows(dup_rows(k))=true;
        end
    end
    bsec(del_rows,:) = [];
end
% Sanity check:
frames = bsec(:,3);
if length(unique(frames))~=length(frames)
    fprintf(1,'Duplicate frame# present in bead_%s.mat after filtering\n',num2str(id));
    fprintf(fid,'Duplicate frame# present in bead_%s.mat after filtering\n',num2str(id));
end
end
% Now save bsec under 'id'
bead_name = ['bead_' num2str(id) '.mat'];
delete([bead_path filesep bead_name]);
save([bead_path filesep bead_name],'bsec');
end
% Eliminate this 'id' from 'v'
del_index = v==id;
v(del_index)=[];
end
% Delete bead files that were merged into bead_id.mat:
num_deletions = length(v);
for k = 1:num_deletions
    bead_name = ['bead_' num2str(v(k)) '.mat'];
delete([bead_path filesep bead_name]);
end
end
% Update log file that function is completed:
fprintf(fid,'%s completed\n',func_name);
end

PlotPostIDs_v1.m

1 % Steven J. Henry
2 % 07/21/2014
%**************************************************************************
% PURPOSE: This function overlays Post IDs on an image.
% %
% % ASSUMPTIONS:
% % n/a
% %
% % INPUT:
% % bead_path = path to location of bead.mat data
% % I = image matrix
% % microntopix = objective calibration for conversion of microns to pixels
% % savep_path = path to location of stored analysis
% % fid = file ID of log file to which progress is recorded
% %
% % OUTPUT:
% % n/a
% %
% % DRIVER/FUNCTION MAP:
% % n/a (calls no subroutines)
% %**************************************************************************

function [] = PlotPostID_v1(bead_path,microntopix,savep_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Select bead.mat files to be processed
listing = dir(bead_path);
num_listings = length(listing);

fig_handle_ids = figure;
for i = 1:num_listings
    if listing(i).isdir == 0
        bead_name = listing(i).name;
        if strcmp(bead_name(1:5),'bead_')==1
            load([bead_path filesep listing(i).name]);
            xpix = bsec(1,1)*microntopix; %pix
            ypix = bsec(1,2)*microntopix; %pix
            ID = bead_name(6:end-4);
            plot(xpix,ypix,'LineStyle','none','Marker','+','
                    'MarkerEdgeColor','r','MarkerFaceColor','r','
                    'MarkerSize',6);
            hold on;
            text(xpix,ypix,ID,'Color','k','FontSize',8);
        end
    end
end
axis('ij');

% Save figure:
fprintf(1,'	Saving figure
');
fprintf(fid,'	Saving figure
');
if exist('fig_handle_ids','var')==1
    saveas(fig_handle_ids,[savep_path filesep 'PostID_TextOnly.fig']);
end

end
% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
end

SetSuperResolutionRefinement_v1.m

% Steven J. Henry
% 06/12/2014

% PURPOSE: This function asks the user to supply a superresolution multiple
% (refine) and tests if the supplied value is allowable.
%
% ASSUMPTIONS:
% n/a
%
% INPUT:
% savep_path = path to location of stored analysis
% fid = file ID of log file to which progress is recorded
%
% OUTPUT:
% refine = user-set refinement value
%
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

%**************************************************************************
function [refine] = SetSuperResolutionRefinement_v1(save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Have user specify refinement value:
escape_flag = 0;
acceptable = [1,2,4,5,8,10,16,20,25,32,50,64,100];
while escape_flag == 0;
    set_refinement = input('Set refinement value {1,2,4,5,8,10,16,20,25,32,50,64,100}:');
    test = set_refinement == acceptable;
    if sum(test)== 1
        refine = set_refinement;
        fprintf(1,'	User selected a refinement of %s
	Centroids will be plotted to nearest %s
',num2str(refine),num2str(1/refine));
        fprintf(fid,'	User selected a refinement of %s
	centroids will be plotted to nearest %s
',num2str(refine),num2str(1/refine));
        % Turn escape flag on:
        escape_flag = 1;
    else
        fprintf(1,'	User requested refine = %s which is not an option.
',num2str(set_refinement));
        fprintf(fid,'	User requested refine = %s which is not an option.
',num2str(set_refinement));
    end
end

% Save refine:
save_name = 'refine.mat';
fprintf(1,'\savesaving \srefine,\save_name);fprintf(fid,'\save,\save_name);if exist('refine','var')==1
    save([save_path filesep save_name],'refine');
end

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
Construct_Bitmap_v4.m

% Steven J. Henry
% 06/12/2014

% PURPOSE:
% This function asks the user to select a refinement value and construct a
% binary bitmap with "on" ("1") pixels at the location of time averaged
% centroids. Centroid positions are placed to the nearest refinement
% multiple.

% For example if a refinement of 2 is specified than centroids can
% be placed to the nearest 1/2 = 0.5 pixel location. If a refinement of 10
% is specified than centroids can be positioned to the nearest 1/10 = 0.1
% pixels.

% ASSUMPTIONS:
% n/a

% INPUT:
% M = three column matrix
%   col1 = postID
%   col2 = x-coordinate of centroid (pix)
%   col3 = y-coordinate of centroid (pix)
% dim = dimensions of empirical data bitmap [#rows, #cols]
% refine = refinement value to construct super-resolution bitmap
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% A = binary array with on (1) pixels located at centroid positions
% LUT = look-up-table mapping postID of each centroid in A

% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

%**************************************************************************

function [A,LUT] = Construct_Bitmap_v4(M,dim,refine,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,'%s running ...
',func_name);

% How many post files did user pass?
num_posts = size(M,1);

% Round centroid locations to nearest multiple of 'refine':
ID = M(:,1);
X = round(M(:,2)*refine);
Y = round(M(:,3)*refine);

% Reserve memory
A = zeros(dim*refine);
[ylim,xlim] = size(A);
LUT = nan(size(M));

for k = 1:num_posts
    i = Y(k);
    j = X(k);
    if 1<=i & i<=ylim & 1<=j & j<=xlim %#ok<AND2>
        A(i,j) = 1;
        LUT(k,1) = ID(k);
    end
end
LUT(k,2) = j;
LUT(k,3) = i;
end

del_id = isnan(LUT(:,1));
LUT(del_id,:) = [];

% A couple of sanity checks:
if sum(sum(A))==num_posts
    fprintf(1,'\sum(sum(A)) = %s which = %s centroids supplied\n',... 
    num2str(sum(sum(A))),num2str(num_posts));
    fprintf(fid,\sum(sum(A)) = %s which = %s centroids supplied\n',... 
    num2str(sum(sum(A))),num2str(num_posts));
else
    fprintf(1,'**WARNING**: sum(sum(A)) = %s which ~= %s centroids supplied\n',... 
    num2str(sum(sum(A))),num2str(num_posts));
    fprintf(fid,**WARNING**: sum(sum(A)) = %s which ~= %s centroids supplied\n',... 
    num2str(sum(sum(A))),num2str(num_posts));
end

edges = zeros(4,1);
edges(1) = sum(A(:,1));
edges(2) = sum(A(:,end));
edges(3) = sum(A(1,:));
edges(4) = sum(A(end,:));
empty_edges = edges==0;
if sum(empty_edges)==0
    fprintf(1,'An "on" pixel resides at each edge of A\n');
    fprintf(fid,An "on" pixel resides at each edge of A\n);
else
    fprintf(1,' %s edges of bitmap have "on" pixels\n',... 
    num2str(sum(~empty_edges)));
    fprintf(fid,%s edges of bitmap have "on" pixels\n',... 
    num2str(sum(~empty_edges)));
end

% Update log file that function is completed:
fprintf(1,'%s completed\n',func_name);
fprintf(fid,'%s completed\n',func_name);

function [theta] = Find_Rotation_Angle_v1(A,save_path,fid)

Find_Rotation_Angle_v1.m
% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,’%n%s running ...
’,func_name);
fprintf(fid,’%n%s running ...
’,func_name);

% Plot array:
fig_handle_input = figure;
imshow(A,’DisplayRange’,[]);
title(’Draw a line along the lattice direction you want parallel with the vertical axis’,’FontSize’,14,’Interpreter’,’None’);

% Have user draw line alone lattice row that should be parallel with the
% vertical axis:

h_line = imline;
coords = wait(h_line);

% If user drew a perfectly vertical line (Y coordinates equivalent),
% than rotation is zero
if coords(1,2)==coords(2,2)
    theta = 0;
else
    % If user drew a perfectly horizontal line (X coordinates equivalent),
    % than the rotation is 90 deg and direction doesn’t matter:
    if coords(1,1)==coords(1,2)
        theta = 90;
    else
        % Otherwise the line is neither vertical or horizontal so set P2 (head
        % point) as coordinate with larger Y value and P1 (tail point) as
        % coordinate with smaller Y value:
        if coords(1,2)>coords(2,2)
            P2 = coords(1,:);
            P1 = coords(2,:);
        else
            P2 = coords(2,:);
            P1 = coords(1,:);
        end
        % Now compute unit vector from P1->P2:
        % v = <X2-X1, Y2-Y1>;
        % |v| = sqrt((X2-X1)^2+(Y2-Y1)^2);
        % vhat = v/|v|;
        vmag = sqrt((P2(1)-P1(1))^2+(P2(2)-P1(2))^2);
        vhat = [P2(1)-P1(1),P2(2)-P1(2)]/vmag;
        % In axis ’ij’ system with origin in NorthWest corner unit vector
        % ’ihat’ in positive y direction is [0 1]. The angle between ’ihat’ and
        % ’vhat’ is the arccos of the dot product:
        ihat = [0,1];
        theta = acosd(ihat(1)*vhat(1)+ihat(2)*vhat(2));
    end

    % Now compute slope to make sure you have the direction of rotation
    % correct:
    m = (P2(2)-P1(2))/(P2(1)-P1(1));
    if m>0 % we need a clockwise rotation so set theta to negative itself:
        theta = -theta;
    end
end

fprintf(1,’theta = %s deg’,num2str(theta));
fprintf(fid,’theta = %s deg’,num2str(theta));

% Save figure:
saveas(fig_handle_input, [save_path ’\PreRotated_Bitmap.fig’], ’fig’);
Apply_Rotation_v3.m

% Steven J. Henry
% 05/27/2014

%**************************************************************************
% PURPOSE:
% Performs a rotation by constructing a rotation matrix and applying that
% rotation matrix to the x,y coordinate pairs of the centroids supplied.
% %
% ASSUMPTIONS:
% n/a
% 
% INPUT:
% X = x-coordinate of centroid (pix)
% Y = y-coordinate of centroid (pix)
% theta = rotation in degrees, (+) counter clockwise, (-) clockwise
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded
% dim = dimensions of raw data image
% 
% OUTPUT:
% Xr = rotated x-coordinate of centroid (pix)
% Yr = rotated y-coordinate of centroid (pix)
% 
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)
% *---------------------------

function [Xr,Yr] = Apply_Rotation_v3(X,Y,theta,save_path,fid,dim)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Did user supply a 'dim' variable:
if nargin < 6
    % If not, assume supplied X and Y coordinates are already translated to
    % viewing window origin at center of image:
    yo = 0;
    xo = 0;
else
    if sum(dim<0)>0
        fprintf(1,'**WARNING** Passed dimensions are not positive!
');
        fprintf(fid,'**WARNING** Passed dimensions are not positive!
');
    end
    ymax = dim(1);
    xmax = dim(2);
    yo = ymax/2;
    xo = xmax/2;
end

% Save 'theta' value in .mat form:
fprintf(1,'Saving theta value
');
fprintf(fid,'Saving theta value
');
save([save_path "\theta.mat"],'theta');

close(fig_handle_input);
update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);

end

%**************************************************************************

Apply_Rotation_v3.m % Steven J. Henry % 05/27/2014 % % PURPOSE: % Performs a rotation by constructing a rotation matrix and applying that % rotation matrix to the x,y coordinate pairs of the centroids supplied. % % ASSUMPTIONS: % n/a % % INPUT: % X = x-coordinate of centroid (pix) % Y = y-coordinate of centroid (pix) % theta = rotation in degrees, (+) counter clockwise, (-) clockwise % save_path = string pointing to save location % fid = file ID of log file to which progress is recorded % dim = dimensions of raw data image % % OUTPUT: % Xr = rotated x-coordinate of centroid (pix) % Yr = rotated y-coordinate of centroid (pix) % % DRIVER/FUNCTION MAP: % n/a (calls no subroutines) % %**************************************************************************

function [Xr,Yr] = Apply_Rotation_v3(X,Y,theta,save_path,fid,dim)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Did user supply a 'dim' variable:
if nargin < 6
    % If not, assume supplied X and Y coordinates are already translated to
    % viewing window origin at center of image:
    yo = 0;
    xo = 0;
else
    if sum(dim<0)>0
        fprintf(1,'**WARNING** Passed dimensions are not positive!
');
        fprintf(fid,'**WARNING** Passed dimensions are not positive!
');
    end
    ymax = dim(1);
    xmax = dim(2);
    yo = ymax/2;
    xo = xmax/2;
end

% Save 'theta' value in .mat form:
fprintf(1,'Saving theta value
');
fprintf(fid,'Saving theta value
');
save([save_path "\theta.mat"],'theta');

close(fig_handle_input);
update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);

end
fprintf(1, 'Origin (xo,yo) = (%s,%s) pix\n', ...
num2str(xo),num2str(yo));
fprintf(fid, 'User-dictated viewing window is %s X %s pix^2\n', ...
num2str(ymax),num2str(xmax));
fprintf(fid, 'Origin (xo,yo) = (%s,%s) pix\n', ...
num2str(xo),num2str(yo));
end

% Sanity checks
[xrow,xcol] = size(X);
[yrow,ycol] = size(Y);
if xcol ~= 1 || ycol ~= 1
fprintf(1, '**WARNING** X and Y coordinates should be supplied in separate column vectors\n');
fprintf(fid, '**WARNING** X and Y coordinates should be supplied in separate column vectors\n');
end
if xrow ~= yrow
fprintf(1, '**WARNING** # x-coordinates (%s) ~= # y-coordinates (%s)\n',num2str(xrow),num2str(yrow));
fprintf(1, '**WARNING** # x-coordinates (%s) ~= # y-coordinates (%s)\n',num2str(xrow),num2str(yrow));
end

% Translate coordinates relative to image center:
X = X-xo;
Y = Y-yo;

% Reserve memory
Xr = zeros(size(X));
Yr = Xr;

num_centroids = length(X);
R = [cosd(theta), sind(theta); -sind(theta), cosd(theta)];
for i = 1:num_centroids
v = [X(i);Y(i)];
vr = R*v;
Xr(i) = vr(1);
Yr(i) = vr(2);
end
Xr = Xr+xo;
Yr = Yr+yo;

% Update log file that function is completed:
fprintf(1, '%s completed\n',func_name);
fprintf(fid, '%s completed\n',func_name);
end

Crop_Bitmap_v2.m
1 % Steven J. Henry
2 % 05/01/2014
13 %**************************************************************************
14 % PURPOSE:
15 % This function crops a supplied image array by an amount calculated from a
16 % user-specified rectangle. The goal is to set the crop boundary in such a
17 % way that the boundaries are periodic.
18 %
19 % ASSUMPTIONS:
20 % n/a
21 %
22 % INPUT:
23 % A = an image array
24 % dilate = integer value by which to expand pixels for purpose of
25 % visualization, (cropping is performed on undilated image)
26 % save_path = string pointing to save location
27 % fid = file ID of log file to which progress is recorded
28 %
29 % VARIABLE OUTPUT:
function [C, verts, pad] = Crop_Bitmap_v2(A, dilate, save_path, fid)
  % Get function name:
  func_name = mfilename;

  % Update log file that function is running:
  fprintf(1, '
%s running ...
', func_name);
  fprintf(fid, '
%s running ...
', func_name);

  % Dilate supplied image and plot:
  fig_handle_dilated_input = figure;
  imshow(imdilate(A, strel('disk', dilate)),'DisplayRange', []);
  title_string = 'Draw cropping rectangle, double click when done';
  title(title_string, 'FontSize', 14, 'Interpreter', 'None');

  % Position rectangle for cropping:
  h_rect = imrect;
  verts = wait(h_rect);
  verts = round(verts);

  % Crop
  C = imcrop(A, verts);

  % Determine pad size to undo crop later:
  [r, c] = size(A);
  xpre = verts(1) - 1;
  ypre = verts(2) - 1;
  xpost = c - (xpre + verts(3) + 1); % width does not include terminal point
  ypost = r - (ypre + verts(4) + 1); % height does not include terminal point
  pad = [ypre, xpre, ypost, xpost];

  % Plot cropped image:
  fig_handle_dilated_output = figure;
  imshow(imdilate(C, strel('disk', dilate)),'DisplayRange', []);
  title_string = 'Dilated cropped image';
  title(title_string, 'FontSize', 20, 'Interpreter', 'None');

  % Save figures:
  fprintf(1, '	Saving figures
');
  fprintf(fid, '	Saving figures
');
  saveas(fig_handle_dilated_input, [save_path ...
  '"Bitmap_Perturbed_PreCropped.fig'], 'fig');
  saveas(fig_handle_dilated_output, [save_path ...
  '"Bitmap_Perturbed_PostCropped.fig'], 'fig');

  % Save 'C' array in .mat and .txt form:
  fprintf(1, '	Saving C array in .mat and .txt form:
');
  fprintf(fid, '	Saving C array in .mat and .txt form:
');
  save('C.mat', 'C', 'verts');
  dlmwrite([save_path '"C.txt'], C, '	');

  % Update log file that function is completed:
  fprintf(1, '%s completed
', func_name);
  fprintf(fid, '%s completed
', func_name);
end

FilterForRestingLattice_v6.m

1 % Steven J. Henry
2 % 07/23/2014
% PURPOSE:
% This function filters a bitmap of post centroids to return the resting (i.e. unperturbed) lattice by filtering out "noise" (i.e. perturbation) in Fourier space.

% ASSUMPTIONS:

% INPUT:
% P = user supplied bitmap of perturbed lattice
% pts = # probe points
% method = 'auto' or 'manual'
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% R2 = filtered resting lattice of same type and size as P.

% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

function [R2] = FilterForRestingLattice_v6(P,pts,method,save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

iterate_flag = 1;
iteration = 1;

while iterate_flag == 1
    if iteration == 1
        % Perform FFT on P:
        fprintf(1,'
	Computing FFT of perturbed bitmap...
');
        fprintf(fid,'
	Computing FFT of perturbed bitmap...
');
        P_tilda = fft2(P);

        % Plot the FFT of P:
        fig_handle_FFT_P = figure;
        imshow(abs(fftshift(P_tilda)),'DisplayRange',[]);
        title('abs(fftshift(fft2(P)))','FontSize',20,'Interpreter','None');

        % Compute complex modulus to determine threshold value:
        abs_P_tilda = abs(P_tilda);

        % Reshape the matrix of complex moduli into a column vector to run through 'hist' function:
        [r,c] = size(abs_P_tilda);
        abs_P_tilda_v = reshape(abs_P_tilda,r*c,1);

        % Compute histogram of pixel intensities:
        nbins = sqrt(length(abs_P_tilda_v));
        [counts,loc] = hist(abs_P_tilda_v,nbins);

        % Plot histogram before threshold
        fig_handle_hist_FFT_P = figure;
        bar(loc,counts,1,'FaceColor','b','EdgeColor','b');
        title('Histogram of Complex Moduli','FontSize',20,'Interpreter','None');
        axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
        set(gca,'FontSize',14,'yscale','log');
    end

% Perform FFT on P:
    fprintf(1,'
	Computing FFT of perturbed bitmap...
');
    fprintf(fid,'
	Computing FFT of perturbed bitmap...
');
    P_tilda = fft2(P);

    % Plot the FFT of P:
    fig_handle_FFT_P = figure;
    imshow(abs(fftshift(P_tilda)),'DisplayRange',[]);
    title('abs(fftshift(fft2(P)))','FontSize',20,'Interpreter','None');

    % Compute complex modulus to determine threshold value:
    abs_P_tilda = abs(P_tilda);

    % Reshape the matrix of complex moduli into a column vector to run through 'hist' function:
    [r,c] = size(abs_P_tilda);
    abs_P_tilda_v = reshape(abs_P_tilda,r*c,1);

    % Compute histogram of pixel intensities:
    nbins = sqrt(length(abs_P_tilda_v));
    [counts,loc] = hist(abs_P_tilda_v,nbins);

    % Plot histogram before threshold
    fig_handle_hist_FFT_P = figure;
    bar(loc,counts,1,'FaceColor','b','EdgeColor','b');
    title('Histogram of Complex Moduli','FontSize',20,'Interpreter','None');
    axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
    set(gca,'FontSize',14,'yscale','log');
end
xlabel('Pixel Intensity','FontSize',16);
ylabel('Counts','FontSize',16);
[r,c] = size(P);
lower = 1;
upper = floor(log10(r^c))-1;
thresholds = round(logspace(lower,upper,pts));

% Reserve a figure to plot resolution metric results
fig_handle_resolution = figure;
labels = cell(1,1);

else
    if pos == 1
        lower = 10^floor(log10(thresholds(pos)))-1;
        upper = thresholds(pos+1);
    elseif pos == length(thresholds)
        lower = thresholds(pos-1);
        upper = 10^floor(log10(thresholds(pos)))+1;
    else
        lower = thresholds(pos-1);
        upper = thresholds(pos+1);
    end
    ldecade = floor(log10(lower));
    udecade = floor(log10(upper));
    thres_max = thresholds(pos);
    if ldecade == udecade || udecade-ldecade == 1
        thresholds = round(linspace(lower,upper,pts));
    else
        thresholds = round(logspace(log10(lower),log10(upper),pts));
    end
    % If threshold value of previous peak is not present in this new
    % thresholds list add it:
    existance = thresholds==thres_max;
    if sum(existance)==0
        thresholds = sort([thresholds thres_max]);
    end
end

fprintf(1,...
'\tThreshold (# retained pixels) range to be tested: %s
',...
num2str(thresholds));
fprintf(fid,...
'\tThreshold (# retained pixels) range to be tested: %s
',...
num2str(thresholds));
resolutions = zeros(size(thresholds));
for j = 1:length(thresholds)
    % Working from the last (highest pixel intensity bin) backwards,
    % determine how many bins you need to use to accumulate
    % 'thresholds' pixels:
    flag = 0;
    tally = 0;
    i = 1;
    while flag == 0
        % Shift the counts row vector by 'i' positions
        shifted_counts = circshift(counts,i,2);
        % Add the corresponding number of pixels residing in this bin
        tally = tally+shifted_counts(1);
        % to the total tally
        if tally >= thresholds(j)
flag = 1;
else
    % If not, increase the amount the circle shift occurs and
    % repeat the tally
    i = i+1;
end
end

% Knowing the last 'i' bins contain 'thresholds(j)' pixels we
% determine the intensity threshold. 'hist' returns 'loc' a column
% vector of bin centers. We will determine the bin edge that
% marks the start of the 'i'th bin and use that edge as our
% threshold value.

% This is half the distance b/n the two bin centers
delta = (loc(end-i+1)-loc(end-i))/2;

% This is the start edge of bin 'end-i+1', also the end edge of bin
% 'end-i'
thes = loc(end-i)+delta;

% Filter P_tilda only retaining pixels that have a complex modulus
% greater or equal to 'thes' and turning off (with complex zero)
% pixels that have modulus less than 'thes'
makecomplexzero = abs_P_tilda<thres;
P_tilda_filtered = P_tilda;
P_tilda_filtered(makecomplexzero) = complex(0,0);

% Plot P_tilda_filtered:
if strcmp(method,'manual')==1
    fig_handle_P_tilda = figure;
    imshow(imdilate(abs(fftshift(P_tilda_filtered)),...
                    strel('disk',3)),'DisplayRange',[]);
    title('dilated Complex Moduli of Thresholded P_tilda',...'
    'FontSize',20,'Interpreter','None');
end

% Perform inverse FFT to take filtered P matrix back to cartesian
% space:
R = ifft2(P_tilda_filtered);

% Plot R:
if strcmp(method,'manual')==1
    fig_handle_R = figure;
    imshow(R,'DisplayRange',[]);
    title('ifft2(Thresholded P_tilda)','FontSize',20,...'
    'Interpreter','None');
end

% Compute "resolution" metric
Rmax = max(max(R));
Rmin = min(min(R));
contrast = (Rmax-Rmin)/(Rmax+Rmin);
resolutions(j) = 1/contrast;

% Plot histogram and cut off point
if strcmp(method,'manual')==1
    fig_handle_hist_thres_FFT_P = figure;
    bar(loc(1:end-i),counts(1:end-i),1,'FaceColor','r',...
        'EdgeColor','k');
    hold all;
    bar(loc(end-i+1:end),counts(end-i+1:end),1,...
        'FaceColor',[0.17, 0.51, 0.34],'EdgeColor','k');
    title(['Complex Moduli of Perturbed Bitmap, Threshold = ' ...'
    num2str(thres) intensity counts'],'FontSize',20,...'
    'Interpreter','None');
    axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
    set(gca,'FontSize',14);
legend('Excluded','Retained');
legend('boxoff');
xlabel('Pixel Intensity','FontSize',16);
ylabel('Counts','FontSize',16);

close(fig_handle_P_tilda,fig_handle_R,fig_handle_hist_thres_FFT_P);
end

figure(fig_handle_resolution);
plot(thresholds,resolutions,'Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k');
hold all;
if iteration>1
    oldlabels = labels;
    newlabel = {[Iteration' num2str(iteration)]};
    labels = vertcat(oldlabels,newlabel);
else
    labels = {[Iteration' num2str(iteration)]};
end
set(gca,'xscale','log','FontSize',12);
xlabel('# pixels retained in fft2(pertrubed bitmap)','FontSize',14);
ylabel('"resolution" metric','FontSize',14);
legend(labels);

pos = find(resolutions==max(resolutions));
fprintf(1,'		Optimal threshold for this iteration = %s
',num2str(thresholds(pos)));
fprintf(fid,'		Optimal threshold for this iteration = %s
',num2str(thresholds(pos)));
fprintf(1,'		Resolution = %s
',num2str(max(resolutions)));
fprintf(fid,'		Resolution = %s
',num2str(max(resolutions)));

if strcmp(method,'manual')==1
    choice = menu('Iterate again or stop?','Another Iteration','Finished');
elseif strcmp(method,'auto')==1
    if length(thresholds(pos))>1
        choice = 2;
    else
        choice = 1;
    end
else choice = 1;
end

if choice == 1
    iteration = iteration+1;
else choice == 2
    iterate_flag = 0;
end

keyboard;

% Plot histogram and cut off point
fig_handle_hist_thres_FFT_P = figure;
bar(loc(1:end-i),counts(1:end-i),1,'FaceColor','r','EdgeColor','r');
hold all;
bar(loc(end-i+1:end),counts(end-i+1:end),1,'FaceColor',[0.17, 0.51, 0.34],'EdgeColor',[0.17, 0.51, 0.34]);
title(["Complex Moduli of Perturbed Bitmap, Threshold = ' ... num2str(thres) 'intensity counts']);
set(gca,'FontSize',14,'yscale','log');
legend('Excluded','Retained');
legend('boxoff');
xlabel('Pixel Intensity','FontSize',16);
ylabel('Counts','FontSize',16);
% Save R as an image rescaled so 0 = min(min(R)) and 255 = max(max(R))
R2 = R - min(min(R));
fig_handle_R2 = figure;
imshow(R2);
saveas(fig_handle_R2, [save_path 'IFFT_RestingLattice_Cropped.fig'], 'fig');

% Save figures:
fprintf(1, 'Saving figures
');
fprintf(fid, 'Saving figures
');
saveas(fig_handle_FFT_P, [save_path 'FFT_PerturbedLattice.fig'], 'fig');
saveas(fig_handle_hist_FFT_P, [save_path 'Histogram_FFT_PerturbedLattice.fig'], 'fig');
saveas(fig_handle_resolution, [save_path 'ResolutionMetric.fig'], 'fig');
saveas(fig_handle_hist_thres_FFT_P, [save_path 'Histogram_FFT_PerturbedLattice_Thresholded.fig'], 'fig');

% Save 'RestingLattice' array in .mat and .txt form:
RL = R2;
fprintf(1, 'Saving Resting Lattice array
');
fprintf(fid, 'Saving Resting Lattice array
');
save('RestingLattice.mat', 'RL');
dlmwrite([save_path 'RestingLattice.txt'], RL, '	');

% Update log file that function is completed:
fprintf(1, 'Completed
');
fprintf(fid, 'Completed
');
end

% Steven J. Henry
% 05/06/2014
%**************************************************************************
% PURPOSE:
% This function "uncrops" an array given the array vertices. It effectively
does this by padding the supplied array.
%**************************************************************************

% ASSUMPTIONS:
% 'pad' is supplied in the form of [ypre,xpre,ypost,xpost]
% INPUT:
% A = input cropped array
% pad = padding dimensions [ypre,xpre,ypost,xpost]
% dilate = integer value by which to expand pixels for purpose of
% visualization, (rotation is performed on undilated image)
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded
% OUTPUT:
% B = padded cropped array A
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

function [B] = UndoCrop_v1(A, pad, dilate, save_path, fid)

func_name = mfilename;

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1, '%s running ...
', func_name);
fprintf(fid, '%s running ...
', func_name);

UndoCrop_v1.m
% 'pad' is supplied in the form of [ypre,xpre,ypost,xpost]
B = padarray(A,[pad(1) pad(2)];'pre');
B = padarray(B,[pad(3) pad(4)];'post');

% Plot dilated bitmap
fig_handle_dilated_bitmap = figure;
imshow(imdilate(B,strel('disk',dilate)),'DisplayRange',[]);
title_string = ['num2str(dilate) ... % X dilated uncropped bitmap of resting lattice centroids'];
title(title_string,'FontSize',20,'Interpreter','None');

% Save figure:
saveas(fig_handle_dilated_bitmap, [save_path ... '\UncroppedDilatedRestingLatticeBitmap.fig','fig']);

% Save parameters:
UncroppedRestingLattice = B;
save([save_path '\UncroppedRestingLattice.mat'],'UncroppedRestingLattice');

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
end

KilfoilInitialize_v3.m

% Steven J. Henry
% 05/08/2014

%**************************************************************************
% PURPOSE:
% This function runs Maria Kilfoil's particle tracking routines to identify
% the resting lattice positions recovered from the Fourier filtering
% performed earlier.
%**************************************************************************

% ASSUMPTIONS:
% User supplies non-meaningful fovn = 0 and frame = 0 values.
% %
% % INPUT:
% % fovn = field of view number (should be zero)
% % frame = frame # (should be zero)
% % save_path = string pointing to save location
% % fid = file ID of log file to which progress is recorded
% %
% % OUTPUT:
% % M2 - All the features found from calling 'feature2D.m'
% % MT - All the features from 'feature2D.m' which were accepted given the
% % % criteria in 'KilfoilRestingLatticeParameters'
% % KilfoilRestingLatticeParameters = cell array storing parameters used to
% % % filter particles
% %
% % DRIVER/FUNCTION MAP:
% % 0 % KilfoilInitialize_v3.m
% % 1 % mpretrack_init.m (Kilfoil)
% % 2 %
% %**************************************************************************

% % Get function name:
% func_name = mfilename;
% %
% % Update log file that function is running:
% fprintf(1,'\n\n%s running ...
\n',func_name);
% fprintf(fid,'\n\n%s running ...
\n',func_name);
% %
% % Initialize parameters:
%
350
351    basepath = [save_path '\'];
352    featsize = 6; % radius of features in pixels
353    barint = 1; % minimum integrated intensity of feature below mask
354    barrg = 20; % maximum radius of gyration squared to be accepted
355    barcc = 1; % maximum eccentricity accepted, 0 = perfect circle
356    IdivRg = barint/barrg; % minimum ratio of integrated intensity to
357        % radius of gyration squared
358    Imin = 0; % minimum intensity of local maximum to be considered
359    masscut = 0; % parameter which defines a threshold for integrated
360        % intensity of features before position refinement,
361        % to speed up the code
362    field = 2; % 0.1 if interlaced, 2 if full frame
363
364    prompt={'featsize';...
365        'barint';...
366        'barrg';...
367        'barcc'};
368    name='Input Parameters for initialize.m';
369    numlines=1;
370
371    % Turn iterate flag on:
372    iterate_flag = 1;
373
374    while iterate_flag == 1
375        defaultanswer={num2str(featsize),...
376            num2str(barint),...
377            num2str(barrg),...
378            num2str(barcc)};
379        options.Resize='on';
380        options.WindowStyle='normal';
381        options.Interpreter='tex';
382        answer=inputdlg(prompt,name,numlines,defaultanswer);
383
384        % radius of features in pixels
385        featsize = str2num(answer{1});
386        % minimum integrated intensity of feature below mask
387        barint = str2num(answer{2});
388        % maximum radius of gyration squared to be accepted
389        barrg = str2num(answer{3});
390        % maximum eccentricity accepted, 0 = perfect circle
391        barcc = str2num(answer{4});
392
393        fig_handle_temp = figure;
394        [M2, MT] = mpretrack_init(basepath, featsize, barint,...
395            barrg, barcc, IdivRg, fovn, frame, Imin, masscut, field);
396
397        % Compute histograms of mod(x,1) and mod(y,1) to screen for pixel bias:
398        nbins = sqrt(length(MT(:,1)));
399        [xcounts,xloc] = hist(mod(MT(:,1),1),nbins);
400        [ycounts,yloc] = hist(mod(MT(:,2),1),nbins);
401
402        % Plot mod(x,1) and mod(y,1) to screen for pixel bias:
403        fig_handle_modhist = figure;
404        subplot(1,2,1)
405        bar(xloc,xcounts,1,'FaceColor','b','EdgeColor','k');
406        title('x coordinate bias','FontSize',20,'Interpreter','None');
407        xlabel('mod(x,1) (pix)','FontSize',14);
408        ylabel('counts','FontSize',14);
409        subplot(1,2,2)
410        bar(yloc,ycounts,1,'FaceColor','b','EdgeColor','k');
411        title('y coordinate bias','FontSize',20,'Interpreter','None');
412        xlabel('mod(y,1) (pix)','FontSize',14);
413        ylabel('counts','FontSize',14);
414
415        % Radius of gyration (col 4) vs. Integrated intensity (col 3)
416        fig_handle_Rg = figure;
417        plot(M2(:,3),M2(:,4),'LineStyle','none','Marker','o',...
% Eccentricity (col 5) vs. Integrated intensity (col 3)
fig_handle_eccentricity = figure;
plot(M2(:,3),M2(:,5),'LineStyle','none','Marker','o',
     'MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',8);
hold on
xlabel('Integrated Intensity');
ylabel('Eccentricity');
plot(MT(:,3),MT(:,5),'LineStyle','none','Marker','o',
     'MarkerEdgeColor','g','MarkerFaceColor','g','MarkerSize',8);
legend({'All Features','Accepted Features'},'Location','NorthWest');

choice = menu('Satisfied with initialization?','Yes',
    'No, re-initialize');
if choice == 1
    iterate_flag = 0;
else
    close(fig_handle_temp,fig_handle_modhist,fig_handle_Rg,...
         fig_handle_eccentricity);
end

% Save initialization parameters:
fprintf(1,'The final initialization parameters were:
');
fprintf(1,'basepath = %s
',basepath);
fprintf(1,'featsize = %s
',answer{1});
fprintf(1,'barint = %s
',answer{2});
fprintf(1,'barrg = %s
',answer{3});
fprintf(1,'barcc = %s
',answer{4});
fprintf(1,'IdivRg = %s
',num2str(IdivRg));
fprintf(1,'fovn = %s
',num2str(fovn));
fprintf(1,'frame = %s
',num2str(frame));
fprintf(1,'Imin = %s
',num2str(Imin));
fprintf(1,'masscut = %s
',num2str(masscut));
fprintf(1,'field = %s
',num2str(field));

fprintf(fid,'
The final initialization parameters were:
');
fprintf(fid,'basepath = %s
',basepath);
fprintf(fid,'featsize = %s
',answer{1});
fprintf(fid,'barint = %s
',answer{2});
fprintf(fid,'barrg = %s
',answer{3});
fprintf(fid,'barcc = %s
',answer{4});
fprintf(fid,'IdivRg = %s
',num2str(IdivRg));
fprintf(fid,'fovn = %s
',num2str(fovn));
fprintf(fid,'frame = %s
',num2str(frame));
fprintf(fid,'Imin = %s
',num2str(Imin));
fprintf(fid,'masscut = %s
',num2str(masscut));
fprintf(fid,'field = %s
',num2str(field));

% Save figures:
fprintf(1,'Saving figure(s)
');
fprintf(fid,'Saving figure(s)
');
saveas(fig_handle_modhist, 'Kilfoil_Initialize_PixelBiasScreen.fig','fig');
saveas(fig_handle_Rg, 'Kilfoil_Initialize_Rg.fig','fig');
saveas(fig_handle_eccentricity, 'Kilfoil_Initialize_Eccentricity.fig','fig');

% Save parameters:
fprintf(1,'Saving variable(s)
');
fprintf(fid,\"Saving variable(s)/n\);
KilfoilRestingLatticeParameters = horzcat(prompt,answer);
save([save_path '\KilfoilRestingLatticeParameters.mat'],...
'KilfoilRestingLatticeParameters');
save([save_path '\MT.mat'],'MT');
save([save_path '\M2.mat'],'M2');

% Update log file that function is completed:
fprintf(1,\"%s completed\n",func_name);
fprintf(fid,\"%s completed\n",func_name);

end

TrajectoriesRelativeToRestingLattice_v9.m

function [filtered_shift,sbead_path]...
    = TrajectoriesRelativeToRestingLattice_v9(R,P_LUT,microntopix,refine,...
        bead_path,save_path,fid)

% Get function name:
func_name = mfilename;

% Establish list of coordinates:
[ypos_R,xpos_R] = find(R==1);

% Perform nearest neighbors search on resting "R" and perturbed "P"
[idR,dist] = knnsearch(Rcoords,Pcoords);
unique_idR = unique(idR);
num_unique = length(unique_idR);
cull = zeros(length(idR),1);

% Sort through idR for duplicate hits:
for i = 1:num_unique
    test_id = unique_idR(i);
    test_indices = idR == test_id;
    min_dist = min(dist(test_indices));
    keep = (idR==test_id & dist==min_dist);
    if sum(keep)==1
        cull(keep) = 1;
    else
        new_keep = find(keep==1,1,'first');
        cull(new_keep) = 1;
    end
end

culled_idR = zeros(length(idR),1);
for i = 1:length(idR)
    if cull(i) == 1
        culled_idR(i) = idR(i);
    else
        culled_idR(i) = NaN;
    end
end

shift = P_LUT;
x_tail = NaN(size(shift,1),1);
y_tail = NaN(size(shift,1),1);
u = NaN(size(shift,1),1);
v = NaN(size(shift,1),1);

for i = 1:length(culled_idR)
    if isnan(culled_idR(i))==0
        x_tail(i) = Rcoords(culled_idR(i),1);
        y_tail(i) = Rcoords(culled_idR(i),2);
        u(i) = Pcoords(i,1)-x_tail(i);
        v(i) = Pcoords(i,2)-y_tail(i);
        shift(i,2) = u(i);
        shift(i,3) = v(i);
    else %otherwise entry in FOC did not have nearest neighbor in R
        shift(i,2) = NaN;
        shift(i,3) = NaN;
    end
end

% Make a new directory to store sbead.mat files:
newfolder = 'sbeads';
kdir(save_path,newfolder);
sbead_path = [save_path filesep newfolder];

% Retrieve all contents that reside inside 'bead_path' folder:
contents = dir(bead_path);
num_items = size(contents,1);
FileName = cell(num_items,1);
ID = NaN(num_items,1);
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
if length(item_name) >= 10 && strcmp(item_name(1:5),'bead_')
    FileName{i} = item_name;
    ID(i) = str2double(item_name(6:end-4));
    keep_ind(i) = true;
end

ID = ID(keep_ind);
FileName = FileName(keep_ind);

num_posts = length(FileName);
for i = 1:num_posts
    if i == 1 || rem(i,50)==0 || i==num_posts
        fprintf(1,'Shifting post trajectory %s of %s
','num2str(i),num2str(num_posts));
        fprintf(fid,'Shifting post trajectory %s of %s
','num2str(i),num2str(num_posts));
    end
    r = find(shift(:,1)==ID(i));
    if isempty(r)==0 && sum(isnan(shift(r,:)))==0
        load([bead_path filesep FileName{i}],'bsec');
        x = bsec(:,1); %um
        y = bsec(:,2); %um
        % These are the coordinates of the resting lattice position or the
        % time-average-centroid in the event no corresponding resting
        % lattice position was located.
        xshift = x(1)-shift(r,2)/(microntopix*refine);
        yshift = y(1)-shift(r,3)/(microntopix*refine);
        new_row = [xshift, yshift, 0, ID(i), 0];
        bsec_old = bsec;
        clear('bsec');
        bsec = vertcat(new_row,bsec_old);
        % Save file
        save([sbead_path filesep 's' FileName{i}],'bsec');
    end
end

% Prepare a histogram of distances between perturbed and resting positions:
% Filter out 'dist' values not relevant:
filtered_shift = shift(~del_idx,:); %del_idx = isnan(shift(:,2));
filtered_dist = sqrt(filtered_shift(:,2).^2+filtered_shift(:,3).^2);
[dcounts,dloc] = hist(filtered_dist,sqrt(length(filtered_dist)));

fig_handle_hist = figure;
sprintf('pixelbasis','Pixel Basis','FontSize',20,'Interpreter','None');
aaxis([min(dloc) max(dloc) 0 max(dcounts)]);
sprintf('Distance Perturbed to Resting (pix)','FontSize',16);
ylabel('Counts','FontSize',16);
sprintf('Distance Perturbed to Resting (um)','FontSize',16);
dloc_um = dloc./(microntopix*refine);
252 bar(dloc_um,dcounts,1,'FaceColor','b','EdgeColor','b');
253 title(['microntopix*refine = ' num2str(microntopix) ' pix/um X ' ...
254 num2str(refine)],'FontSize',20,'Interpreter','None');
255 axis([min(dloc_um) max(dloc_um) 0 max(dcounts)]);
256 set(gca,'FontSize',14);
257 xlabel('Distance Perturbed to Resting (\mum)','FontSize',16);
258 ylabel('Counts','FontSize',16);
259
260 % Prepare a quiver plot of vectors pointing from resting lattice
261 % (x_tail(i), y_tail(i)) to perturbed position in frame one (u(i),v(i)).
262 x_tail(del_idx)=[];
263 y_tail(del_idx)=[];
264 u(del_idx)=[];
265 v(del_idx)=[];
266 fig_handle_quiv = figure;
267 plot(Rcoords(:,1)/microntopix,Rcoords(:,2)/microntopix,...
268 'LineStyle','none','Marker','o','MarkerFaceColor','g',....
269 'MarkerEdgeColor','k','MarkerSize',4);
270 hold all
271 plot(Pcoords(:,1)/microntopix,Pcoords(:,2)/microntopix,...
272 'LineStyle','none','Marker','o','MarkerFaceColor','r',....
273 'MarkerEdgeColor','k','MarkerSize',4)
274 quiver(x_tail/microntopix,y_tail/microntopix,u/(microntopix*10),...
275 v/(microntopix*10));
276 axis('ij');
277 axis('square');
278 xlabel('x (\mum)','FontSize',16);
279 ylabel('y (\mum)','FontSize',16);
280 title('tail = resting lattice, head = frame 1 perturbed lattice',...
281 'FontSize',18);
282
283 % Prepare a scatter plot of differences in resting and perturbed lattice
284 % positions:
285 fig_handle_scatter = figure;
286 plot(shift(:,2),shift(:,3),'LineStyle','none','Marker','o',....
287 'MarkerFaceColor','b','MarkerSize',6);
288 axis('square');
289 xlabel(['\Deltax (superpix = pix/' num2str(refine) ')'],'FontSize',14);
290 ylabel(['\Deltay (superpix = pix/' num2str(refine) ')'],'FontSize',14);
291 title('Difference Resting to Perturbed','FontSize',16);
292
293 %Save plots if the figure handle exists as a variable:
294 fprintf(1,'%s completed
',func_name);
295 fprintf(fid,'%s completed
',func_name);
296 end
297 %**************************************************************************
298 % PURPOSE: This function computes the position of post trajectories
299
TrajectoriesInCellReferenceFrame_v5.m
% relative to a cell reference frame formed by the unit vectors parallel
% and orthogonal to the line that connects the user-define centroid with
% that post. This is essentially translating the post position (x,y) from
% the lab reference frame (cartesian space) into a cell reference frame
% (still cartesian space but rotated and translated).
%
% ASSUMPTIONS:
% The user-supplied centroid position is supplied in the same frame of
% reference as the post trajectories were constructed. In MATLAB images are
% by default processed with the origin (0,0) located in the North West
% corner of the image, not the South West corner. This is natural when you
% consider that an image is an array of pixels and indexing into/out of
% that array requires specifying an (i,j) pair relative to the North
% West corner of the array.
%
% INPUT:
% sbead_path = path to folder which contains sbead.mat
% microntopix = calibration in units of pixels/um
% I = image that will be used for user to select approximate geometric
% centroid of cell
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded
%
% OUTPUT:
% manual_centroid = user defined approximate geometric centroid
% rsbead_manualcentroid_path = path to folder containing rsbead.mat files
% of same size as sbead.mat files but with positions translated
% into the cell reference frame (radial and tangential deflections)
%
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)
%**************************************************************************
function [manual_centroid,rsbead_manualcentroid_path]...
    = TrajectoriesInCellReferenceFrame_v5(sbead_path,microntopix,I,...
        save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%s running ...
',func_name);
fprintf(fid,'%s running ...
',func_name);

% Open an image and select approximate geometry centroid of cell:
fig_handle_manual_centroid = figure;
imshow(I,'DisplayRange',[]);
title('Click on approximate geometric centroid','FontSize',14,...
    'Interpreter','None');
h_point = impoint;
coords = wait(h_point);

% Retrieve all contents that reside inside 'sbead_path' folder:
contents = dir(sbead_path);
num_items = size(contents,1);
FileName = cell(num_items,1);
keep_ind = false(num_items,1);
for i = 1:num_items
    item_name = contents(i).name;
    % Process each item_name...
if length(item_name) >= 11 && strcmp(item_name(1:6),'sbead_')
    FileName{i} = item_name;
    keep_ind(i) = true;
end
end

FileName = FileName(keep_ind);

% Make a new directory to store rsbead.mat files:
newfolder = 'rsbeads_manualcentroid';
mkdir(save_path,newfolder);
rsbead_manualcentroid_path = [save_path filesep newfolder];

% How many files:
um_posts = length(FileName);

for i = 1:num_posts
    load([sbead_path filesep FileName{i}],'bsec);
    r = bsec;
    xlab = bsec(:,1)*microntopix;
    ylab = bsec(:,2)*microntopix;
    % Zero out old positions:
r(:,1) = 0;
r(:,2) = 0;
    % Retrieve first observation of post:
    xp1 = bsec(1,1)*microntopix;
    yp1 = bsec(1,2)*microntopix;
    % Construct unit vectors:
    rmag = sqrt((xp1-xc)^2+(yp1-yc)^2);
    r_para_hat = [xp1-xc,yp1-yc]./rmag;
    r_perp_hat = [-(yp1-yc),xp1-xc]./rmag;
    % Translate coordinates:
    r(:,1) = (xlab-xp1)*r_para_hat(1)+(ylab-yp1)*r_para_hat(2);
    r(:,2) = (xlab-xp1)*r_perp_hat(1)+(ylab-yp1)*r_perp_hat(2);
    % Back to microns:
    r(:,1) = r(:,1)/microntopix;
    r(:,2) = r(:,2)/microntopix;
    % Clear original 'bsec'
    clear('bsec');
    bsec = r;
    % Save file
    save([rsbead_manualcentroid_path filesep 'r' FileName{i}],'bsec');
end

fprintf(1,'	Saving variable(s) and figure(s)\n');
fprintf(fid,'	Saving variable(s) and figure(s)\n');

save([save_path filesep 'UserSetCentroid_pix.mat'],'manual_centroid');
saveas(fig_handle_manual_centroid, [save_path filesep ...
    'UserSetCentroid.fig'], 'fig');

% Update log file that function is completed:
fprintf(fid,'%s completed\n',func_name);

end

IdentifyEngagedPosts_v4.m

% Steven J. Henry
% 09/22/2014
%**************************************************************************
% PURPOSE: A function that assists the user in selecting cell-engaged posts
% by plotting the variance of all posts. After dedrifting, non-cell engaged
% posts have extremely small variances compared to cell-engaged posts.
% Assumptions:
% n/a
% INPUT:
% bead_path = path to bead.mat files
% rsbead_manualcentroid_path = path to folder containing rsbead.mat files
% files of same size as sbead.mat files but with positions translated
% into the cell reference frame (radial and tangential deflections)
% microntopix = calibration in units of pixels/um
% I = image that will be used for user to select approximate geometric
% centroid of cell
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded

% OUTPUT:
% ID_in = list of post IDs declared cell-engaged
% ens_manualcentroid_rsbead_path = path to folder containing rsbead.mat
% files belonging to cell-engaged posts.

% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

function [ID_in,ens_manualcentroid_rsbead_path] = IdentifyEngagedPosts_v4(bead_path,rsbead_manualcentroid_path,...
microntopix,I,save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(fid,'%s running ...
',func_name);

% Retrieve all contents that reside inside 'rsbead_manualcentroid_path' folder:
contents = dir(rsbead_manualcentroid_path);
num_items = size(contents,1);
FileName = cell(num_items,1);
ID = NaN(num_items,1);
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
        FileName{i} = item_name;
        ID(i) = str2double(item_name(8:end-4));
        keep_ind(i) = true;
    end
end

FileName = FileName(keep_ind);
ID = ID(keep_ind);

% How many post files did user select?
num_posts = length(FileName);
x_var = zeros(num_posts,1);
y_var = x_var;
satisfied = 0;

while satisfied ~= 1
    if satisfied == 0
        for i = 1:num_posts
            item_name = contents(i).name;
            if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
                FileName(i) = item_name;
                ID(i) = str2double(item_name(8:end-4));
                keep_ind(i) = true;
            end
        end
        FileName = FileName(keep_ind);
        ID = ID(keep_ind);
        satisfied = 0;
    end
    % If this is the first time in this loop have the user select the
    % rsbead.mat files you're filtering:
    if satisfied == 0
        for i = 1:num_posts
...
if i == 1 || rem(i,50)==0 || i==num_posts
    fprintf(1,'Evaluating post trajectory %s of %s
',num2str(i),num2str(num_posts));
end
load([rsbead_manualcentroid_path filesep FileName{i}]);
x = bsec(2:end,1)*microntopix; %#ok<NODEF>
y = bsec(2:end,2)*microntopix;
x_var(i) = var(x);
y_var(i) = var(y);
end
fig_handle_var_ids = figure;
plot(x_var,y_var,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','b');
hold all;
axis('square');
set(gca,'FontSize',12);
xlabel('VAR(r(| | |)) (pix^2)','FontSize',14);
ylabel('VAR(r(\perp)) (pix^2)','FontSize',14);

h = impoly;
verts = wait(h);
data_in = inpolygon(x_var,y_var,verts(:,1),verts(:,2));

fig_handle_engaged_overlay = figure;
imshow(I);
axis('ij');
axis('square');
hold all;
num_in = sum(data_in);
ID_in = ID(data_in);
for j = 1:num_in
    loadID = ID_in(j);
    load([bead_path filesep 'bead_' num2str(loadID) '.mat']);
x = bsec(:,1)*microntopix; %pix
y = bsec(:,2)*microntopix; %pix
plot(x,y,'Color',[0.17, 0.51, 0.34]);
text(x(1)-5,y(1)-5,num2str(loadID),'Color','k','FontSize',8,'Background','r','Margin',1);
end
choice = menu('Satisfied with filter?','No, repeat','Yes, done');
if choice == 1
    close(fig_handle_var,fig_handle_engaged_overlay);
% -1 avoids us having to reload the sbead.mat and bead.mat files
satisfied = -1;
else
  satisfied = 1;
end
end

% Make a new directory to store a copy of the rsbead.mat files:
newfolder = 'ens_manualcentroid';
mkdir(sbead_manualcentroid_path,newfolder);
ens_manualcentroid_rsbead_path = [rsbead_manualcentroid_path filesep ...
  newfolder];

for k = 1:num_in
  file = ['rsbead_' num2str(ID_in(k)) '.mat'];
  source = [rsbead_manualcentroid_path filesep file];
  copyfile(source,ens_manualcentroid_rsbead_path);
end

%Save plots:
fprintf(1,'	Saving variable(s) and figure(s)
');
fprintf(fid,'	Saving variable(s) and figure(s)
');

save([save_path filesep 'EngagedPostIDs.mat'],'ID_in');
saveas(fig_handle_var_ids,[save_path ...
  '\VarianceFilterScatterPlot_IDs.fig']);
saveas(fig_handle_var,[save_path ...
  '\VarianceFilterScatterPlot.fig']);
saveas(fig_handle_engaged_overlay,[save_path ...
  '\CellEngagedPostsOverlay.fig']);

end

%**************************************************************************
% PURPOSE: After the set of cell-engaged posts has been determined using an
% approximate user-selected geometric centroid the true geometric centroid
% of the set of engaged posts is determined. Then the cell ref trajectories
% (% sbead.mat') files are revised.
% % ASSUMPTIONS:
% % INPUT:
% % sbead_path = path to sbead.mat files
% % engaged_post_IDs = list of post IDs declared cell-engaged
% % microntopix = calibration in units of pixels/um
% % save_path = string pointing to save location
% % fid = file ID of log file to which progress is recorded
% % OUTPUT:
% % centroid = revised geometric centroid based-upon set of cell-engaged post
% % rsbead_path = path to rsbead.mat files (now revised)
% % ens_rsbead_path = path to cell-engaged rsbead.mat files (now revised)
% % % DRIVER/FUNCTION MAP:
% % n/a (calls no subroutines)
% %****************************************************************************

function [centroid,rsbead_path,ens_rsbead_path] = ReviseCellRefTrajectories_v1(sbead_path,engaged_post_IDs,...
  microntopix,save_path,fid)
% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'\n\n%s running ...
',func_name);
fprintf(fid,'\n\n%s running ...
',func_name);

% Compute the actual geometric centroid of the set of engaged posts using
% their associated resting lattice positions:
num_engaged = length(engaged_post_IDs);

xp = zeros(num_engaged,1);
yp = xp;
for i = 1:num_engaged
    ID = engaged_post_IDs(i);
    load([sbead_path filesep 'sbead_' num2str(ID) '.mat'],'bsec');
    xp(i) = bsec(1,1)*microntopix; %pix
    yp(i) = bsec(1,2)*microntopix; %pix
end
xc = mean(xp);
yc = mean(yp);
centroid = [xc,yc];

fprintf(1,...
\n\tGeometric centroid of engaged posts (xc,yc) = (%s,%s) pix
',num2str(xc),num2str(yc));
fprintf(fid,...
\n\tGeometric centroid of engaged posts (xc,yc) = (%s,%s) pix
',num2str(xc),num2str(yc));

% Retrieve all contents that reside inside 'sbead_path' folder:
contents = dir(sbead_path);

num_items = size(contents,1);
FileName = cell(num_items,1);
keep_ind = false(num_items,1);
for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 11 && strcmp(item_name(1:6),'sbead_')
        FileName{i} = item_name;
        keep_ind(i) = true;
    end
end
FileName = FileName(keep_ind);

% Make a new directory to store rsbead.mat files:
newfolder = 'rsbeads';
mkdir(save_path,newfolder);
rsbead_path = [save_path filesep newfolder];

% How many files:
num_posts = length(FileName);
for i = 1:num_posts
    load([sbead_path filesep FileName{i}],'bsec');
r = bsec;
    xlab = bsec(:,1)*microntopix;
ylab = bsec(:,2)*microntopix;
    % Zero out old positions:
    r(:,1) = 0;
r(:,2) = 0;
    % Retrieve first observation of post:
    xp1 = bsec(1,1)*microntopix;
    yp1 = bsec(1,2)*microntopix;
    % Construct unit vectors:
    rmag = sqrt((xp1-xc)^2+(yp1-yc)^2);
r_para_hat = [xp1-xc,yp1-yc]/rmag;
r_perp_hat = [-(yp1-yc),xp1-xc]/rmag;
% Translate coordinates:
\[ r(:,1) = (xlab-xp1)*r_para_hat(1)+(ylab-yp1)*r_para_hat(2); \]
\[ r(:,2) = (xlab-xp1)*r_perp_hat(1)+(ylab-yp1)*r_perp_hat(2); \]
% Back to microns:
\[ r(:,1) = r(:,1)/\text{microntopix}; \]
\[ r(:,2) = r(:,2)/\text{microntopix}; \]
% Clear original 'bsec'
clear('bsec');
bsec = r;
% Save file
save([rsbead_path filesep 'r' FileName{i}],'bsec');
end

% Now retrieve the revised 'rsbead.mat' files that belong to the set of
% engaged posts:
newfolder = 'ens';
mkdir(rsbead_path,newfolder);
ens_rsbead_path = [rsbead_path filesep newfolder];
for i = 1:num_engaged
    ID = engaged_post_IDs(i);
    file = ['rsbead_' num2str(ID) '.mat'];
    source = [rsbead_path filesep file];
    destination = [ens_rsbead_path filesep file];
copyfile(source,destination);
end
fprintf(1,'Saving variable(s) and figure(s)\n');
\% Update log file that function is completed:
fprintf(fid,'Saving variable(s) and figure(s)\n');
save([save_path filesep 'GeometricCentroid_pix.mat'],'centroid');

GeoSortEngagedPosts_v2.m

% Steven J. Henry
% 09/22/2014
%**************************************************************************
% PURPOSE: This has the user set a threshold in nearest-neighbor distance
% to sort peripheral posts from core posts.
%**************************************************************************
% ASSUMPTIONS: n/a
% INPUT:
\% sbead_path = path to sbead.mat files
\% rsbead_path = path to rsbead.mat files
\% microntopix = calibration in units of pixels/um
\% engaged_IDs = list of cell-engaged post IDs
\% I = image for superposition of geometrically sorted post IDs
\% save_path = string pointing to save location
\% fid = file ID of log file to which progress is recorded
\% OUTPUT:
\% core_IDs = list of posts residing in geometric core
\% core_rsbead_path = path to rsbead.mat files belonging to core posts
\% perim_IDs = list of posts residing at geometric periphery
\% perim_rsbead_path = path to rsbead.mat files belonging to periphery posts
\%**************************************************************************
% DRIVER/FUNCTION MAP: n/a (calls no subroutines)
function [core_IDs, core_rsbead_path, perim_IDs, perim_rsbead_path]... = GeoSortEngagedPosts_v2(sbead_path,rsbead_path,microntopix,...
    engaged_IDs,l,save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

% Retrieve all contents that reside inside 'sbead_path' folder:
contents = dir(sbead_path);
num_items = size(contents,1);
FileName = cell(num_items,1);
ID = NaN(num_items,1);
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 11 && strcmp(item_name(1:6),'sbead_')
        FileName{i} = item_name;
        ID(i) = str2double(item_name(7:end-4));
        keep_ind(i) = true;
    end
end
FileName = FileName(keep_ind);
ID = ID(keep_ind);

% Loop over all the files in 'FileName' and retain only those that have and
% ID that is also found within the 'engaged_IDs' list:
num_files = length(FileName);
resting_coords = NaN(num_files,6);
keep_ind = false(num_files,1);
for i = 1:num_files
    engaged_test = ID(i) == engaged_IDs;
    if sum(engaged_test)>0
        keep_ind(i) = true;
        load([sbead_path filesep FileName{i}]);
        x = bsec(1,1)*microntopix; %pix
        y = bsec(1,2)*microntopix; %pix
        resting_coords(i,1) = ID(i);
        resting_coords(i,2) = x;
        resting_coords(i,3) = y;
    end
end

% Eliminate rows without data as these rows were reserved for files within
% 'FileName' that did not belong to engaged posts.
resting_coords = resting_coords(keep_ind,:);

% For each entry in 'resting_coords' compute the mean distance to the six
% nearest neighbors:
um_posts = size(resting_coords,1);
for i = 1:num_posts
    xy = resting_coords(i,2:3);
    keep_ind = true(num_posts,1);
    keep_ind(i) = false;
xy_subset = resting_coords(keep_ind,2:3);

 [~,dist] = knnsearch(xy_subset,xy,'k',6);

 resting_coords(i,4) = mean(dist);
 resting_coords(i,5) = std(dist);
 resting_coords(i,6) = nnz(dist);

 end

 % Plot a histogram of the average 6 nearest neighbor distances for each
 % post
 [dcounts,dloc] = hist(resting_coords(:,4),sqrt(num_posts));

 fig_handle_dist = figure;
 subplot(1,2,1)
 bar(dloc,dcounts,1,'FaceColor','b','EdgeColor','b');
 axis([min(dloc) max(dloc) 0 max(dcounts)]);
 set(gca,'FontSize',14);
 xlabel('<knn> dist (pix), k = 6','FontSize',16);
 ylabel('Counts','FontSize',16);

 % Plot scatter plot of the average 6 nearest neighbor distances for each
 % post with the associated standard deviation in rank order from smallest
 % to largest
 [~,sort_ind] = sort(resting_coords(:,4),'ascend');
 sorted_resting_coords = resting_coords(sort_ind,:);

 subplot(1,2,2)
 errorbar(sorted_resting_coords(:,4),sorted_resting_coords(:,5),...
 'LineStyle','none','Color',[0.83,0.82,0.78],'Marker','o',...
 'MarkerFaceColor','b','MarkerEdgeColor','b','MarkerSize',8);
 set(gca,'FontSize',14);
 xlabel('Rank Order','FontSize',16);
 ylabel('<knn> dist (pix) ± SD, k = 6','FontSize',16);

 % Have user set a threshold above which will be declared peripheral posts,
 % below which will be declared core posts:
 thres = input('Set nearest neighbor distance (pix) threshold below which posts are "core", above which posts are "perim" = 
');

 fprintf(1,'	User set core vs. peripheral nearest neighbor distance threshold = %s pix
',num2str(thres));
 fprintf(fid,'	User set core vs. peripheral nearest neighbor distance threshold = %s pix
',num2str(thres));

 core_ind = sorted_resting_coords(:,4)<thres;
 core_IDs = sorted_resting_coords(core_ind,1);
 perim_IDs = sorted_resting_coords(~core_ind,1);

 % Overlay the post IDs for each category (core vs. perim):
 fig_handle_ids = figure;

 % Make a new directory to store a copy of the rsbead.mat files belonging to
 % the core posts:
 newfolder = 'core';
 mkdir(rsbead_path,newfolder);
 core_rsbead_path = [rsbead_path filesep newfolder];
 num_core = length(core_IDs);
 for k = 1:num_core
 file = ['rsbead_' num2str(core_IDs(k)) '.mat'];
 source = [rsbead_path filesep file];
 copyfile(source,core_rsbead_path);
 load([sbead_path filesep 'sbead_' num2str(core_IDs(k)) '.mat'],'bsec');
 xo = bsec(1,1)*microntopix; %pix
 yo = bsec(1,2)*microntopix; %pix
 subplot(1,2,1);
 imshow(I,'DisplayRange',[]);
%**************************************************************************
% PURPOSE: User selects folder containing rsbead.mat files for three
categories: ensemble, core, and peripheral. The function records
% the IDs of the rsbead.mat files within those respective folders.
% % ASSUMPTIONS:
% % n/a
% % INPUT:
% % ens_rsbead_path = path to rsbead.mat files of cell-engaged posts
% % core_rsbead_path = path to rsbead.mat files of core cell-engaged posts

axis('ij');
hold all;
title('Core Posts','FontSize',16);
xlabel('x coord (pix)','FontSize',14);
ylabel('y coord (pix)','FontSize',14);
end

text(xo,yo,num2str(core_IDs(k)),'Color','k','FontSize',8,...
'Background',[0.17, 0.51, 0.34],'Margin',1);
end

% Make a new directory to store a copy of the rsbead.mat files belonging to
% the peripheral posts:
newfolder = 'perim';
mkdir(rsbead_path,newfolder);
perim_rsbead_path = [rsbead_path filesep newfolder];
um_perim = length(perim_IDs);
for k = 1:num_perim
    file = ['rsbead_ ' num2str(perim_IDs(k)) '.mat'];
    source = [rsbead_path filesep file];
copyfile(source,perim_rsbead_path);
load([sbead_path filesep 'sbead_' num2str(perim_IDs(k)) '.mat'],...
'bsec');
xo = bsec(1,1)*microntopix; %pix
yo = bsec(1,2)*microntopix; %pix
subplot(1,2,2);
if k == 1
    imshow(I,'DisplayRange',[]);
    axis('ij');
    hold all;
title('Perim Posts','FontSize',16);
    xlabel('x coord (pix)','FontSize',14);
    ylabel('y coord (pix)','FontSize',14);
end
    text(xo,yo,num2str(perim_IDs(k)),'Color','k','FontSize',8,...
        'Background','r','Margin',1);
end

% Save figure:
fprintf(1,'	Saving variable(s) and figure(s)
');
fprintf(fid,'	Saving variable(s) and figure(s)
');
save([save_path filesep 'Core_PostIDs.mat'],'core_IDs');
save([save_path filesep 'Perim_PostIDs.mat'],'perim_IDs');
saveas(fig_handle_dist,[save_path filesep 'PerimVsCoreKnnDist.fig']);
saveas(fig_handle_ids,[save_path filesep 'PerimVsCoreIDsOverlay.fig']);

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
end
% perim_rsbead_path = path to rsbead.mat files of peripheral cell-engaged
% posts
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded
% OUTPUT:
% engaged_post_IDs = list of cell-engaged posts
% core_post_IDs = list of core cell-engaged posts
% perim_post_IDs = list of peripheral cell-engaged posts
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)
%**************************************************************************

function [engaged_post_IDs, core_post_IDs, perim_post_IDs]...
    = RepopulatePostIDLists_v2(ens_rsbead_path,core_rsbead_path,...
    perim_rsbead_path,save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

for i = 1:3
    if i == 1
        % Set path to folder containing ensemble of engaged posts:
        path = ens_rsbead_path;
    elseif i == 2
        % Set path to folder containing ensemble of engaged posts:
        path = core_rsbead_path;
    elseif i == 3
        % Set path to folder containing ensemble of engaged posts:
        path = perim_rsbead_path;
    end

    contents = dir(path);
    num_items = size(contents,1);
    ID = NaN(num_items,1);

    for j = 1:num_items
        item_name = contents(j).name;
        if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
            ID(j) = str2double(item_name(8:end-4));
        end
    end

    del_ind = isnan(ID);
    ID(del_ind) = [];

    if i == 1
        engaged_post_IDs = ID;
    elseif i == 2
        core_post_IDs = ID;
    elseif i == 3
        perim_post_IDs = ID;
    end

    clear('ID');
end

% Save figure:
fprintf(1,'\nSaving variable(s) and figure(s)\n');
fprintf(fid,'\nSaving variable(s) and figure(s)\n');
save([save_path filesep 'ManualEdit_Ens_PostIDs.mat'],'engaged_post_IDs');
save([save_path filesep 'ManualEdit_Core_PostIDs.mat'],'core_post_IDs');
save([save_path filesep 'ManualEdit_Perim_PostIDs.mat'],'perim_post_IDs');

% Update log file that function is completed:
fprintf(1,'%s completed
',func_name);
fprintf(fid,'%s completed
',func_name);
end

function [mpara,sdpara,cpara,separa,...
    mperp,sdperp,cperp,seperp,...
    mpara_F,sdpara_F,cpara_F,separa_F,...
    mperp_F,sdperp_F,cperp_F,seperp_F,...
    t,Fmax,Fss] = PlotCellRefTrajectories_v7(cat_rsbead_path,time_int,kspring,...
     cat,color,save_path,fid,transition_time)

% Get function name:
func_name = mfilename;
% Update log file that function is running:
fprintf(1,"%n%s running ...
",func_name);
fprintf(fid,"%n%s running ...
",func_name);

% Retrieve all contents that reside inside 'cat_rsbead_path' folder:
contents = dir(cat_rsbead_path);

num_items = size(contents,1);
FileName = cell(num_items,1);
keep_ind = false(num_items,1);

for i = 1:num_items
    item_name = contents(i).name;
    if length(item_name) >= 12 && strcmp(item_name(1:7),\'rsbead_\')
        FileName{i} = item_name;
        keep_ind(i) = true;
    end
end

FileName = FileName(keep_ind);

n = length(FileName);
ID = NaN(n,1);
max_frame = [];
ymin = 0;
ymax = 0;

t = frames*time_int;

load([cat_rsbead_path filesep FileName{1}],'bsec');
rpara = bsec(:,1)*1000; %#ok<NODEF> % nm
rperp = bsec(:,2)*1000; % nm
frames = bsec(:,3);

if isempty(max_frame)
    max_frame = max(frames);
else
    max(frames)>max_frame
end
max_frame = max(frames);
end
ID(i) = bsec(1,4);
if isempty(max_frame)
    max_frame = max(frames);
else
    max(frames)>max_frame
end
max_frame = max(frames);
end

if min(rpara)<ymin
    ymin = min(rpara);
end
if min(rperp)<ymin
    ymin = min(rperp);
end
if max(rpara)>ymax;
    ymax = max(rpara);
end
if max(rperp)>ymax;
    ymax = max(rperp);
end

t = frames*time_int;

if i == 1
    fig_handle_r = figure;
    fig_handle_f = figure;
end
labels = cell(n,1);
% Plot r vs. t trajectory:
figure(fig_handle_r)
subplot(1,2,1);
plot(t,rpara,'LineStyle','-','Color',color,'Marker','none');
if i == 1
title(['Individual r(t)_|_|, ' cat],'FontSize',20,...
    'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('r(t)_|_| (nm)','FontSize',16);
end
hold all;
if i == n
axis([0 max_frame*time_int ymin ymax]);
end
subplot(1,2,2);
plot(t,rperp,'LineStyle','-','Color',color,'Marker','none');
if i == 1
title(['Individual r(t)_\perp, ' cat],'FontSize',20,...
    'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('r(t)_\perp (nm)','FontSize',16);
end
hold all;
if i == n
axis([0 max_frame*time_int ymin ymax]);
end
labels{i} = ['post' num2str(ID(i))];

% Plot f vs. t trajectory:
figure(fig_handle_f)
subplot(1,2,1);
plot(t,rpara*kspring,'LineStyle','-','Color',color,'Marker','none');
if i == 1
title(['Individual f(t)_|_|, ' cat],'FontSize',20,...
    'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('f(t)_|_| (pN)','FontSize',16);
end
hold all;
if i == n
axis([0 max_frame*time_int ymin*kspring ymax*kspring]);
end
subplot(1,2,2);
plot(t,rperp*kspring,'LineStyle','-','Color',color,'Marker','none');
if i == 1
title(['Individual f(t)_\perp, ' cat],'FontSize',20,...
    'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('f(t)_\perp (pN)','FontSize',16);
end
hold all;
if i == n
axis([0 max_frame*time_int ymin*kspring ymax*kspring]);
end
labels{i} = ['post' num2str(ID(i))];

for k = 1:2
    if k == 1
        % ...
figure(fig_handle_r);
elseif k == 2
    figure(fig_handle_f);
end

subplot(1,2,1);
legend(labels);
legend('hide');
subplot(1,2,2);
legend(labels);
legend('hide');

% Create array for averaging:
Rpara = NaN(max_frame,n);
Rperp = Rpara;

for j = 1:n
    load([cat_rsbead_path filesep FileName{j}],'bsec');
    rpara = bsec(:,1);  % um
    rperp = bsec(:,2);  % um
    frames = bsec(:,3);
    % Exclude row 1 which contains the distance to the resting lattice
    % position from the resting lattice position (i.e. 0).
    rpara(1) = [];
    rperp(1) = [];
    frames(1) = [];
    n_frames = length(frames);
    for k = 1:n_frames
        print_row = frames(k);
        Rpara(print_row,j) = rpara(k);
        Rperp(print_row,j) = rperp(k);
    end
end

% Convert to nanometers
Rpara = Rpara*1000;
Rperp = Rperp*1000;

% Convert to piconewtons
Rpara_F = Rpara*kspring;
Rperp_F = Rperp*kspring;

[rows,cols] = size(Rpara);
mpara = NaN(rows,1);
sdpara = mpara;
cpara = mpara;
mperp = mpara;
sdperp = mpara;
cperp = mpara;

mpara_F = mpara;
sdpara_F = mpara;
cpara_F = mpara;
mperp_F = mpara;
sdperp_F = mpara;
cperp_F = mpara;

for k = 1:rows
    keep_id1 = ~isnan(Rpara(k,:));
    mpara(k) = mean(Rpara(k,keep_id1));
    sdpara(k) = std(Rpara(k,keep_id1));
    cpara(k) = sum(keep_id1);
    keep_id2 = ~isnan(Rperp(k,:));
    mperp(k) = mean(Rperp(k,keep_id2));
    sdperp(k) = std(Rperp(k,keep_id2));
end
cperp(k) = sum(keep_id2);
keep_id3 = ~isnan(Rpara_F(k,:));
mpara_F(k) = mean(Rpara_F(k,keep_id3));
sdpara_F(k) = std(Rpara_F(k,keep_id3));
cpara_F(k) = sum(keep_id3);
keep_id4 = ~isnan(Rperp_F(k,:));
mperp_F(k) = mean(Rperp_F(k,keep_id4));
sdperp_F(k) = std(Rperp_F(k,keep_id4));
cperp_F(k) = sum(keep_id4);
end
separa = sdpara./sqrt(cpara);
seperp = sdperp./sqrt(cperp);
separa_F = sdpara_F./sqrt(cpara_F);
seperp_F = sdperp_F./sqrt(cperp_F);
t = (0:size(Rpara,1)-1)'*time_int;

% Retrieve force of each post at Fmax of mpara_F and save to array for % future plotting.
[row_of_max] = max(mpara_F);
Fmax_vec = Rpara_F(row_of_max,:);
keep_id = ~isnan(Fmax_vec);
Fmax_vec = Fmax_vec(keep_id)';
ID_vec = ID(keep_id);
Fmax = horzcat(ID_vec,Fmax_vec);

% Retrieve steady state force of each post after 'transition_time' past % t_max
Fss = NaN(size(Fmax,1),3);
for k = 1:cols
    f_trajec = Rpara(:,k);
    keep_ind = ~isnan(f_trajec);
    t_trajec = t(keep_ind);
    f_trajec = f_trajec(keep_ind);
    avg_ind = t_trajec > t_Fmax+transition_time;
    Fss(k,1) = ID(k);
    Fss(k,2) = mean(f_trajec(avg_ind));
    Fss(k,3) = std(f_trajec(avg_ind));
end

% Displacements plots standard deviation bars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara-sdpara);
ymin(2) = min(mperp-sdperp);
ymin = min(ymin);
ymax(1) = max(mpara+sdpara);
ymax(2) = max(mperp+sdperp);
ymax = max(ymax);
fig_handle_rsdbars = figure;
subplot(1,2,1);
errorbar(t,mpara,sdpara,'LineStyle','none',',Color',[0.83,0.82,0.78],...'
'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...'
'MarkerSize',8);
title(['"Ensemble Average r(t)_|_|, ' cat two quotes]);
set(gca,'FontSize',14);
xlabel('time (s)',',FontSize',16);
ylabel('r(t)_|_| \pm SD (nm)',',FontSize',16);
axis([0 t(end) ymin ymax]);
subplot(1,2,2);
errorbar(t,mperp,sdperp,'LineStyle','none',',Color',[0.83,0.82,0.78],...
'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,....
'MarkerSize',8);
title(['Ensemble Average r(t)_\perp, ' cat],'FontSize',20,...
'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<r(t)_\perp> \pm SD (nm)','FontSize',16);
axis([0 t(end) ymin ymax]);

% Displacement plots standard error bars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara-separa);
ymin(2) = min(mperp-seperp);
ymin = min(ymin);
ymax(1) = max(mpara+separa);
ymax(2) = max(mperp+seperp);
ymax = max(ymax);

fig_handle_rsebars = figure;
subplot(1,2,1);
errorbar(t,mpara,separa,'LineStyle','none','Color',[0.83,0.82,0.78],...
'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
'MarkerSize',8);
title(['Ensemble Average r(t)_\perp, ' cat],'FontSize',20,...
'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<r(t)_\perp> \pm SE (nm)','FontSize',16);
axis([0 t(end) ymin ymax]);

% Displacement plots no error bars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara);
ymin(2) = min(mperp);
ymin = min(ymin);
ymax(1) = max(mpara);
ymax(2) = max(mperp);
ymax = max(ymax);

fig_handle_rmeans = figure;
subplot(1,2,1);
plot(t,mpara,'LineStyle','none','Marker','o','MarkerFaceColor',color,...
'MarkerEdgeColor',color,'MarkerSize',8);
title(['Ensemble Average r(t)_\perp, ' cat],'FontSize',20,...
'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<r(t)_\perp> (nm)','FontSize',16);
axis([0 t(end) ymin ymax]);
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<r(t)_\perp> (nm)','FontSize',16);
axis([0 t(end) ymin ymax]);

% Force plots standard deviation bars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara_F-sdpara_F);
ymin(2) = min(mperp_F-sdperp_F);
ymin = min(ymin);
ymax(1) = max(mpara_F+sdpara_F);
ymax(2) = max(mperp_F+sdperp_F);
ymax = max(ymax);

fig_handle_fsdbars = figure;
subplot(1,2,1);
errorbar(t,mpara_F,sdpara_F,'LineStyle','none','Color',[0.83,0.82,0.78],... 'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,... 'MarkerSize',8);
title(['Ensemble Average f(t)_|_|, ' cat],'FontSize',20,... 'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<f(t)_|_|> \pm SD (pN)','FontSize',16);
axis([0 t(end) ymin ymax]);

subplot(1,2,2);
errorbar(t,mperp_F,sdperp_F,'LineStyle','none','Color',[0.83,0.82,0.78],... 'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,... 'MarkerSize',8);
title(['Ensemble Average f(t)_\perp, ' cat],'FontSize',20,... 'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<f(t)_\perp> \pm SD (pN)','FontSize',16);
axis([0 t(end) ymin ymax]);

% Force plots standard error bars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara_F-separa_F);
ymin(2) = min(mperp_F-seperp_F);
ymin = min(ymin);
ymax(1) = max(mpara_F+separa_F);
ymax(2) = max(mperp_F+seperp_F);
ymax = max(ymax);

fig_handle_fsebars = figure;
subplot(1,2,1);
errorbar(t,mpara_F,separa_F,'LineStyle','none','Color',[0.83,0.82,0.78],... 'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,... 'MarkerSize',8);
title(['Ensemble Average f(t)_|_|, ' cat],'FontSize',20,... 'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<f(t)_|_|> \pm SE (pN)','FontSize',16);
axis([0 t(end) ymin ymax]);

subplot(1,2,2);
errorbar(t,mperp_F,seperp_F,'LineStyle','none','Color',[0.83,0.82,0.78],... 'Marker','o','Marker','o','MarkerFaceColor',color,... 'MarkerEdgeColor',color,'MarkerSize',8);
title(['Ensemble Average f(t)_\perp, ' cat],'FontSize',20,... 'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
% Force plots no errorbars
ymin = zeros(1,2);
ymax = ymin;
ymin(1) = min(mpara_F);
ymin(2) = min(mperp_F);
ymin = min(ymin);
ymax(1) = max(mpara_F);
ymax(2) = max(mperp_F);
ymax = max(ymax);

% Set figure and axes
fig_handle_fmeans = figure;
subplot(1,2,1);
plot(t,mpara_F,'LineStyle','none','Marker','o','MarkerFaceColor',color,...
'MarkerEdgeColor',color,'MarkerSize',8);
title(['Ensemble Average f(t)_|_|, ' cat], 'FontSize',20,...
'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<f(t)_|_|> (pN)','FontSize',16);
axis([0 t(end) ymin ymax]);

subplot(1,2,2);
plot(t,mperp_F,'LineStyle','none','Marker','o','Marker','o',...%
'MarkerEdgeColor',color,'MarkerEdgeColor',color,'MarkerSize',8);
title(['Ensemble Average f(t)_\perp, ' cat], 'FontSize',20,...
'Interpreter','Tex');
set(gca,'FontSize',14);
xlabel('time (s)','FontSize',16);
ylabel('<f(t)_\perp> (pN)','FontSize',16);
axis([0 t(end) ymin ymax]);

% Save plots
saveas(fig_handle_r,[save_path 'vst_' cat '.fig']);
saveas(fig_handle_rsdbars,[save_path 'vst_meanSDerrorbars_' cat '.fig']);
saveas(fig_handle_rsebars,[save_path 'vst_meanSEerrorbars_' cat '.fig']);
saveas(fig_handle_rmeans,[save_path 'vst_mean_' cat '.fig']);
saveas(fig_handle_f,[save_path 'vst_' cat '.fig']);
saveas(fig_handle_fsdbars,[save_path 'vst_mean_SDerrorbars_' cat ...
'.fig']);
saveas(fig_handle_fsebars,[save_path 'vst_mean_SEerrorbars_' cat ...
'.fig']);
saveas(fig_handle_fmeans,[save_path 'vst_mean_' cat '.fig']);

% Update log file that function is completed:
fprintf(1, "%s completed
", func_name);
fprintf(fid, "%s completed
", func_name);
end

% Steven J. Henry
% 09/25/2014
%**************************************************************************
% PURPOSE: To plot time at Fmax, Fmax, and <F(t)>thres> vs radial distance
% of post from centroid.
% ASSUMPTIONS:
% n/a
% INPUT:
% exp_date = 8 digit number date of experiment (yyyyymmmdd)
% exp_donor = donor ID string ('DXX')
% exp_cond = string describing experimental condition (e.g. 'Control')
% fvn = field of view number
% sbead_path = path to sbead.mat files
% centroid = geometric centroid of cell
% microntopix = microntopix conversion in pixels/um
% transition_thres = time (s) after Fmax to consider system @ steady state
% cat = string denoting category being analyzed
% color = plotting color a string or vector
% save_path = string pointing to save location
% fid = file ID of log file to which progress is recorded
% Fmax = radial maximum force observed for each post
% Fss = radial mean steady state force observed for each post
% t = time vector
% mpara_F = ensemble average of radial deflections * kspring (pN)

% OUTPUT:
% radialmetrics = array of metrics as a function of radial distance of post
% from cell centroid
% DRIVER/FUNCTION MAP:
% n/a (calls no subroutines)

%**************************************************************************
%**************************************************************************
%**************************************************************************
%**************************************************************************
function [radialmetrics] = PlotMetricsVsRadialDist_v7(...
    exp_date,exp_donor,exp_cond,fovn,...
    sbead_path,centroid,microntopix,...
    transition_thres,cat,color,save_path,fid,...
    Fmax,Fss,t,mpara_F)
% Get function name:
func_name = mfilename;
% Update log file that function is running:
fprintf(1,'
%s running ...
',func_name);
fprintf(fid,'
%s running ...
',func_name);

num_posts = size(Fmax,1);
% Array to hold relevant metrics:
% col1 = ID
% col2 = x coordinate of resting post in lab reference frame (um)
% col3 = y coordinate of resting post in lab reference frame (um)
% col4 = radial distance from resting post position to cell centroid (um)
% col5 = Fmax_para (pN)
% col6 = <F(tau>transition_time)_para> (pN)
% col7 = std of F(tau>transition_time)_para (pN)
% r = NaN(num_posts,7);
% xc = centroid(1)/microntopix;
% yc = centroid(2)/microntopix;
for i = 1:num_posts
    ID = Fmax(i,1);
    if(Fmax(i,1)==Fss(i,1))
        fprintf(1,'
WARNING: Post ID of row %s in Fmax = %s but in Fss = %s
',num2str(i),num2str(Fmax(i,1)),num2str(Fss(i,1)));
        fprintf(fid,'
WARNING: Post ID of row %s in Fmax = %s but in Fss = %s
',num2str(i),num2str(Fmax(i,1)),num2str(Fss(i,1)));
    end
    load([sbead_path filessep 'sbead_' num2str(ID) '.mat'],'bsec');
    xpost = bsec(1,1); % um
    ypost = bsec(1,2); % um
r(i,1) = ID;
r(i,2) = xpost; %um
r(i,3) = ypost; %um
r(i,4) = sqrt((xpost-xc)^2+(ypost-yc)^2); %um
r(i,5) = Fmax(i,2); % pN
r(i,6) = Fss(i,2); % pN
r(i,7) = Fss(i,3); % pN
end

% Report per post mean:
keep_ind = ~isnan(r(:,5));
ppm_Fmax = mean(r(keep_ind,5));
ppsd_Fmax = std(r(keep_ind,5));
ppc_Fmax = length(r(keep_ind,5));
fprintf(1,'Per post <Fmax> +/- SD (pN) = %s +/- %s, (n = %s)
',num2str(ppm_Fmax),num2str(ppsd_Fmax),num2str(ppc_Fmax));
fprintf(fid,'Per post <Fmax> +/- SD (pN) = %s +/- %s, (n = %s)
',num2str(ppm_Fmax),num2str(ppsd_Fmax),num2str(ppc_Fmax));

keep_ind = ~isnan(r(:,6));
ppm_Fss = mean(r(keep_ind,6));
ppsd_Fss = std(r(keep_ind,6));
ppc_Fss = length(r(keep_ind,6));
fprintf(1,'Per post <F(tau>%s)> +/- SD (pN) = %s +/- %s, (n = %s)
',num2str(transition_thres),num2str(ppm_Fss),num2str(ppsd_Fss),num2str(ppc_Fss));
fprintf(fid,'Per post <F(tau>%s)> +/- SD (pN) = %s +/- %s, (n = %s)
',num2str(transition_thres),num2str(ppm_Fss),num2str(ppsd_Fss),num2str(ppc_Fss));

% Plot metrics vs. radial distance without text overlay:
fig_handle_radial = figure;
subplot(1,2,1)
plot(r(:,4),r(:,5),'LineStyle','none','Color',color,'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor','k','MarkerSize',6);
hold all;
tit_str = ['Fmax vs. R,' cat];
title(tit_str,'FontSize',16,'Interpreter','Tex');
set(gca,'FontSize',12);
xlabel('Radial Distance (\mum)'),'FontSize',14);
ylabel('Fmax (pN)'),'FontSize',14);
xlim([0,max(r(:,4))]);

% Plot metrics vs. radial distance with text overlay:
fig_handle_radial2 = figure;
subplot(1,2,1)
plot(r(:,4),r(:,5),'LineStyle','none','Color',color,'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor','k','MarkerSize',6);
hold all;
tit_str = ['Fmax vs. R,' cat];
title(tit_str,'FontSize',16,'Interpreter','Tex');
set(gca,'FontSize',12);
xlabel('Radial Distance (\mum)'),'FontSize',14);
ylabel('Fmax (pN)'),'FontSize',14);
xlim([0,max(r(:,4))]);
set(gca,'FontSize',12);
xlabel('Radial Distance (\textmu m)');
ylabel('F_{max} (pN)');
xlim([0,max(r(:,4))]);

% <F(t>transition_thres) vs. R
subplot(1,2,2)
errorbar(r(:,4),r(:,6),r(:,7),'LineStyle','none',
'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,
'MarkerEdgeColor','k','MarkerSize',6);
hold all;
tit_str = [<F(tau> num2str(transition_thres) ')s)> vs. R, ' cat];
title(tit_str,'FontSize',16,'Interpreter','Tex');hold all;
set(gca,'FontSize',12);
xlabel('Radial Distance (\textmu m)');
ylabel(<F(tau> num2str(transition_thres) ')> (pN)');
xlim([0,max(r(:,4))]);

% Plot heatmaps. Because MATLAB does not allow you to associate different
% colormaps with different axes within the same figure the various metrics
% are divided among different figures unlike the previous radial plots
% which were all subplots within a single figure.

% Find spatial plotting limits:
xmin = min(floor(r(:,2)-xc))-1;
xmax = max(ceil(r(:,2)-xc))+1;
ymin = min(floor(r(:,3)-yc))-1;
ymax = max(ceil(r(:,3)-yc))+1;
if xmax>ymax
    lmax = xmax;
else
    lmax = ymax;
end
if xmin<ymin
    lmin = xmin;
else
    lmin = ymin;
end

% F_{max}
% Only generate the figure if there are at least two data points:
um_data = sum(~isnan(r(:,5)));
if num_data > 1
    fig_handle_heat_Fmax = figure;
colormap('jet');
    subplot(1,2,1);
    scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,5),'fill');
    hold all;
    tit_str = [F_{max}, ' cat];
title(tit_str,'FontSize',16,'Interpreter','Tex');
set(gca,'FontSize',12);
xlabel('x-x_c (\textmu m)');
ylabel('y-y_c (\textmu m)');
axis([lmin lmax lmin lmax]);
axis('square');
axis('ij');
caxis([min(r(:,5)) max(r(:,5))]);
c = colorbar('EastOutside');
set(c,'FontSize',12);
ylabel(c,'pN','FontSize',12);

subplot(1,2,2);
scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,5),'fill');
hold all;
text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)),'Color','k');
361 tit_str = ['IDs, Fmax, ' cat];
362 title(tit_str,'FontSize',16, 'Interpreter','Tex');
363 set(gca,'FontSize',12);
364 xlabel('x-x_c (\mum)','FontSize',14);
365 ylabel('y-y_c (\mum)','FontSize',14);
366 axis([lmin lmax lmin lmax]);
367 axis('square');
368 axis('ij');
369 caxis([min(r(:,5)) max(r(:,5))]);
370 c = colorbar('EastOutside');
371 set(c,'FontSize',12);
372 ylabel(c,'pN');
373 end
374
text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)),'Color','k');
375 tit_str = ['IDs, Fmax, ' cat];
376 title(tit_str,'FontSize',16, 'Interpreter','Tex');
377 set(gca,'FontSize',12);
378 xlabel('x-x_c (\mum)','FontSize',14);
379 ylabel('y-y_c (\mum)','FontSize',14);
380 axis([lmin lmax lmin lmax]);
381 axis('square');
382 axis('ij');
383 caxis([min(r(:,5)) max(r(:,5))]);
384 c = colorbar('EastOutside');
385 set(c,'FontSize',12);
386 ylabel(c,'pN');
387
text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)),'Color','k');
388 tit_str = ['IDs, Fmax, ' cat];
389 title(tit_str,'FontSize',16, 'Interpreter','Tex');
390 set(gca,'FontSize',12);
391 xlabel('x-x_c (\mum)','FontSize',14);
392 ylabel('y-y_c (\mum)','FontSize',14);
393 axis([lmin lmax lmin lmax]);
394 axis('square');
395 axis('ij');
396 caxis([min(r(:,5)) max(r(:,5))]);
397 c = colorbar('EastOutside');
398 set(c,'FontSize',12);
399 ylabel(c,'pN');
400 end
401
text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)),'Color','k');
402 tit_str = ['IDs, Fmax, ' cat];
403 title(tit_str,'FontSize',16, 'Interpreter','Tex');
404 set(gca,'FontSize',12);
405 xlabel('x-x_c (\mum)','FontSize',14);
406 ylabel('y-y_c (\mum)','FontSize',14);
407 axis([lmin lmax lmin lmax]);
408 axis('square');
409 axis('ij');
410 caxis([min(r(:,5)) max(r(:,5))]);
411 c = colorbar('EastOutside');
412 set(c,'FontSize',12);
413 ylabel(c,'pN');
414 end
415
416 % <F(t>transition_thres)>
417 % Only generate the figure if there are at least two data points:
418 num_data = sum(~isnan(r(:,6)));
419 if num_data > 1
420 fig_handle_heat_meanFss = figure;
421 colormap(flipud(colormap('jet')));
422 subplot(1,2,1);%<F(t>transition_thres)>
423 scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,6),'fill');
424 hold all;
425 tit_str = ['<F(t>' num2str(transition_thres) 's), ' cat];
426 title(tit_str,'FontSize',16, 'Interpreter','Tex');
427 set(gca,'FontSize',12);
428 xlabel('x-x_c (\mum)','FontSize',14);
429 ylabel('y-y_c (\mum)','FontSize',14);
430 axis([lmin lmax lmin lmax]);
431 axis('square');
432 axis('ij');
433 caxis([min(r(:,6)) max(r(:,6))]);
434 c = colorbar('EastOutside');
435 set(c,'FontSize',12);
436 ylabel(c,'pN');
437 subplot(1,2,2);
438 scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,6),'fill');
439 hold all;
440 text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)),'Color','k');
441 tit_str = ['IDs, <F(t>' num2str(transition_thres) 's), ' cat];
442 title(tit_str,'FontSize',16, 'Interpreter','Tex');
443 set(gca,'FontSize',12);
444 xlabel('x-x_c (\mum)','FontSize',14);
445 ylabel('y-y_c (\mum)','FontSize',14);
446 axis([lmin lmax lmin lmax]);
447 axis('square');
448 axis('ij');
449 caxis([min(r(:,6)) max(r(:,6))]);
450 c = colorbar('EastOutside');
451 set(c,'FontSize',12);
452 ylabel(c,'pN');
453 end
454
455 % Compute FWHM for this category's ensemble average curve:
456 width = FWHM_NoDisplay_PosOnly(t,mpara_F); % s or NaN
457 fprintf(1,'	Ensemble FWHM (s) = %s, (n = %s)
',num2str(width),num2str(ppc_Fmax));
458 fprintf(fid,'	Ensemble FWHM (s) = %s, (n = %s)
',num2str(width),num2str(ppc_Fmax));
459
460 % Record values in an excel spreadsheet for easier compiling across
461 % multiple experiments:
462 fprintf(1,'
	Saving mean metrics in Excel spreadsheet..
');
% Steven J. Henry
1 % 02/17/2015
2 %**************************************************************************
3 % PURPOSE: To plot force vs time for each post vertically shifting each
4 % force trajectory so they are individually distinguishable.
5 %
6 % ASSUMPTIONS:
7 % n/a
8 %
9 % INPUT:
10 % rsbead_path = path to rsbead.mat files for category 'cat'
11 % radialmetrics = array of metrics as a function of radial distance of post
12 % from cell centroid
13 % kspring = post sprint constant (pN/nm)
14 % time_int = time interval between frames (sec)
15 % cat = string denoting category being analyzed
16 % color = string or vector to specify plotting color
17 % save_path = path to location where analysis is being saved
18 % fid = logfile handle
19 %
20 % OUTPUT:
21 % n/a
22 %
23 % DRIVER/FUNCTION MAP:
24 % n/a (calls no subroutines)
25 %**************************************************************************
26 function [] = Plot_fvst_Strips_v1(rsbead_path,radial_metrics,...
kspring, time_int, cat, color, save_path, fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'n%s running ...
', func_name);
fprintf(fid,'n%s running ...
', func_name);

[~, sort_ind] = sort(radial_metrics(:, 4), 'ascend');
ID = radial_metrics(sort_ind, 1);
num_posts = length(ID);

fig_handle_strip = figure;
tick = 0;
step = 150;
ticks = NaN(num_posts, 1);
labels = cell(num_posts, 1);

for i = 1:num_posts
    load([rsbead_path filesep 'rsbead_' num2str(ID(i)) '.mat'], 'bsec');
    rpara = bsec(:, 1) * 1000;  % nm
    rperp = bsec(:, 2) * 1000;  % nm
    frames = bsec(:, 3);
    % Exclude row 1 which contains the distance to the resting lattice
    % position from the resting lattice position (i.e. 0).
    rpara(1) = []; 
    rperp(1) = []; 
    frames(1) = [];

t = frames * time_int;
ticks(i) = tick;

s1 = subplot(1, 2, 1);
plot(t, rpara * kspring + tick, 'LineStyle', '-', 'Color', color, ...
    'Marker', 'none');
hold all;

s2 = subplot(1, 2, 2);
plot(t, rperp * kspring + tick, 'LineStyle', '-', 'Color', color, ...
    'Marker', 'none');
hold all;

tick = tick + step;
labels{i} = ['post ' num2str(ID(i))];

end

subplot(s1)
title(['f(t)_I_I_ (pN), ' cat], 'FontSize', 20, 'Interpreter', 'Tex');
set(s1, 'FontSize', 14);
xlabel('time (s)', 'FontSize', 16);
ylabel('Radial distance from centroid (\mu m)', 'FontSize', 16);
set(s1, 'YTick', ticks);
set(s1, 'YTickLabel', num2str(radial_metrics(sort_ind, 4)), 'FontSize', 8);
legend(labels);
legend('hide');

subplot(s2)
title(['f(t)_\perp (pN), ' cat], 'FontSize', 20, 'Interpreter', 'Tex');
set(s2, 'FontSize', 14);
xlabel('time (s)', 'FontSize', 16);
ylabel('Radial distance from centroid (\mu m)', 'FontSize', 16);
IndividualPostAutoCorrelation_v2.m

% Steven J. Henry
% 10/06/2014
%**************************************************************************
% PURPOSE: This function performs autocorrelations of individual posts.
%**************************************************************************

function [] = IndividualPostAutocorrelation_v2(rsbead_path,subsetIDs,...
    max_num_frames,time_int,cat,color,save_path,fid)

% Get function name:
func_name = mfilename;

% Update log file that function is running:
fprintf(1,'%n%s running ...
',func_name);
fprintf(fid,'%n%s running ...
',func_name);

% How many post files did user select?
num_posts = length(subsetIDs);

% Initialize plot that will hold all autocorrelation functions:
fig_handle_corr = figure;
subplot(1,2,1);
xlabel('\tau (s)','FontSize', 14,'Interpreter','Tex');
ylabel('Normalized Autocorrelation','FontSize', 14,'Interpreter','Tex');
title(['Individual r|\tau|_{|_|}, ' cat],'FontSize',16,'Interpreter','Tex');
hold all;

% Specify plotting color:
color_list = {'r','b','m','k','g'};

% Loop through post files:
for i = 1:num_posts
    % Load post file:
    load(fullfile(rsbead_path,subsetIDs(i),'.mat'));

    % Select relevant variables:
    data = dataset(fullfile(rsbead_path,subsetIDs(i),'.mat'));
    x = data(:,1);
    y = data(:,2);

    % Calculate autocorrelation:
    [r,tau] = autocorr(x);

    % Plot autocorrelation:
    subplot(1,2,1);
    plot(tau, r, color_list{i});
    hold all;

    % Save figures:
    fprintf(1,'%n	Saving variable(s) and figure(s)
');
    fprintf(fid,'%n	Saving variable(s) and figure(s)
');
    saveas(fig_handle_corr,[save_path filesep 'fvst_strip_' cat '.fig']);

    % Update log file that function is completed:
    fprintf(1,'%n%s completed
',func_name);
    fprintf(fid,'%n%s completed
',func_name);
end
hold all;
labels = cell(num_posts,1);

max_length = 2*max_num_frames-1;
C_para = NaN(max_length,num_posts);
C_perp = C_para;

for i = 1:num_posts
    load(fullfile('rsbead_' num2str(subsetIDs(i)) '.mat',...'
'obj'),...
'vars','bsec');
    r_para = bsec(:,1); %#ok<NODEF> % um
    r_perp = bsec(:,2); % um
    % Exclude row 1 which contains the distance to the resting lattice
    % position from the resting lattice position (i.e. 0).
    r_para(1) = [];
    r_perp(1) = [];
    [c_para, lags_para] = xcorr(r_para,'coeff');
    [c_perp, lags_perp] = xcorr(r_perp,'coeff');
    % These fancy indices are to ensure that lag = 0 row is aligned for all
    % autocorrelations to be averaged:
    L = length(c_para);
    half_pad = (max_length - L)/2;
    C_para(half_pad+1:half_pad+L,i) = c_para;
    C_perp(half_pad+1:half_pad+L,i) = c_perp;
    subplot(1,2,1)
    plot(lags_para*time_int,c_para,'LineStyle','-','Color',color);
    hold all;
    subplot(1,2,2)
    plot(lags_perp*time_int,c_perp,'LineStyle','-','Color',color);
    hold all;
    labels{i} = ['post' num2str(subsetIDs(i))];
end

m_para = NaN(max_length,1);
sd_para = m_para;
m_perp = m_para;
sd_perp = m_para;

figure(fig_handle_corr)
subplot(1,2,1)
plot(lags_para*time_int,c_para,'LineStyle','-','Color',color);
hold all;
subplot(1,2,2)
plot(lags_perp*time_int,c_perp,'LineStyle','-','Color',color);
hold all;
labels{i} = ['post' num2str(subsetIDs(i))];
end
figure(fig_handle_corr)
subplot(1,2,1)
legend(labels);
legend('hide');
subplot(1,2,2)
legend(labels);
legend('hide');

m_para = NaN(max_length,1);
sd_para = m_para;
m_perp = m_para;
sd_perp = m_para;

figure(fig_handle_mean)
subplot(1,2,1)
plot(lags_para*time_int,c_para,'LineStyle','-','Color',color);
hold all;
subplot(1,2,2)
plot(lags_perp*time_int,c_perp,'LineStyle','-','Color',color);
hold all;
labels{i} = ['post' num2str(subsetIDs(i))];
'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',6);
xlabel('	au (s)','FontSize', 14,'Interpreter','Tex');
ylabel('Mean Normalized Autocorrelation','FontSize', 14,...
'Interpreter','Tex');
title(['Mean r(tau)_||, ' cat]','FontSize',16,'Interpreter','Tex');

subplot(1,2,2);
plot(m_lags,m_perp,'LineStyle','none','Color',color,'Marker','o',
'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',6);
xlabel('	au (s)','FontSize', 14,'Interpreter','Tex');
ylabel('Mean Normalized Autocorrelation','FontSize', 14,...
'Interpreter','Tex');
title(['Mean r(tau)_\perp, ' cat]','FontSize',16,'Interpreter','Tex');

fig_handle_mean_error = figure;
subplot(1,2,1);
errorbar(m_lags,m_perp,LineStyle,'none','Color',color,'Marker','o',
'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',6);
xlabel('	au (s)','FontSize', 14,'Interpreter','Tex');
ylabel('Mean Normalized Autocorrelation \pm SD','FontSize', 14,...
'Interpreter','Tex');
title(['Mean r(t)_||, ' cat]','FontSize',16,'Interpreter','Tex');

figure();
errorbar(m_lags,m_perp,LineStyle,'none',
'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,....
'MarkerEdgeColor',color,'MarkerSize',6);
xlabel('tau (s)', 'FontSize', 14,'Interpreter','Tex');
ylabel('Mean Normalized Autocorrelation \pm SD','FontSize', 14,...
'Interpreter','Tex');
title(['Mean r(t)_\perp, ' cat]','FontSize',16,'Interpreter','Tex');

% Find tau (s) at which mean normalized autocorrelation = 1/e (~0.3679)
mid = (length(m_para)-1)/2+1;
m_para_trunc = m_para(mid:end);
m_perp_trunc = m_perp(mid:end);
m_lags_trunc = m_lags(mid:end);
[-row_para] = min(abs(m_para_trunc-1/exp(1)));
characteristic_tau_para = m_lags_trunc(row_para);

% Record values in an excel spreadsheet for easier compiling across
% multiple experiments:
fwrite(1,'
	Saving mean metrics in Excel spreadsheet..
');
fwrite(fid,'
	Saving mean metrics in Excel spreadsheet..
');

header_info = {
	cat ' tau@1/e para (s)
	cat ' tau@1/e perp (s)
};
xls_name = [save_path filesep 'MeanMetrics.xlsx'];
xlswrite(xls_name,header_info,cat,'Q1');
data_to_log = {characteristic_tau_para,characteristic_tau_perp};
xlswrite(xls_name,data_to_log,cat,'Q2');

% Save figures:
fwrite(1,'
	Saving variable(s) and figure(s)
');
fwrite(fid,'
	Saving variable(s) and figure(s)
');
save([save_path filesep 'Autocorrelation_Variables_' cat '.mat'],...
'Autocorrelation_AllTrajectories_' cat '.fig');
saveas(fig_handle_corr,[save_path filesep...
'Autocorrelation_AllTrajectories_' cat '.fig']);
saveas(fig_handle_mean,[save_path filesep ...
'Autocorrelation_Mean_' cat '.fig']);
saveas(fig_handle_mean_error,[save_path filesep ...
'Autocorrelation_Mean_Errorbars_' cat '.fig']);

% Update log file that function is completed:
fwrite(1,'%s completed
',func_name);

fclose(fid);
function [] = Tidy_Up_v2(logfile)
dir_info = dir(pwd);
num_entities = length(dir_info);
um_files = 0;
for i = 1:num_entities
    if dir_info(i).isdir == 0
        num_files = num_files+1;
    end
end
extensions = cell(num_files,1);
k = 1;
for i = 1:num_entities
    if dir_info(i).isdir == 0
        extensions(k,1) = dir_info(i).name(end-3:end);
k = k+1;
end
unique_extensions = unique(extensions);
num_unique = size(unique_extensions,1);
for i = 1:num_unique
    ext = unique_extensions(i);
    ext_path = [pwd filesep ext(end-2:end)];
    mkdir(ext_path);
    movefile(['*' ext],ext_path);
    pause(3);
% If you're moving .txt files undo logfile move:
if strcmp(ext,'.txt')==1
    % Go into txt folder
    cd(ext_path);
    % Move logfile up one level
    movefile(logfile,'..');
    % Reset directory up one level
    cd('..');
end
end
end
References

