2021

Learning Representations For Matching

Stephen Phillips
University of Pennsylvania

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

Part of the Computer Sciences Commons

Recommended Citation
https://repository.upenn.edu/edissertations/3894

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/edissertations/3894
For more information, please contact repository@pobox.upenn.edu.
Learning Representations For Matching

Abstract
Matching is an old and fundamental problem in Computer Vision. Ranging from low level feature matching for extracting the geometry of a scene to high level semantic matching for scene understanding, there is a broad scope of applications to the matching problem. However, there are many challenges, such as noise and outliers, that make the problem especially difficult. Recent work has shown that using multiple images improves matching performance over pairwise matches. Additionally, in recent years, deep learning has shown great promise in Computer Vision. Deep learning techniques are state of the art in object detection, segmentation, and image generation. Deep learning techniques excel at feature learning, and prior distribution learning implicitly helps them to achieve state of the art. We hope to leverage this power to learn better representations for matching problems. In this work we propose to use various deep learning techniques to learn better matches by learning better feature representations for match. We use graph neural networks to handle the sparse nature of many of these matching problems, using multi-image cycle consistency and geometric consistency losses to learn robust representations. We propose a framework for handling outlier rejection in training the deep neural networks using primal-dual optimization. We will apply these techniques to Structure from Motion sub-problems (such as two-view or multi-view matching), shape and point cloud matching.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Kostas Daniilidis

Subject Categories
Computer Sciences

This dissertation is available at ScholarlyCommons: https://repository.upenn.edu/edissertations/3894
LEARNING REPRESENTATIONS FOR MATCHING

Stephen Phillips

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Supervisor of Dissertation

Kostas Daniilidis, Professor, Computer and Information Science

Graduate Group Chairperson

Mayur Naik, Professor of Computer and Information Science

Dissertation Committee

Jean Gallier, Professor, Computer and Information Science
Jianbo Shi, Professor, Computer and Information Science
Alejandro Ribeiro, Professor, Electrical and Systems Engineering
Luca Carlone, Professor, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology
I am grateful for the kindness, wisdom, and support that Kostas has given throughout my journey here. I have immensely enjoyed the lab he has built, and the people I have met through it. I have grown much as a researcher and teacher here at Penn and I am grateful for all the opportunities that Kostas and Penn have given me. I deeply appreciate Kostas for welcoming me to Penn and making it all possible.

I also want to thank Jean Gallier, for the wonderful conversations we have had. I have enjoyed taking his classes and mini-lectures as well as helping him teach his classes. I give many thanks for the fun and enjoyment throughout the learning process and I hope to keep hearing more of Jean’s insights in the future.

I also want to thank my committee members Jianbo Shi, Alejandro Ribeiro, and Luca Carlone, for the inspiration and direction they have given me. I am grateful for the understanding and flexibility for this dissertation process. Their work has inspired me and I strive to be a better research because of it.

I also want to thank all the members of Kostas’ research group. I will always remember fondly the conversations we had over group meetings, lunch, and conferences. They were all a key part of my growth as a researcher. I appreciate Alex Zhu and George Pavlakos for navigating the Ph.D. with me. I particularly want to thank my co-author on my early papers Drew Jaegle for the fruitful conversations and work we did together. I also want to thank my research collaborators Daphne Ippolito, Kenneth Chaney, Bernadette Bucher, and Santiago Paternain. The interdisciplinary work I have done with William Schmenner has inspired me, and I appreciate the new perspectives I have learned from him.

Finally I want to thank my friends and family throughout my Ph.D. My life has been happier thanks to their support and love, and I cannot imagine my Ph.D. without them.
ABSTRACT

LEARNING REPRESENTATIONS FOR MATCHING
Stephen Phillips
Kostas Daniilidis

Matching is an old and fundamental problem in Computer Vision. Ranging from low level feature matching for extracting the geometry of a scene to high level semantic matching for scene understanding, there is a broad scope of applications to the matching problem. However, there are many challenges, such as noise and outliers, that make the problem especially difficult. Recent work has shown that using multiple images improves matching performance over pairwise matches.

Additionally, in recent years, deep learning has shown great promise in Computer Vision. Deep learning techniques are state of the art in object detection, segmentation, and image generation. Deep learning techniques excel at feature learning, and prior distribution learning implicitly helps them to achieve state of the art. We hope to leverage this power to learn better representations for matching problems.

In this work we propose to use various deep learning techniques to learn better matches by learning better feature representations for match. We use graph neural networks to handle the sparse nature of many of these matching problems, using multi-image cycle consistency and geometric consistency losses to learn robust representations. We propose a framework for handling outlier rejection in training the deep neural networks using primal-dual optimization. We will apply these techniques to Structure from Motion sub-problems (such as two-view or multi-view matching), shape and point cloud matching.
CONTENTS

1 INTRODUCTION 1
   1.1 Matching Methods 3
      1.1.1 Classical Matching Methods 4
      1.1.2 Robust Matching 6
   1.2 Representations for Matching in Images 9
   1.3 Deep Learning for Matching 11
   1.4 Goals of this Work 14

2 FAST, ROBUST, CONTINUOUS MONOCULAR EGO-MOTION COMPUTATION 15
   2.1 Motivation 17
   2.2 Background 19
      2.2.1 Egomotion/visual odometry 19
      2.2.2 Continuous, monocular approaches 20
      2.2.3 Robust optimization 21
   2.3 Problem formulation and approach 22
      2.3.1 Visual egomotion computation and the motion field 23
      2.3.2 Robust Formulations 35
      2.3.3 Robust estimation using a lifted kernel 40
   2.4 Experiments 42
      2.4.1 Goodness of Fit for Laplacian Distribution 44
      2.4.2 Comparison Of Difference ERL Likelihood Schemes 45
      2.4.3 Evaluation on KITTI 45
      2.4.4 Synthetic sequences 47
   2.5 Conclusions 49

3 LEARNING GEOMETRIC AND CYCLE-CONSISTENT REPRESENTATIONS WITH GRAPH
   CONVOLUTIONAL NETWORKS 50
   3.1 Introduction 51
   3.2 Background 53
      3.2.1 Feature Matching 53
      3.2.2 Multi-image Matching 54
LIST OF TABLES 116
LIST OF ILLUSTRATIONS 117
BIBLIOGRAPHY 123
Computer vision is an important piece in making autonomous systems, as cameras are one of the most rich and robust sensor modalities we know. The first work of computer vision began back in 1966 with the “Summer Vision Project” [99]. Finding correspondences between objects or images, known as matching, is an old and fundamental problem in computer vision, dating to some of the earliest days in the field. The earliest applications of matching were mainly for geometric computer vision, such as stereo matching as shown by Hannah [49]. Matching is still relevant today, where it is used in optical flow [64], two view matching [142], multi-image matching [79], point cloud matching [137], semantic matching [131], semantic SLAM [10], and more.

As this has been an active field of research in computer vision literature, what about matching makes it so difficult? While the motion field equations govern the motion of each point projection in an image for small inter-image baseline [67], there are many ambiguities when dealing with actual images, like the aperture problem [115]. When dealing with larger baselines, you run into large appearance variation and occlusions, thus features of interest are difficult to match. If attempting for higher level semantic matches, then the appearance of each component to match is completely unreliable, and thus more robust features capturing the relationship between the parts are needed. Often you have to deal with large color and light variation between images being matched e.g. matching a building on images taken at day and night. Due to these problems, even when you are
Figure 1.1: Illustration of the many variations and applications of the matching problem. Matching can be applied to low level geometric problems as in two-view and multi-view feature matching or optical flow. Alternatively, matching can be applied to semantic matching for high level analysis of images. All this can come together for applications such as semantic SLAM. Figures taken from [64, 142, 79, 131, 10].

able to obtain matches they often have high amounts of noise or matches between features outside of the assumed model i.e. outliers.

Over the last 50 years the computer vision community has had to find ways to address these problems. Traditional methods have created hand crafted features that are partially invariant to the variations above. Most of the methods were applied to low level or geometric matching. However, most of the methods can be or have been applied to semantic matching if sufficiently good features have been obtained (normally DNN features). In this thesis, we hope not only to gain a better understanding of matching, but also to construct better representations for the purposes of matching. We focus on looking at how matching is done once given such representations, and then from that understand the desirable properties that feature representations used for matching would need.
Matching Task | Description
--- | ---
Optical Flow | Matching pixels in one image to pixels in another. Assumed brightness constancy and for images to have been taken close to each other in space.
Two-view Matching | Matching pixels or points in one image to pixels/points in another. Only assumption is that the views overlap - wide baselines, lighting variation, etc. all possible. Often used to find Rigid Transformation between the images.
Multi-view Matching | Matching pixels or points in multiple images to each another. Thus one point has matches in many images. Same assumptions as two view matching, but each view only has to overlap with one other image. Often used to find the pose graph of all the images.
Point Cloud Matching | Matching 3D points from several point clouds taken from the same object from different perspectives or sensors. One may only be a very small part of the other point cloud.
Mesh Alignment | Matching two meshes together - most often done via graph matching over meshes or functional maps.
Semantic Matching | Matching human defined features from one image/mesh to another. Most common examples is part matching on humans, animals, or vehicles.
(Semantic) SLAM | Simultaneous Localization and Mapping - finding position of sensors and building a map of areas traversed. More encompassing that just matching but data association is an integral part. Semantic SLAM incorporates human defined objects into the map.

Table 1.1: Table of the different types of matching problems

1.1 MATCHING METHODS

What is matching? The most encompassing definition would be given two sets of objects, find a one-to-one mapping between them. The objects could be low level descriptions of points on the image, vertices on a graph, vertices on a 3D mesh, point in a point cloud, or high level ideas of parts of an object (such as human joints). While for a general definition
of matching the objects do not matter, in practical problems the objects affect the choice of
algorithm one uses. Thus one needs a wide range of matching algorithms to determine
which one will give the best performance on their specific problem.

1.1.1 Classical Matching Methods

The simplest approach to solve the matching problem would be brute force matching.
However, these methods take exponential time in the worst case and are not efficient in
practice. A better approach would be branch and bound methods [37] however these still
are not generally efficient in the worst case scenario. One can speed such computations us-
ing dynamic programming. The standard example of dynamic programming in computer
vision is stereo matching [88], where the structure of the camera positions constrains the
potential matches. The classical approach is to use dynamic programming across the rows
(which correspond to the epipolar lines) in order to enforce ordering constraints, making
for more efficient matching.

If the particular matching problem being faced can be formulated as a bipartite graph
matching problem, one can use the Hungarian Algorithm [77], one of the oldest methods for
finding matches. This method given a square $n \times n$ cost matrix finds the permutation that
maximizes the score (or equivalently minimizes the cost) of the sum of entries chosen by
the permutation. The algorithm achieves this by marking the minimum values of rows and
columns and adjusting accordingly when there are multiple minimal values in the same
column. Another way of looking at the algorithm is that it is a variation on the simplex
algorithm with the dual variables implicitly being optimized. It is still in wide use today
as a final step to refine the matches of other algorithms [156, 79].

In many image matching scenarios, relationships between objects within the same image
(intra-image matches) can be helpful in understanding matches between images (inter-
image matches). The classical approach to handle this is Graph Matching, where the features are nodes in the graph and intra- and inter-image similarities are given as attributed edges. Given two graphs with edge attributes $G_1 = (V_1, E_1, A_1)$ and $G_2 = (V_2, E_2, A_2)$, the objective of graph matching is to find the best mapping of the nodes $M : V_1 \to V_2$ that best preserves the edge attributes according to some loss $\sum_{e_1 \in E_1, e_2 \in E_2} \ell(A_1(e_1), A_2(e_2))$. This is often formulated as an Integer Quadratic Program (IQP) \[27, 120, 140\]. Graph matching has been used for face identification \[134\], object matching \[140\], person re-identification \[66\] and more. The main differences of the works are in how they modify the original IQP, such as relaxing the integer constraint \[27\] or adding a regularizer to the matches \[120\].

(a) Bipartite matching for the Hungarian Algorithm

(b) Graph matching for face matching (Figure from \[134\])

(c) Matching painting an images (Figure from \[4\])

Figure 1.2

Related to above is the notion of Spectral Matching. If the cost function of a particular matching problem can be formulated as a quadratic loss of some kind (as in the relaxed IQP problem above) then you can find the solution using the eigendecomposition. For
instance in two-way segmentation, normalized cuts [114] reformulated segmentation as a Rayleigh quotient problem i.e. a specialized eigenvalue problem, and follow-up work generalized it for multiple classes [144]. Eigenvalue based matching is fairly common, and can be used as a relaxation for graph matching [18], or solving for motion given a set of matches [23]. Even more dramatically, it can be used to match extremely visually disparate images using intra- and inter-image feature matches [4] with high accuracy. In all these cases, you take the matches and form them into a matrix, and with the help of some post-processing or additional regularizers, find the solution by extracting the eigenvalues from that matrix.

### 1.1.2 Robust Matching

In all of the discussed methods, we implicitly assumed good matching scores and measurements with some significant but manageable amount of noise. However, not all matches given have nice small levels of noise, and some lie completely outside the distribution of any model we are fitting. These matches, known as outliers, need to be treated separately from simply noisy points, or else any algorithm run on top of the matches will fail catastrophically. To handle these outliers, an understanding of the ‘correct’ distribution to fit to is needed, or in other words one needs to know what distribution a match is an outlier from. Thus you no longer are dealing with matches and matching scores alone but also a model that you want to fit and a noise model of what is considered ‘inlier noise’. In all examples discussed here, this model has parameters that we want to fit and a way of quantifying how bad a match fits our model (typically a log-likelihood score).

A classic method for handling outliers is the Hough Transform, which transforms the points in matching space to votes in parameter space [65]. First you create a discretized version of parameter space (normally a tensor of some kind), then you take each match and
add 1 to each point in the parameter space where a model could fit it, a ‘vote’. Once you do this you pick the parameter point with the most votes as your model. While this method has several advantages, such as robustness to a large number of uncorrelated outliers and the ability to handle multiple model fitting, it suffers from memory requirements exponential in the number of model parameters. This weakness means for all but the simplest of models, such as lines and circles, this method is impractical.

In practical settings, the gold standard for dealing with outliers in computer vision is Random Sample Consensus or RANSAC [33]. The algorithm is actually quite simple: pick uniformly at random the minimum number of matches required for fitting parameters to the model, check the number of points within threshold $\tau$ of that (denoted the inliers) and repeat for a certain number of steps. Then pick the model parameters with the largest number of inliers and refit the model with all the inliers of those parameter. While this seems overly simplistic, it works remarkably well in practice [111, 35]. Improvements can be made on top of RANSAC, for instance using a likelihood function instead of a simple inlier threshold [125], but the essence stays the same. As this algorithm is random, the probability of success $p_{\text{succ}}$ depends on the probability a point is an inlier $p_i$, the number of parameters in the model $n$, and number of iterations $k$, in the relation $p_{\text{succ}} = 1 - (1 - p_i^n)^k$. Thus for a desired probability of success (typically around 99%) you would need at least $k = \log(1 - p_{\text{succ}}) / \log(1 - p_i^n)$ iterations. Thus for a fixed probability of success, RANSAC’s performance would improve if you increase the proportion of inliers or decrease the number of model parameters, which is why much work has been done to find minimal solvers [42]. The downside of RANSAC comes from its dependence on the success probability and a minimal solver.

If the problem being dealt with does not have a minimal solver with a small number of parameters then RANSAC is ineffective, and, thus, robust global solvers are needed. The simplest approach to this is to optimize some convex loss that does not penalize heavy
tails, such as the $\ell_1$ loss or Huber loss [11], thus guaranteeing convergence. One of the simplest methods for robust estimation is the Iteratively Re-weighted Least Squares (IRLS) method for optimizing robust kernels [57]. However better models of robustness require non-convex losses and thus we do not have guarantees of a good solution in any attempt to optimize these directly. One way to address this issue is to 'lift' the problem into a higher dimensional space by adding weights representing the quality of each match then jointly optimizing for the quality of the matches and the parameters of the model being used. This can be used for non-linear least squares optimization [145, 67], graduated non-convex optimization [136], or semi-definite and moment-relaxations [139]. Work discussed later in Chapters 2 and 4 fits in this regime. Alternatively one can skip using weights and optimize over sets of matches directly while testing model fit [17, 128]. Voting schemes (or convex relaxations of voting schemes) can also be used [137] if one is careful about formulating the problem to avoid the exponential blowup in parameter space.

There are more topics to be covered in matching, which will be discussed in more depth in later sections. For instance, deep learning more recently has been used for matching in computer vision, but this will be covered in Section 1.3. And for computer vision in particular, there is also the problem of multi-image matching. Having multiple views means combinatorially more matches than in the two view case and more data to handle at once. But while in some ways it is more challenging than standard two view matching, it has the potential for more robustness as there are more observations per point. One can solve the combinatorial problem using a latent variable model, to get both speed and robustness. This will be covered in greater detail in Chapter 3.
1.2 REPRESENTATIONS FOR MATCHING IN IMAGES

In the methods discussed thus far we assume some initial matches are given with some type of matching scores, but there has been no discussion as to where these matches come from. The answer to this question has been a subject of intense research for many decades for computer vision, and many other fields besides. For computer vision in particular, it has several facets:

- What parts of an image should match?
- How should we quantify the quality of a match?
- How should we create the initial matches?

Each of these has their own answer and years of research behind it.

For the matter of what parts of an image should we match, the real question is one of distinguishability. If the objects to match were perfectly distinguishable then the matching would be as trivial as iterating through the objects. Alas this is not the case in computer vision, and so we need to find measures of distinguishability. Specifically, if an algorithm were given two patches in an image, would it be able to distinguish them and how well would it be able to do so? For small image patches, the answer relies on the variability of image gradients of that patch (‘texturedness’ or ‘cornerness’). If there is no texture at all, then it is a patch of uniform color and is indistinguishable from any other patch of uniform color. Similarly, if the texture only changes in one direction then the patch can only be distinguished moving in that direction. These two cases are known as the aperture problem, as if you looked at the world from a small aperture you would not be able to properly discern their motion. To determine the ‘texturedness’ computationally, one takes the eigenvalues of the second order moment matrix \[50\] or some function thereof.
then determines how textured the feature is by taking eigenvalues. This can be done hierarchically, and is a standard for tracking via optical flow [87]. Some more recent methods use binary checks for computational speed up [107]. This methodology can even be applied to different sensor modalities, such as event based cameras [157].

At a higher level, consider if one saw an object partially occluded. When would one be able to determine its identity? When would the occlusion be too much to handle? As of yet, we do not know how to automatically answer this question. However, given human annotated parts, one can find good correspondences for many different kinds of objects. For instance, it is possible to find the 6DoF pose of the object using deep learning [100], or find accurate 3D human pose [102]. Another standard technique is to use functional maps to find correspondences, most often used on meshes [97].

Once we have distinguishable patches, how should we quantify the quality of matches? The answer to this question is related to the previous question - we need some description of the points based on their distinguishable characteristics. Which characteristics to use and how to compute them is another area of intense research. If the differences between the patches is small enough, the patches found from the ‘texturedness’ criterion would be features in of themselves. However, due to the issues mentioned in the introduction (large viewpoint variation, lighting changes, etc.) the patch itself can be quite brittle. In an Information Theoretic sense the main purpose of these descriptors is to keep the information relevant to the task at hand (in this case matching) and to discard everything else. Thus, our desiderata are descriptors of the patches - some representation invariant to the nuisance variability unimportant to our task [2] such as the aforementioned viewpoint variations and lighting changes. The most famous of these is the Scale Invariant Feature Transform, or SIFT, which is designed to be invariant to lighting changes and planar rotations, translations, and scaling transformations [86]. Many similar descriptors followed, such as the Speeded up robust feature SURF [6], or the oriented binary descriptor ORB.
After computing the descriptor of choice, one can compute an appropriate distance function between the descriptors to determine the quality of the match. For instance, for SIFT an $L_2$ distance or cosine similarity will do, while for binary descriptors like ORB one needs to use the Hamming distance. In more recent works, deep learning has also been used to compute descriptors but that will be addressed in Section 1.3.

Having answered this question the third question, how to pick the initial matches, becomes much easier. We simply pick the matches with highest matching score, and cut off after a certain number or some score threshold. While this works adequately, one can improve matching if one has extra information e.g. knowing the direction of the gravity vector [35] or knowledge of intra-image matches (like in graph matches). In Section 1.1.2 we also discussed various methods to help reject egregiously bad matches. However, are there ways we can make the initial matches themselves better? Deep learning seems to hold promise in answering that.

1.3 DEEP LEARNING FOR MATCHING

Having given some answer for both the question of how to match points given putative matches and the finding of said putative matches, we shift topics to address deep learning. Deep learning and Deep Neural Networks (DNNs) have been shown to generate robust and generalizable features for a wide variety of computer vision tasks [113]. Deep learning in computer vision has been used to gain significant improvements in image recognition [76], object detection [105], and segmentation [84], among other tasks. Many new architectures have come out improving performance [53], as well as new techniques for generating samples [47], predicting video frames [90]. Needless to say deep learning has enjoyed wild success across many areas of computer vision.
Figure 1.3: (a) The first deep learning submission to the ImageNet [109] competition, AlexNet [76], dramatically improved results compared to the non-Deep Learning counterparts, nearly by a factor of 2. All winners of ImageNet since then have used deep Learning. (b) The percentage of papers with deep learning or related techniques (e.g. LSTMs) explicitly in the title has dramatically increased in recent years. This is not including papers using deep learning without mentioning it in the title. (c) The number of papers submitted to the major Computer Vision conferences has been rising steadily since the increase in popularity in deep learning.

Naturally there has been plenty of work in deep learning for matching. For instance, in recent years all state of the art optical flow has been done using deep learning [64, 83]. Human pose annotation has also achieved great success using deep learning, able to extract 3D pose from 2D images with high accuracy [101, 102, 75]. DNNs are state of the art in semantic feature extraction and detection [121, 100]. DNN features have been used for the initial features in multi-image semantic matching as well [131].

However, there has been less work on large baseline matching using DNNs. A large factor of this is the lack of easily available datasets, as finding ground truth correspondences
over large baselines can be quite challenging. One strategy is to avoid all labeling and matching tasks and use unsupervised losses for learning depth and pose from images [80]. However if one needs correspondences for a downstream task such as map building in Simultaneous Localization and Mapping (SLAM), we cannot avoid the matching task.

Much of the previous attempts to improve image matching techniques using machine learning have focused on learning the descriptors given ground truth correspondence from curated datasets. LIFT by Yi et al [141] takes the simplest approach to use a neural network to replace standard feature descriptors such as SIFT. Alternatively, Zagoruyko et al [148] use patches as input and output a matching score, avoiding thus the need for descriptors. However, there is little evidence that using simple DNN features as descriptors in the standard structure from motion pipeline is not enough to the current descriptors [154]. However, research into the effectiveness of DNN based descriptors compared to classical descriptors are mixed [154, 142].

Applying DNNs to multi-view and dense matching has also been explored. Hartmann et al [52] use ground truth correspondences to train a network to handle n-patch similarity. They extract correspondences from multiple views for the purposes of depth estimation of a scene. DNNs are typically formulated as dense pixel-wise feature extractors, and thus dense feature matching has been explored by various authors. For instance, Choy et al [24] takes in point matches as training and uses a fully convolutional network to learn feature matches for a variety of tasks, including optical flow and semantic feature matching.

There are also novel architectures to handle the matching problem. For instance, Yi et al [142] output weights for each match as probabilities for standard RANSAC. DSAC [12] formulates RANSAC in a differentiable manner and train to improve the performance of RANSAC in two view matching. Neural Guided RANSAC [13] similarly improves RANSAC but does not require differentiability, instead relying on methods borrowed from reinforcement learning. Rocco et al [106] use dense 4D convolutional networks and a
weakly supervised loss to find neighborhood matches. Sarlin et al [110] use graph neural networks (GNNs) to handle pairwise matching. In a concurrent work to theirs, we address multi-image matching with GNNs in Chapter 3.

1.4 Goals of This Work

In this work we propose to use various deep learning techniques to learn better matches by learning better feature representations for matches. Specifically, given initial matches and initial representations, we hope to build better representations from these to improve the matching task. Our focus is applying these techniques to structure from motion sub-problems (such as two-view or multi-view matching), but in the last chapter we explore other applications such as point cloud matching.

First in Chapter 2 we discuss a pre-deep-learning method for learning weights for point-wise outlier rejection in optical flow. In the later chapters of this work we hope to leverage deep learning’s ability to learn distributions to improve learning of outlier distributions. In Chapter 3 we use graph neural networks for multi-image matching, using cycle consistency as a loss for training the network. Finally, we conclude with a general framework for learning outlier distributions explicitly with deep learning in Chapter 4.
This first chapter will discuss a robust estimation technique using the class of weighing schemes discussed in Subsection 1.1.2. We propose robust methods for estimating camera egomotion in noisy, real-world monocular image sequences in the general case of unknown observer rotation and translation with two views and a small baseline. This is a difficult problem because of the non-convex cost function of the perspective camera motion equation and because of non-Gaussian noise arising from noisy optical flow estimates and scene non-rigidity. To address this problem, we introduce the expected residual likelihood method (ERL), which estimates confidence weights for noisy optical flow data using likelihood distributions of the residuals of the flow field under a range of counterfactual model parameters. We show that ERL is effective at identifying outliers and recovering appropriate confidence weights in many settings. We compare ERL to a novel formulation of the perspective camera motion equation using a lifted kernel, a recently proposed optimization framework for joint parameter and confidence weight estimation with good empirical properties. We incorporate these strategies into a motion estimation pipeline that avoids falling into local minima. We find that ERL outperforms the lifted kernel method and baseline monocular egomotion estimation strategies on the KITTI dataset, while adding almost no runtime cost over baseline egomotion methods.
Figure 2.1: Schematic depiction of the ERL method for egomotion estimation from noisy flow fields. Figure best viewed in color. (A) Example optical flow field from two frames of KITTI odometry (sequence 5, images 2358-2359). Note the outliers on the grass in the lower right part of the image and scattered throughout the flow field. (B) We evaluate the flow field under M models with translation parameters sampled uniformly over the unit hemisphere. The residuals for the flow field under three counterfactual models are shown. Each black point indicates the translation direction used. Residuals are scaled to $[0,1]$ for visualization. (C) We estimate the likelihood of each observed residual under each of the models by fitting a Laplacian distribution to each set of residuals. The final confidence weight for each flow vector is estimated as the expected value of the residual likelihood over the set of counterfactual models. Likelihood distributions are shown for the three models above. (D) The weighted flow field is used to make a final estimate of the true egomotion parameters. The black point indicates the translation direction estimated using ERL and the green point indicates ground truth. The unweighted estimate of translation is not visible as it is outside of the image bounds.
2.1 MOTIVATION

Visual odometry in real-world situations has attracted increased attention in the past few years in large part because of its applications in robotics domains such as autonomous driving and unmanned aerial vehicle (UAV) navigation. Stereo odometry and simultaneous localization and mapping (SLAM) methods using recently introduced depth sensors have made dramatic progress on real-world datasets. Significant advances have also been achieved in the case of monocular visual odometry when combined with inertial information.

State-of-the-art visual odometry uses either the discrete epipolar constraint to validate feature correspondences and compute inter-frame motion [111] or directly estimates 3D motion and 3D map alignment from image intensities [34]. In contrast to the state of the art, in this chapter we revisit the continuous formulation of structure from motion (SfM), which computes the translational and rotational velocities and depths up to a scale from optical flow measurements. Our motivation lies in several observations:

- UAV control schemes often need to estimate the translational velocity, which is frequently done using a combination of monocular egomotion computations and single-point depths from sonar [14].

- Fast UAV maneuvers require an immediate estimate of the direction of translation (the focus of expansion) in order to compute a time-to-collision map.

- Continuous SfM computations result in better estimates when the incoming frame rate is high and the baseline is very small.

However, estimating camera motion and scene parameters from a single camera (monocular egomotion estimation) remains a challenging problem. This problem case arises in many
contexts where sensor weight and cost are at a premium, as is the case for lightweight UAVs and consumer cameras. Situations involving monocular sensors on small platforms pose additional problems: computational resources are often very limited and estimates must be made in real time under unusual viewing conditions (e.g. with a vertically flipped camera, no visible ground plane, and a single pass through a scene). These contexts present many sources of noise. Real-time flow estimation produces unreliable data, and the associated noise is often pervasive and non-Gaussian, which makes estimation difficult and explicit outlier rejection problematic. Furthermore, violations of the assumption of scene rigidity due to independent motion of objects in the scene can lead to valid flow estimates that are outliers nonetheless. Even in the noise-free case, camera motion estimation is plagued with many suboptimal interpretations (illusions) caused by the hilly structure of the cost function. Additionally, forward motion, which is very common in real-world navigation, is known to be particularly hard for monocular visual odometry [96].

We propose an algorithm suitable for the robust estimation of camera egomotion and scene depth from noisy flow in real-world settings with high-frame-rate video, large images, and a large number of noisy optical flow estimates. Our method runs in real-time on a single CPU and can estimate camera motion and scene depth in scenes with noisy optical flow with outliers, making it suitable for integration with filters for real-time navigation and for deployment on light-weight UAVs. The technical contributions of this chapter are:

- A novel robust estimator based on the expected residual likelihood (ERL) of flow data that effectively attenuates the influence of outlier flow measurements and runs at 30-40 Hz on a single CPU.
2.2 Background

2.2.1 Egomotion/visual odometry

Many approaches to the problem of visual odometry have been proposed. A distinction is commonly made between feature-based methods, which use a sparse set of matching feature points to compute camera motion, and direct methods, which estimate camera motion directly from intensity gradients in the image sequence. Feature-based approaches can again be roughly divided into two types of methods: those estimating camera motion from point correspondences between two frames (discrete approaches) and those estimating camera motion and scene structure from the optical flow measurements induced by the motion between the two frames (continuous approaches). In practice, point correspondences and optical flow measurements are often obtained using similar descriptor matching strategies. Nonetheless, the discrete and continuous approaches use different problem formulations, which reflect differing assumptions about the size of the baseline between the two camera positions.

The continuous approach is the appropriate choice in situations where the real-world camera motion is slow relative to the sampling frequency of the camera. Our approach is primarily intended for situations in which this is the case, e.g. UAVs equipped with high-frame-rate cameras. Accordingly, we focus our review on continuous, monocular methods. For a more comprehensive discussion, see [88].
2.2.2 Continuous, monocular approaches

In the absence of noise, image velocities at 5 or 8 points can be used to give a finite number of candidate solutions for camera motion [85] [51] [95]. With more velocities, there is a unique optimal solution under typical scene conditions [58]. Many methods have been proposed to recover this solution, either by motion parallax [85] [55] [54] [68] or by using the so-called continuous epipolar constraint [88]. The problem is nonlinear and non-convex, but various linear approximation methods have been proposed to simplify and speed up estimation [69] [160] [70].

Although the problem has a unique optimum, it is characterized by many local minima, which pose difficulties for linear methods [23]. Furthermore, in the presence of noise, many methods are biased and inconsistent in the sense that they do not produce correct estimates in the limit of an unlimited number of image velocity measurements [151]. Many methods also fail under many common viewing conditions or with a limited field of view [28]. Recently, [36] and [37] proposed branch-and-bound methods that estimate translational velocity in real time and effectively handle a large numbers of outliers. However, these methods deal with the case of pure translational camera motion, while our approach estimates both translational and rotational motion.

Most directly related the work in this chapter is the robust estimation framework presented in [150]. They propose a method based on a variant of a common algebraic manipulation and show that this manipulation leads to an unbiased, consistent estimator. They pose monocular egomotion as a nonlinear least-squares problem in terms of the translational velocity. In this framework, angular velocity and inverse scene depths are also easily recovered after translational velocity is estimated. To add robustness, they use a loss function with sub-quadratic growth, which they solve by iteratively re-weighted least squares (IRLS). We use a similar formulation but demonstrate several novel methods for
estimating the parameters of a robust loss formulation. Our methods have properties that are well-suited for dealing with image sequences containing several thousand flow vectors in real time. In particular, we demonstrate that the ERL method adds robustness without requiring costly iterative re-weighting, resulting in very little runtime overhead.

Other methods for monocular odometry augment velocity data with planar homography estimates [43] [118] or depth filters [34] to estimate scale. In this chapter, we do not rely on ground-plane estimation in order to maintain applicability to cases such as UAV navigation, where image sequences do not always contain the ground plane. Because we focus on frame-by-frame motion estimation, we cannot rely on a filtering approach to estimate depth. Our method can be augmented with domain-appropriate scale or depth estimators as part of a larger SLAM system.

### 2.2.3 Robust optimization

In this chapter, we propose to increase the robustness of monocular egomotion estimation (1) by estimating each flow vector’s confidence weight as its expected residual likelihood (ERL) and (2) by using a lifted robust kernel to jointly estimate confidence weights and model parameters. ERL confidence weights are conceptually similar to the weights recovered in the IRLS method for optimizing robust kernels [57]. Robust kernel methods attempt to minimize the residuals of observations generated by the target model process ("inliers") while limiting the influence of other observations ("outliers"). Such methods have been used very successfully in many domains of computer vision [45] [7]. However, we are unaware of any previous work that attempts to estimate confidence weights based on the distribution of residuals at counterfactual model parameters, as we do in the ERL method.
The lifted kernel approach offers another method to design and optimize robust kernels in particularly desirable ways. Lifted kernels have recently been used in methods for bundle adjustment in SfM \[145\], object pose recovery \[147\], and non-rigid object reconstruction \[161\]. Our lifted kernel approximates the truncated quadratic loss, which has a long history of use in robust optimization in computer vision \[8\] and has demonstrated applicability in a wide variety of problem domains.

Previous studies have used robust loss functions for monocular egomotion \[150\], visual SLAM \[94\], and RGB-D odometry \[71\]. To our knowledge, we present the first application of lifted kernels for robust monocular egomotion. Noise is typically handled in odometry by using sampling-based iterative methods such as RANSAC, which makes use of a small number of points to estimate inlier sets (typically five or eight points in monocular methods). The use of a robust kernel allows us to derive our final estimate from a larger number of points. This is desirable because the structure of the problem of continuous monocular odometry admits fewer correct solutions when constrained by a larger number of input points, which can better reflect the complex depth structure of real scenes. Our robust methods allow us to take advantage of a large number of flow estimates, which, while noisy, may each contribute weakly to the final estimate.

2.3 Problem Formulation and Approach

In this section, we present the continuous formulation of the problem of monocular visual egomotion. We describe and motivate our approach for solving the problem in the presence of noisy optical flow. We then describe two methods for estimating the confidence weights for each flow vector in a robust formulation of the problem, as well as the pipeline we use
to estimate camera motion and scene depth. We derive the equations in more detail for non-robust and robust versions.

2.3.1 Visual egomotion computation and the motion field

In the continuous formulation, visual egomotion methods attempt to estimate camera motion and scene parameters from observed local image velocities (optical flow). The velocity of an image point due to camera motion in a rigid scene under perspective projection is given by

\[ \mathbf{u}(\mathbf{x}_i) = \rho(\mathbf{x}_i)A(\mathbf{x}_i)\mathbf{t} + B(\mathbf{x}_i)\mathbf{\omega}, \]

where \( \mathbf{u}_i(\mathbf{x}_i) = (u_i, v_i)^T \in \mathbb{R}^2 \) is the velocity (optical flow) at image position \( \mathbf{x}_i = (x_i, y_i)^T \in \mathbb{R}^2 \), \( \mathbf{t} = (t_x, t_y, t_z)^T \in \mathbb{R}^3 \) is the camera’s instantaneous translational velocity, \( \mathbf{\omega} = (\omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^3 \) is the camera’s instantaneous rotational velocity, and \( \rho(\mathbf{x}_i) = \frac{1}{Z(\mathbf{x}_i)} \in \mathbb{R} \) is the inverse of scene depth at \( \mathbf{x}_i \) along the optical axis. We normalize the camera’s focal length to 1, without loss of generality. In the case of calibrated image coordinates,

\[ A(\mathbf{x}_i) = \begin{bmatrix} 1 & 0 & -x_i \\ 0 & 1 & -y_i \end{bmatrix}, \]

\[ B(\mathbf{x}_i) = \begin{bmatrix} -x_i y_i & 1 + x_i^2 & -y_i \\ -1 - y_i^2 & x_i y_i & x_i \end{bmatrix}. \]

This formulation is appropriate for the small-baseline case where point correspondences between frames can be treated as 2D motion vectors.
The goal of monocular visual egomotion computation is thus to estimate the six motion parameters of \( t \) and \( \omega \) and the \( N \) values for \( \rho \) from \( N \) point velocities \( u \) induced by camera motion. \( t \) and \( \rho \) are multiplicatively coupled in equation \( 2.1 \) above, so \( t \) can only be recovered up to a scale. We therefore restrict estimates of \( t \) to the unit hemisphere, \( \|t\| = 1 \).

The full expression for the set of \( N \) point velocities can be expressed compactly as

\[
\mathbf{u} = \mathbf{A}(t)\mathbf{\rho} + \mathbf{B}\mathbf{\omega}. \tag{2.2}
\]

where the expressions for \( A(x) \), \( B(x) \), and \( \rho(x) \) for all \( N \) points are

\[
A(t) = \begin{bmatrix}
A(x_1)t & 0 & \ldots & 0 \\
0 & A(x_2)t & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & A(x_N)t
\end{bmatrix} \in \mathbb{R}^{2N \times N}
\]

\[
B = \begin{bmatrix}
B(x_1) \\
B(x_2) \\
\vdots \\
B(x_N)
\end{bmatrix} \in \mathbb{R}^{2N \times 3}
\]

and the velocity and depth for each of the points are concatenated to form \( \mathbf{u} = (\mathbf{u}_1^T, \mathbf{u}_2^T, \ldots, \mathbf{u}_N^T)^T \in \mathbb{R}^{2N \times 1} \) and \( \mathbf{\rho} = (\rho(x_1), \rho(x_2), \ldots, \rho(x_N))^T \in \mathbb{R}^{N \times 1} \). We estimate camera motion and scene depth by minimizing the objective

\[
\min_{t, \mathbf{\rho}, \mathbf{\omega}} E(t, \mathbf{\rho}, \mathbf{\omega}) = \min_{t, \mathbf{\rho}, \mathbf{\omega}} L(r(t, \mathbf{\rho}, \mathbf{\omega}))
\]

\[
= \min_{t, \mathbf{\rho}, \mathbf{\omega}} \|A(t)\mathbf{\rho} + B\mathbf{\omega} - \mathbf{u}\|_2^2. \tag{2.3}
\]
Here, \( L(x) : R^N \rightarrow R \) is a loss function and \( r(t, \rho, \omega) : R^{N+6} \rightarrow R^N \) is a residual function for the flow field depending on the estimated model parameters. We first describe the case of an unweighted residual function under a quadratic loss, which is suitable for the case of Gaussian noise.

Following [150], we note that no loss of generality occurs by first solving this objective for \( \rho \) in the least-squares sense. Minimizing over \( \rho \) gives

\[
\min_{t, \omega} \min_{\rho} \| A(t) \rho + B \omega - u \|_2^2
\]

\[
= \min_{t, \omega} \| A^\perp(t)^T (B \omega - u) \|_2^2
\]

(2.4)

where \( A^\perp(t) \) is the orthogonal compliment to \( A(t) \). This expression no longer depends on \( \rho \) and depends on \( t \) only through \( A^\perp(t)^T \), which is fast to compute due to the sparsity of \( A(t) \).

In the absence of noise, we could proceed by directly minimizing the non-robust loss function (equation 2.4) in \( t \) and \( \omega \). In particular, given a solution for \( t \), we can directly solve for \( \omega \) by least squares in \( O(N) \) time. In the noiseless case, we estimate \( t \) by optimizing

\[
\min_t \| A^\perp(t)^T (B \hat{\omega}(t) - u) \|_2^2,
\]

(2.5)

where \( \hat{\omega}(t) \) is the least-squares estimate of \( \omega \) for a given \( t \). This method of estimating \( t, \rho, \) and \( \omega \) was shown to be consistent in [151]. That is, in the absence of outliers, this method leads to arbitrarily precise, unbiased estimates of the motion parameters as the sample size increases.

In the next few sections, we will derive the specifics of Equation 2.4 in the context of non-robust least squares estimation from Zhang et al [150] and Soatto and Brockett [117].
2.3 Problem Formulation and Approach

Derivation of Standard Linear Least Squares Estimate

Now we derive Equation 2.4 in greater detail. This derivation is similar to that in Zhang et al [150]. We first minimize Equation 2.4 with respect to the inverse depths \( \rho \), giving

\[
\min_\rho E(t, \rho, \omega) = \min_\rho \| A(t) \rho + B \omega - u \|_2^2
\]

\[
= \| - A(t) \left( A^T(t) A(t) \right)^{-1} A^T(t) (B \omega - u) + B \omega - u \|_2^2
\]

\[
= \left\| \left( I - A(t) \left( A^T(t) A(t) \right)^{-1} A^T(t) \right) (B \omega - u) \right\|_2^2
\]

\[
= \left\| A^\perp(t)^T (B \omega - u) \right\|_2^2.
\]

We now have an expression in terms of the orthogonal complement of \( A(t) \). Finding this orthogonal complement is fairly simple since it is sparse. To show this, we first note that the orthogonal complement of \( A(t) \) is the null space of \( A^T(t) \). \( A^T(t) \) is of the form

\[
A^T(t) = \begin{bmatrix}
  t^T A^T(x_1) & 0 & \ldots & 0 \\
  0 & t^T A^T(x_2) & \ldots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \ldots & t^T A(x_n)^T
\end{bmatrix} \in \mathbb{R}^{n \times 2n}.
\]

Each of the rows of \( A^T(t) \) are orthogonal, so we can consider each of the rows individually. Consider the vector

\[
\phi_i = \left( 0, 0, \ldots, 0, t^T A^T(x_i) J^T, 0, \ldots, 0 \right)^T,
\]
where

\[
J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}
\]

is a skew-symmetric matrix in \( \mathbb{R}^{2 \times 2} \). By construction, this vector is orthogonal to the \( i \)th column of \( A^T(t) \). We normalize and concatenate these vectors to form the matrix

\[
A^\perp(t) = \begin{bmatrix} \phi_1 \|\phi_1\| & \phi_2 \|\phi_2\| & \ldots & \phi_n \|\phi_n\| \end{bmatrix}.
\]

This matrix is very sparse, so we can compute products with it very efficiently: \( A^\perp(t)^T B \) and \( A^\perp(t)^T u \) can be computed in \( O(n) \) time. From here, the least squares estimate of \( \omega \) (used in Equation 2.5) can be computed as:

\[
\hat{\omega}(t) = \left( B^T A^\perp(t) A^\perp(t)^T B \right)^{-1} B^T A^\perp(t) A^\perp(t)^T u.
\] (2.6)

In summation notation:

\[
\hat{\omega}(t) = \left( \sum_{i=1}^{n} \frac{B^T(x_i)JA(x_i)tt^T A^T(x_i)J^T B(x_i)}{\|JA(x_i)V\|^2} \right)^{-1} \left( \sum_{i=1}^{n} \frac{B^T(x_i)JA(x_i)tt^T A^T(x_i)J^T u_i}{\|JA(x_i)t\|^2} \right).
\] (2.7)

There are \( 2n \) terms to compute, and one inversion of a 3 by 3 matrix, making this \( O(n) \) time to compute. Taken altogether, we compute the residual given \( t \) by Equation 2.5, listed again for clarity:

\[
\|A^\perp(t)^T (B\hat{\omega}(t) - u)\|_2^2.
\]
To compute the residual, we use the error vector \( E \), defined as:

\[
E_i(t) = \frac{t^T A^T(x_i) J^T}{\|J A(x_i) t\|} (B(x_i) \hat{\omega}(t) - u_i) \quad \text{for } i = 1, \ldots, n \tag{2.8}
\]

\[
E(t) = (E_1(t), E_2(t), \ldots, E_n(t))^T. \tag{2.9}
\]

The residual is exactly \( \|E(t)\|^2 = \sum_i \|E_i(t)\|^2 \). As there are \( n \) of these error terms, and \( \hat{\omega} \) takes \( O(n) \) to compute, this residual calculation takes \( O(n) \) to compute. This was shown to be an unbiased estimator in [150].

**Implementation Details of Soatto/Brockett Algorithm**

Now we derive an alternate way of optimizing Equation 2.4, derived in [117]. This method first simplifies the loss function so that it is easier and faster to optimize. Below is an extension of the original work deriving a method to express the cost function as a degree 6 rational function with precomputed coefficients. This method can be computed much faster than the method from the previous section, but as shown in Section 2.4, it is less robust.

**EXPRESSION OF \( \hat{\omega} \) ** Recall Equation 2.7. We rewrite this as:

\[
\hat{\omega}(t) = G_{full}(t)^{-1} H_{full}(t) \tag{2.10}
\]

\[
G_{full}(t) = \sum_{i=1}^{n} \frac{B^T(x_i) J A(x_i) t t^T A^T(x_i) J^T B(x_i)}{\|J A(x_i) V\|^2} \tag{2.11}
\]

\[
H_{full}(t) = \sum_{i=1}^{n} \frac{B^T(x_i) J A(x_i) t t^T A^T(x_i) J^T u_i}{\|J A(x_i) t\|^2}. \tag{2.12}
\]
As in [23], we drop the denominator terms. This gives us: The first term we need to consider is the 3 by 3 matrix we need to invert.

\[ \dot{\omega}(t) = G(t)^{-1}H(t) \]  

(2.13)

\[ G(t) = \sum_{i=1}^{n} B^T(x_i)JA(x_i)tt^TA^T(x_i)J^TB(x_i) \]  

(2.14)

\[ H(t) = \sum_{i=1}^{n} B^T(x_i)JA(x_i)tt^TA^T(x_i)J^Tu_i. \]  

(2.15)

We focus on \( G(t) \) first. We can write this out in terms of quadratic terms of \( tt^T \) by introducing the matrices \( S_{ij}^k \), defined as:

\[ S_{kl}^{ij} = \begin{cases} 
1 & \text{if } i = k, j = l, \text{ or } i = l, j = k \\
0 & \text{otherwise} 
\end{cases} \]
\[ G(t) = t_i^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{11}A^T(x_i)J^T B(x_i) \right) \]
\[ + t_1 t_2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{12}A^T(x_i)J^T B(x_i) \right) \]
\[ + t_1 t_3 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{13}A^T(x_i)J^T B(x_i) \right) \]
\[ + t_2^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{22}A^T(x_i)J^T B(x_i) \right) \]
\[ + t_2 t_3 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{23}A^T(x_i)J^T B(x_i) \right) \]
\[ + t_3^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{33}A^T(x_i)J^T B(x_i) \right) \]
\[ = \sum_{i<j} t_i t_j G^{ij}, \]

where \( G^{ij} \) is defined appropriately. We will use \( G_k^{ij} \) to denote the \( k^{th} \) column of \( G^{ij} \). We know that the inverse of a 3 by 3 matrix with columns \( c_1, c_2, c_3 \) has an inverse given by

\[
\frac{1}{\det([c_1 \ c_2 \ c_3])} \begin{bmatrix} (c_2 \times c_3)^T \\ (c_3 \times c_1)^T \\ (c_1 \times c_2)^T \end{bmatrix}.
\]
We also know that the cross product is bi-linear, so from this we can write out the inverse of $G$ analytically.

$$G^{-1}(t) = \frac{1}{\det(G(t))} \left[ \left( \sum_{i<j} t_i t_j G_{ij}^2 \right) \times \left( \sum_{k<l} t_k t_l G_{kl}^3 \right)^T + \left( \sum_{i<j} t_i t_j G_{ij}^3 \right) \times \left( \sum_{k<l} t_k t_l G_{kl}^1 \right)^T + \left( \sum_{i<j} t_i t_j G_{ij}^1 \right) \times \left( \sum_{k<l} t_k t_l G_{kl}^2 \right)^T \right]$$

The terms in the matrix component become a $4^{th}$ degree polynomial of 3 variables with 15 terms (after grouping) with matrix coefficients. We can also compute the
determinant explicitly using the fact that the determinant of a 3 by 3 matrix is the triple product of its columns.

\[
\det(G(t)) = (G_2(t) \times G_3(t))^\top G_1(t)
\]

\[
= \left( \sum_{i<j,k<l} t_i t_j t_k t_l \left( G_{ij}^2 \times G_{kl}^3 \right) \right) \left( \sum_{p<q} t_p t_q G_{pq}^1 \right)
\]

\[
= \sum_{i<j,k<l,p<q} t_i t_j t_k t_l t_p t_q \left( G_{ij}^2 \times G_{kl}^3 \right)^\top G_{pq}^1
\]

After grouping terms, this becomes a 6\(^{th}\) degree polynomial with 28 terms. This makes each element of \( G^{-1} \) a 6\(^{th}\) degree rational function.
In a similar fashion, we find the expression

\[
H(t) = t_1^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{11}A^T(x_i)J^T u_i \right) 
+ t_1 t_2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{12}A^T(x_i)J^T u_i \right) 
+ t_1 t_3 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{13}A^T(x_i)J^T u_i \right) 
+ t_2^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{22}A^T(x_i)J^T u_i \right) 
+ t_2 t_3 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{23}A^T(x_i)J^T u_i \right) 
+ t_3^2 \left( \sum_{i=1}^{n} B^T(x_i)JA(x_i)S^{33}A^T(x_i)J^T u_i \right) 
= \sum_{i<j} t_i t_j H^{ij}
\]

This gives the final form of the equation:

\[
\hat{\omega}(t) = \frac{1}{\text{det}(G(t))} \sum_{i<j,k<l,p<q} t_i t_j t_k t_l t_p t_q \begin{bmatrix} (G_2^{ij} \times G_3^{kl})^T \\ (G_3^{ij} \times G_1^{kl})^T \\ (G_1^{ij} \times G_2^{kl})^T \end{bmatrix} H^{pq} \tag{2.16}
\]

We are left with \(\hat{\omega}(t)\) as a degree-6 rational function of \(V\), meaning it has 28 terms in the numerator and denominator for each element.
First, we express the cost function as:

\[ f(t) = \sum_i \left( t^T A^T(x_i) J^T (B(x_i) \hat{\omega}(t) - u_i) \right)^2 \]  \hspace{1cm} (2.17)

Now we expand and simplify this by plugging in the definitions given above for G and H:

\[
\begin{align*}
    f(t) &= \sum_i \left( t^T A^T(x_i) J^T (B(x_i) \hat{\omega}(t) - u_i) \right)^2 \\
    &= \sum_i (B(x_i) \hat{\omega}(t) - u_i)^T J A(x_i) t t^T A^T(x_i) J^T (B(x_i) \hat{\omega}(t) - u_i) \\
    &= \hat{\omega}(t)^T \left( \sum_i B^T(x_i) J A(x_i) t t^T A^T(x_i) J^T B(x_i) \right) \hat{\omega}(t) \\
    & \quad - \left( \sum_i B^T(x_i) J A(x_i) t t^T A^T(x_i) J^T u_i \right)^T \hat{\omega}(t) \\
    & \quad + t^T \left( \sum_i A^T(x_i) J^T u_i u_i^T J A(x_i) \right) T \\
    &= \hat{\omega}(t)^T G(t) \hat{\omega}(t) - 2H(t)^T \hat{\omega}(t) + t^T S t \\
    &= \hat{\omega}(t)^T G(t) (G^{-1}(t) H(t)) - 2H(t)^T \hat{\omega}(t) + t^T S t \\
    &= \hat{\omega}(t)^T H(t) - 2H(t)^T \hat{\omega}(t) + t^T S t \\
    &= t^T S t - H(t)^T \hat{\omega}(t),
\end{align*}
\]

where

\[
S = \sum_i A^T(x_i) J^T u_i u_i^T J A(x_i).
\]

This gives us the final equation

\[
f(t) = t^T S t - H(t)^T \hat{\omega}(t). \hspace{1cm} (2.18)
\]
2.3.2 Robust Formulations

However, the manipulations introduced in Equations 2.4 and 2.5 rely on least-squares solutions and are not stable in the presence of outliers. Accordingly, instead of directly solving 2.5, we propose to solve a robust form. To do so, we introduce a confidence weight for each flow vector \( w_i(u_i) \in [0, 1] \) to give

\[
\begin{align*}
\min_t & L(r(t, \hat{\omega}(t)), w) \\
= & \min_t \| w \circ A^\perp(t)^\top (B\hat{\omega}(t) - u) \|_2^2, \\
\end{align*}
\]

where \( w = (w(u_1), w(u_2), \ldots, w(u_N))^\top \in [0, 1]^N \) is the vector of all weights, \( r \in \mathbb{R}^N \) is the vector of residuals for the flow field at some estimate of \( t \), and \( \circ \) is the Hadamard product.

Each entry \( w(u_i) \) of \( w \) attempts to weight the corresponding data point \( u_i \) proportionally to its residual at the optimal model parameters \((\hat{t}, \hat{\rho}, \hat{\omega})\), reflecting the degree to which the point is consistent with a single generating function for the motion in the scene, possibly with Gaussian noise. In other words, it reflects the degree to which \( u_i \) is an inlier for the optimal model of camera motion in a rigid scene. This is equivalent to replacing the choice of \( L(x) = x^2 \) as the loss in Equation 2.5 with a function that grows more slowly e.g. truncated square.

We introduce a method to directly estimate the confidence weights as the expected residual likelihood (ERL) for each flow vector given the distribution of residuals for the flow field at a range of model parameters consistent with the solution in 2.5. We interpret each weight in terms of an estimate of the validity of the corresponding point under the model: that is, as an estimate of the point’s residual at the optimal model parameters in a noise-free context. We compare ERL to a method that replaces \( L(x) = x^2 \) in 2.5 with a
lifted truncated quadratic kernel \cite{145} and jointly optimizes the confidence weights and model parameters. We demonstrate that ERL outperforms the lifted kernel approach on the KITTI dataset, and both of these approaches outperform existing methods for monocular egomotion computation.

**Confidence weight estimation by expected residual likelihood**

Here, we describe the ERL method for estimating the confidence weights in Equation 2.19, and we demonstrate that this method provides a good estimate of the appropriate confidence weights in the case of optical flow for visual egomotion.

At the optimal model parameters, \((t^*, \rho^*, \omega^*)\), the residuals for inlier points (i.e. correct flow vectors due to rigid motion) are distributed according to a normal distribution, reflecting zero-mean Gaussian noise. However, in the presence of outliers, a zero-mean Laplacian distribution provides a better description of the residual distribution (see Fig. 2.5). Accordingly, we can fit a Laplacian distribution to the observed residuals at the optimal model parameters to approximate the probability density function for residuals.

We use this property to identify outliers as those points that are inconsistent with the expected residual distribution at a range of model values. For each point, we compute the likelihood of each observed, scaled residual as

\[
p(r_{m1}^u | t_m, \rho_m, \omega_m) = \mathcal{L}(\hat{r}_u; \hat{\mu}_m, \hat{b}_m),
\]

where \(\hat{r}_{m1}^u\) is the scaled residual under the \(m\)th model \((t_m, \rho_m, \omega_m)\) at the \(i\)th flow vector and \(\hat{r}_u = (\hat{r}_{u1}^m, \hat{r}_{u2}^m, ..., \hat{r}_{uN}^m)^T\). We fit \(\hat{\mu}_m\) and \(\hat{b}_m\), respectively the location and scale parameters of the Laplacian distribution, to the set of scaled residuals \(\hat{r}_u\) using maximum likelihood.
Figure 2.2: A 2D line-fitting problem demonstrating how ERL weights inliers and outliers. Inliers are generated as $y_i \approx 2x_i + 1$ with Gaussian noise. Each data points is colored according to its estimated confidence weight.

Because inliers exhibit smaller self-influence than outliers [63], inlier residuals will typically be associated with higher likelihood values. However, the distribution used to estimate the likelihood reflects both the inlier and outlier points. If the counterfactual model parameters used to estimate the $m^{th}$ likelihood correspond to a model that is highly suboptimal, some outliers may be assigned higher likelihoods than they would be at the optimal model. Moreover, the presence of Gaussian noise means that the estimated likelihood for individual inliers may be erroneously low by chance for a particular model even if the optimal exponential distribution is exactly recovered.

To arrive at more reliable estimates and to discount the effect of erroneous likelihoods due to the specific model parameters being evaluated, we estimate the expected residual likelihood for each data point by evaluating the likelihood under $M$ models,

$$\hat{w}_i = \mathbb{E}[\hat{r}^m_{u_i}] = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(\hat{r}_{u_i}; \hat{\mu}_m, \hat{b}_m).$$

(2.21)

This method returns a vector $\hat{w} \in \mathbb{R}^N$. To use $\hat{w}$ as confidence weights in a robust optimization context, we scale them to the interval $[0, 1]$. Scaling the maximum $\hat{w}_i$ to 1 and the minimum $\hat{w}_i$ to 0 for each flow field works well in practice.
Figure 2.3: Robust methods recover the error surface of the outlier-free flow field. (A) Example optical flow field from two frames of KITTI odometry (sequence 10, images 14-15). Note the prominent outliers indicated by the yellow box. Error surfaces on this flow field for (A) the raw method (Equation 2.5) with all flow vectors, (B) with outliers removed by hand, and (C) with confidence weights estimated by ERL or (D) the lifted kernel. The green point is the true translational velocity and the black point the method’s estimate. Blue: low error. Red: high error. Translation components are given in calibrated coordinates.

The full process to estimate weights by ERL is shown in Algorithm 1. This method returns confidence weights in $O(MN)$ time, where $M$ is set by the user. Empirically, the ERL method gives results that reflect the inlier structure of the data with small values of $M$ (we use $M \approx 100$), allowing very quick run times. In practice, the method assigns high weights to very few outliers while assigning low weights to acceptably few inliers. Thus, the method balances a low false positive rate against a moderately low false negative rate. This is a good strategy because our method takes a large number of flow vectors as input,
Algorithm 1 ERL confidence weight estimation

Input: Measured flow \( \{u_n\}_{n=1}^N \), sampled translational velocities \( \{t_m\}_{m=1}^M \)

Output: Estimated confidence weights \( \{\hat{w}_n\}_{n=1}^N \)

for all \( m \) do
  Compute scaled residuals:
  \[
  \hat{r}_u = |A^⊥(t_m)^T(B\hat{\omega}_m(t_m) - u)|
  \]
  Compute maximum likelihood estimators of residual distribution:
  \[
  \hat{\mu}_m = \text{median}(\hat{r}_u)
  \]
  \[
  \hat{b}_m = \frac{1}{N} \sum_{n=1}^{N} \| \hat{r}_{u_n} - \hat{\mu}_m \|
  \]
end for

for all \( n \) do
  Compute confidence weights as expected likelihood under Laplacian fits:
  \[
  \hat{w}_n = \frac{1}{M} \sum_{m=1}^{M} L(\hat{r}_{u_n}; \hat{\mu}_m, \hat{b}_m)
  \]
end for

return \( \{\hat{w}_n\}_{n=1}^N \)

which leads to redundancy in the local velocity information. Fig. 2.2 illustrates the ERL method’s use in a simple 2D robust line-fitting application.

As discussed above, choosing values for the confidence weights in a least squares objective is equivalent to fitting a robust kernel. We note that regression under the assumption of Laplacian noise leads to an \( L_1 \) cost. However, we have no guarantees about the form of the robust kernel corresponding to the weights chosen by the ERL method. Accordingly, we also explored using a robust kernel with known properties.
2.3.3 Robust estimation using a lifted kernel

Here, we explore the effect of jointly optimizing the confidence weights, $w(u)$, and $\omega$ for a given value of $t$ using the lifted kernel approach described in [145]. In our case, a lifted kernel takes the form

$$
\min_{t,\omega,w} \hat{L}(r(t, \omega), w)
= \min_t \min_{\omega,w} \|w \circ A^\perp(t)^\top (B\omega(t) - u)\|_2^2 + \sum_{i=1}^N \kappa^2(w_i^2)),
\tag{2.22}
$$

where the lifted kernel of the loss $L$ is denoted as $\hat{L}$. $\kappa(x) : \mathbb{R} \rightarrow \mathbb{R}$ is a regularization function applied to the weights. Because this approach does not rely on the least squares solution for rotational velocity, $\hat{\omega}$, it may gain additional robustness to noise. This approach also allows us to estimate the confidence weights for particular values of $t$, unlike the ERL approach, which relies on estimates at several values of $t$ to produce stable results.

Different choices of $\kappa$ produce different kernels. We use

$$
\kappa(w^2) = \frac{\tau}{\sqrt{2}}(w^2 - 1),
\tag{2.23}
$$

which gives a kernel that is a smooth approximation to the truncated quadratic loss [145]. $\tau$ is a hyper-parameter that determines the extent of the quadratic region of the truncated quadratic loss. We set $\tau = 0.05$ for all results shown here, but other choices give similar results.

The lifted kernel approach to solving nonlinear least squares problems is similar to IRLS insofar as it incorporates confidence weights on each of the data points and optimizes the values of these weights in addition to the value of the target model parameters. However, rather than alternately estimating the best weights given estimated model parameters and
the best model parameters given estimated weights, the lifted approach simultaneously optimizes for both weights and model parameters, effectively "lifting" a minimization problem to a higher dimension.

The lifted kernel approach has several properties that are particularly beneficial for encouraging fast convergence. First, by using the weights to increase the dimensionality of the optimization problem, the lifted kernel minimizes the extent of regions of low gradient in the cost function. This ensures the method can quickly and reliably converge to minima of the function. Second, optimization can exploit the Gauss-Newton structure of the joint nonlinear least-squares formulation for faster convergence than the slower iterative-closest-points-like convergence exhibited by IRLS.

To illustrate the effect of our two robust optimization strategies, we display the error surfaces for the ERL and lifted-kernel methods on a sample flow field from KITTI (Fig. 2.3). The error surfaces are shown as a function of the translational velocity. Both methods recover error surfaces that resemble the error due to inlier flow vectors. The confidence weights estimated by ERL generally more closely resemble the pattern of inliers and outliers in flow data. To produce the results for the case with outliers removed, we strengthened the maximum bidirectional error criterion for flow inclusion to eliminate noisy matches and manually removed obvious outliers from the flow field.

**Lifted Weights Formulation**

Now we derive the exact form of the lifted kernel loss function for Equation 2.19. Fixing the t term, the equation becomes linear:

\[
\min_\omega \| A^\perp(t)^T B \omega - A^\perp u \|^2_2 = \| f(\omega) \|^2_2.
\]
So we see the Jacobian is given by $\nabla f(\omega) = A^\perp(t)^\top B$ and thus, as in Zach’s work \cite{zach2011}, our lifted cost function takes the form

$$
\min_{\omega, w} \left\| \begin{pmatrix} w \circ f(\omega) \\ \kappa(w \circ w) \end{pmatrix} \right\|_2^2.
$$

Here we use the smooth truncated quadratic for our $\kappa$ function (applied element-wise):

$$
\kappa(w^2) = \frac{\tau}{\sqrt{2}}(w^2 - 1),
$$

where $\tau$ is a hyper-parameter. Therefore the Jacobian used for the Gauss-Newton iteration is:

$$
\hat{J} = \begin{pmatrix}
\text{diag}(w) \nabla f(\omega) & \text{diag}(f(\omega)) \\
0 & \nabla \kappa(w \circ w),
\end{pmatrix}
$$

where $\nabla f$ and $\nabla \kappa$ denote the Jacobian of $f$ and $\kappa$, respectively. From here, the standard Gauss-Newton update to compute the update step for the weights and parameters $\omega$.

We follow Equations (29) and (35) from Zach’s work \cite{zach2011} in our code for more efficient computation.

## 2.4 Experiments

We compare the performance of the proposed methods (called ”ERL” and ”Lifted Kernel” in the figures) to several baseline methods for monocular egomotion/visual odometry from the literature: 5-point epipolar+RANSAC (using \cite{zhang1989}), 8-point epipolar+RANSAC (using \cite{tomasi1992}), and two continuous epipolar methods - Zhang/Tomasi \cite{zhang2004}, which is identical to
Equation 2.5, and Soatto/Brockett [117] (for implementation details see Section 2.3.1 for more details). All experiments were run on a desktop with an Intel Core i7 processor and 16 GB of RAM. A single CPU core was used for all experiments.

With \( \sim 1000 \) flow vectors, the ERL method runs at 30-40 Hz in an unoptimized C++ implementation. Because of the low overhead of the ERL procedure, this is effectively the same runtime as the Zhang/Tomasi method. The lifted kernel optimization has no convergence guarantees, and it typically runs at <1 Hz in a MATLAB implementation. Note that both of these run times can be significantly improved with better optimization.

The Soatto/Brockett method runs extremely quickly (>500 Hz), but performs poorly on real sequences. The implementation of epipolar+RANSAC used here runs at \( \sim 25 \) Hz.

Optical flow for all our results was extracted using a multi-scale implementation of the KLT method [87] [124].

For both ERL and the lifted approach, we optimize \( t \) using Gauss-Newton. We initialize \( t \) at a grid of values spaced over the unit hemisphere to decrease the chance of converging to a non-global minimum. We then prune the grid to a single initial value \( t_0 \) by choosing the grid point that gives the lowest residual under Equation 2.19 or 2.22 for ERL or the lifted kernel, respectively. We then optimize to convergence starting from \( t_0 \). This pruning strategy is effective at avoiding local minima because good estimates for the weights return an error surface that is very similar to the noiseless case (see Fig. 2.3) and this error surface
is smooth with respect to the sampling density we use (625 points) [23]. Confidence weights for ERL are computed using model parameters sampled on a coarser grid (100 points), as this is adequate to give good confidence weight estimates.

For all tests using the lifted kernel, we optimize the expression in Equation 2.22 using the efficient Schur compliment implementation of Levenberg-Marquardt described in [145]. Details of the optimization procedure used here are given in the next section, Section 2.4.1. We did not explore jointly optimizing over $t$, $\omega$, and $w$, but joint optimization over these model parameters with a lifted kernel is possible, and we plan to explore its use in future work.

### 2.4.1 Goodness of Fit for Laplacian Distribution

To justify the use of a Laplacian distribution for ERL, we used ground truth flow fields to examine the distribution of errors in estimated optical flow. Ground truth flow was obtained using the KITTI Stereo dataset. Flow was produced according to Equation 2.1. All images containing both depth and odometry ground truth were used. Errors were obtained for flow at all points that both contained ground truth depth and produced a sufficiently good KLT flow vector, using the same inclusion criteria as the main paper. We fit Laplacian and Gaussian distributions to the errors in the estimated optical flow, and computed the sum of the log likelihoods of each errors in the estimated distributions. In Figure 2.5 we plot the relative likelihoods of the data under the two distributions, and it is clear that the Laplacian fits consistently produce a higher likelihood than the Gaussian fits.
Figure 2.5: Log likelihoods of Laplacian and Gaussian fits to the optical flow error, Gaussian likelihood on the x-axis and Laplacian likelihood on the y-axis. Laplacian fits are consistently better than Gaussian fits.

2.4.2 Comparison Of Difference ERL Likelihood Schemes

We also compared the results obtained by ERL different likelihood functions over the KITTI Odometry dataset. We compare the results obtained using a Laplacian or a Gaussian fit to compute the weights in ERL. Results are shown in Figure 2.6. The use of a Laplacian distribution leads to a small but consistent improvement.

2.4.3 Evaluation on KITTI

We evaluate the performance of our method using the KITTI dataset [44], which is a collection of real-world driving sequences with ground-truth camera motion and depth data. The sequences contained in the dataset are challenging for state-of-the-art odometry methods for several reasons. First, they contain large inter-frame motions and repetitive scene structures that make estimating accurate flow correspondences difficult in real time. Second, several sequences feature little to no camera motion, which typically causes
monocular odometry methods to fail. Finally, some sequences contain independent motion due to other vehicles and pedestrians, which violates the assumption of scene rigidity and makes reliable odometry more difficult.

All results are performed on neighboring frames of the KITTI odometry dataset (no skipped-frame sequences are evaluated), as these image pairs better match the modeling assumptions of continuous egomotion/odometry methods. All sequences were captured at 10 Hz at a resolution of 1392 x 512 pixels. We evaluated all methods on all 16 sequences of the KITTI odometry test set.
Figure 2.7: Distributions of errors. (a) Full distribution of translational velocity errors. (b) Full distribution of rotational velocity errors.

The results for methods on KITTI are shown in Figs. 4-6. For ease of visualization, the results for the 5-point epipolar method with RANSAC are not shown (they were significantly worse than all other methods we attempted). This could be because the small baselines in our datasets makes the degenerate cases of the 5-point solver more pronounced. ERL produces the best estimates of translational velocity, while the lifted kernel produces results of similar quality to 8-point epipolar with RANSAC and the Zhang/Tomasi method. ERL, the lifted kernel, and Zhang/Tomasi produce rotational velocity estimates of similar quality. The 8-point epipolar method produces worse estimates in this case because of the large baseline assumption, which is not suitable for rotational velocity estimation under these conditions. Soatto/Brockett produces bad estimates in these test cases because of the bias introduced by its algebraic manipulation.

2.4.4 Synthetic sequences

To estimate the robustness of our methods to outliers, we test the methods on synthetic data. Synthetic data were created by simulating a field of 1500 image points distributed
uniformly at random depths between 2 and 10 m in front of the camera and uniformly in x and y throughout the frame. A simulated camera is moved through this field with a translational velocity drawn from a zero-mean Gaussian with standard deviation of 1 m/frame and a rotational velocity drawn from a zero-mean Gaussian with standard deviation of 0.2 radians/frame. Flow was generated from the resulting 3D point trajectories by perspective projection using a camera model with a 1 m focal length. All flow vectors were corrupted with noise in a random direction and magnitude drawn from a zero-mean Gaussian with a standard deviation $1/10$ the mean flow vector magnitude. Outliers were created by replacing a fraction of the points with random values drawn from a Gaussian fit to the magnitude and direction of all inlier flow vectors. We ran 100 iterations at each outlier rate. We ran all egomotion methods on the same data.

The errors in translational motion estimated on this data are shown in Fig. 2.8. As expected, the two robust methods outperform least-squares methods for reasonable numbers of outliers. At higher outlier rates, however, the performance of both robust methods deteriorates. Interestingly, the performance of the lifted kernel method is stable even when the majority of data points are outliers. We are uncertain why the lifted kernel performs better than ERL on synthetic data, while the opposite is true for KITTI. This difference may be due to the way the data were generated - in KITTI, outliers often reflect real structures.
in the scene and may contain some information about camera motion, but this is not the case in the synthetic data. The difference may also be due in part to the difference in depth structures in KITTI and the synthetic data. In KITTI, flow magnitude for both inliers and outliers is reflective of depth structure, and depth in real scenes is not distributed uniformly.

2.5 CONCLUSIONS

We have introduced new techniques for robust, continuous egomotion computation from monocular image sequences. We described ERL, a novel robust method that directly estimates confidence weights for the vectors of a flow field by evaluating the distribution of flow residuals under a set of self-consistent counterfactual model parameters. We also introduced a new formulation of the perspective motion equation using a lifted kernel for joint optimization of model parameters and confidence weights. We compared the results of ERL and the lifted kernel formulation, and showed that while the lifted kernel appears to be more stable in the presence of a large fraction of outliers, ERL performs better in a real-world setting. The ERL method achieves good results on KITTI without relying on stereo data or ground plane estimation and accordingly is well-suited for use in lightweight UAV navigation. We are unable to directly evaluate our methods on this target domain because there are currently no UAV datasets with suitable ground truth. Although the empirical results here are promising, we have no guarantees on the weights recovered by ERL, and this remains a topic for future work.

Our code is publicly available at https://github.com/stephenphillips42/erl_egomotion.
In chapter 2 we explored rejecting outliers using a weighing scheme. In this chapter, we will step away from direct outlier rejection and explore multi-image matching for more robust matches. As mentioned briefly in Section 1.1.2, multi-image feature matching gives more accurate and robust solutions compared to simple two-image matching for geometric computer vision applications. In this chapter, we formulate multi-image matching as a graph embedding problem, then use a Graph Neural Network to learn an appropriate embedding function for aligning image features. We use cycle consistency to train our network in an unsupervised fashion, since ground truth correspondence can be difficult or expensive to acquire. Geometric consistency losses are added to aid training, though unlike optimization based methods no geometric information is necessary at inference time. To the best of our knowledge, no other works have used graph neural networks for multi-image feature matching before ours. Our experiments show that our method is competitive with other optimization based approaches.
### 3.1 Introduction

As discussed in the Chapter 1, feature matching is an essential part of many geometric computer vision applications, particularly Structure from Motion applications. The goal in multi-image feature matching is to take 2D feature positions from three or more images and find which ones correspond to the same point in the 3D scene. Methods such as SIFT feature matching [86] combined with RANSAC [33] have been the standard for decades. Current RANSAC-based approaches are limited to matching pairs of images, which can lead to global inconsistencies in the matching. Due to the reliance on a minimal solver, it is difficult to apply RANSAC in a multi-image matching setting. Other works, such as [131], have shown improvement in performance by optimizing cycle consistency, i.e. enforcing the pairwise feature matches to be globally consistent.

However, these multi-view consistency algorithms struggle in noisy settings. Having image features suited for this task would help improve performance, and deep learning has revolutionized how image features are computed [141]. In this chapter, we want to leverage the power of deep representations in order to compute feature descriptors that are robust across multiple views.

Unfortunately, there are obstacles to applying multi-view constraints directly to deep learning. Multi-view constraints are formulated in terms of sparse features, which traditional convolutional neural nets are not designed to handle. Thus we will need a new architecture to handle such constraints. More fundamentally, deep neural networks need large amounts of labeled data to train. Consequently unsupervised training is a more practical approach. In the absence of direct supervision, the additional signal of geometric constraints can help disambiguate visually similar features and reject outliers. Thus incorporating such constraints is important in training a network to solve this task.
Figure 3.1. An illustration of the approach of this work. The Graph Neural Neural Network (GNN) [5] takes as input the graph of matches and then outputs a low rank embedding of the adjacency matrix of the graph. The GNN operates on an embedding over the vertices of the graph. In the figure, the GNN vertex embeddings are represented by different colors. The final embedding is used to construct a pairwise similarity matrix, which we train to be a low dimensional cycle-consistent representation of the graph adjacency matrix, thus pruning the erroneous matches. We train the network using a reconstruction loss on the similarity matrix with the noisy adjacency matrix, and thus do not need ground truth matches. In addition, we can use geometric consistency information, such as epipolar constraints, to assist training the network.

In this chapter, we address these concerns using Graph Neural Networks (GNNs). The proposed method works directly on the graph of correspondences between the image features, which is agnostic to how the correspondences were computed, thus allowing the algorithm to work in a broad class of environments. To the best of our knowledge this work is the first to apply graph neural networks to the multi-view feature matching problem. We use an unsupervised loss, the cycle consistency loss, to train the network. We use geometric consistency losses to aid training, though no geometric information is used at inference time. Although our network is simple, it shows promising results...
compared to baselines which optimize for cycle-consistency without learned embeddings, using a matrix factorization loss [156, 79]. Furthermore, since inference requires only a single forward pass over the neural network, our approach is faster to achieve comparable accuracy than methods which must solve an optimization problem every time. We perform experiments on the Rome16K dataset [81] to test the effectiveness of our method compared to optimization based methods. Our contributions in this chapter are:

- We use a novel architecture to address the multi-image feature matching problem using GNNs with graph embeddings.
- We introduce an unsupervised multi-view cycle consistency loss that does not require labeled correspondences to train.
- We demonstrate the effectiveness of geometric consistency losses in improving training.

3.2 BACKGROUND

3.2.1 Feature Matching

We will give a refresher on feature matching and learning, also discussed in Subsection 1.1.1. Robust two-view matching with small baselines (i.e. optical flow) was covered in detail in Chapter 2. For large baseline matching, much work has be done using hand-crafted feature descriptors such as SIFT [86], SURF [6], BRIEF [19], or ORB [92]. The combination of RANSAC [33] and hand-crafted feature descriptors has constituted the bulk of the matching literature for the last 40 years. More recently Suh et al [120] and
Hu et al [59] have shown graph matching of the features can be added for more robust matches between images.

3.2.2 Multi-image Matching

Multi-image matching has traditionally been done using optimization based methods minimizing a cycle consistency based loss (see Section 3.4.3). Pachauri et al [98] and Arrigoni et al [3] use the eigenvectors of the matching matrix to obtain a low dimensional embedding. However, the assumption of low Gaussian noise is not realistic. Zhou et al [156] and Wang et al [131] use more sophisticated optimization techniques on the matching matrix and thus produce more robust solutions. [79] implement a distributed optimization scheme to solve for cycle consistency. [122] implement a convex relaxation of the low dimensional embedding problem. Swoboda et al [116] use tensor power iterations to solve the matching problem, also taking into account the intra-image matching graph. As an alternative to optimization based techniques, [127] used density based clustering techniques to compute multi-image correspondence. Fathian et al [31] use a similar technique but formulates it as a generalized Rayleigh quotient problem to achieve better results. Moving away from feature matching, Zach et al [146] uses cycle-consistency-like constraints on pose graphs quite effectively. To the best of our knowledge, this work is the first to use graph neural networks for multi-image matching.

3.2.3 Deep Learning for Matching

We have gone over deep learning in the context of matching extensively in Chapter 1. The unique contribution of this work is the combination of graph neural networks and
multi-image matching. Zhang et al [152], for instance, use graph neural networks to do intra-image feature processing before doing two image inter-image feature similarity. However, their work only trains for pairs of image, while ours is explicitly trained for multi-image matching. Hartmann et al [52] learns multi-image matching but requires heavy supervision from 3D object reconstructions, which can be difficult or expensive to obtain. Zhou et al [155], unlike our method, uses dense correspondence, but uses cycle consistency to find semantic matches across multiple views. Suwajanakorn et al [121] find 3 dimensional latent keypoints, trained using ground truth rotation and translation. However, their method is restricted to a limited number of object categories, which is different from the SfM setting we are considering here.

Another contribution of the work in this chapter is the use of cycle consistency as a loss for a deep network; the combination has not been used in the context of multi-image matching. Like us Zhu et al [159] use cycle consistency in their loss; however their method is for image generation and is restricted to pairwise cycle consistency. Furthermore, their work is focused on image generation/transformation, not on matching.

### 3.3 Introduction to Graph Neural Networks

Graphs are a well known data structure that encodes pairwise relationships (known as edges/links) between a set of objects (known as nodes/vertices). Graph neural networks (GNNs), true to their name, are deep neural networks operating in graph domains. They are relatively new in the deep learning literature, and thus do not have as firm of a research footing as other deep networks such as convolutional neural networks. Due to their utility, they have received more research attention recently [15, 29, 74, 112, 41, 39, 5]. This chapter heavily uses graph neural networks, hence the concepts of graph neural networks are
worth expanding here. For a more in depth review, please refer to Bronstein et al [15], Zhou et al [153], Wu et al [135] for excellent overviews of the literature.

3.3.1 Motivation

There is the question as to why do we need to use graphs in a learning context. We will go over this in more technical detail in Section 3.3.3 but first some higher level motivation is in order. Traditional deep learning tends to focus on very regularly structured data that can be modelled as tensors, such as images or sound-waves. However, much data encountered in the world is not in a regular form. Some common examples given elsewhere in the literature are social networks and knowledge graphs, and chemistry/physics applications [5]. Sparse matching in geometric vision domains is the application of greatest interest of this dissertation. We have seen one such application in Chapter 2 and we will go over more specific applications in later in this chapter and Chapter 4.

These problems have structure but cannot be coerced into having the standard tensor structures of standard deep convolutional neural networks. For sparse matching, a graph structure is natural where objects/features we want to match are the nodes of the graph and putative correspondences would be edges in the graph. The problem then would be then predicting which edges are correct matches and which are incorrect. This means we need to find a way to learn over this graph structure, which gives the motivation for using graph neural networks.
3.3 Introduction to Graph Neural Networks

### 3.3.2 Background on Graphs

**Graph Definitions**

In formal notation, a graph is denoted $G = (V, E)$, with $V$ encoding the nodes and $E$ the edges. There are no restrictions on the set $V$, but in deep learning settings it typically refers to features of the nodes (e.g. features in a feature matching graph). To distinguish between the index of a node and the features of a node, we use a variables $i$ to refer the index of the node and $v_i$ to refer to the feature associated with the $i^{\text{th}}$ node. For simple graphs, the edges are simply a set of pairs of nodes, $E \subseteq V \times V$. In undirected graphs, $(i, j) \in E \iff (j, i) \in E$. As undirected graphs are special case of directed graphs, ‘graphs’ here refers to directed graphs unless stated otherwise. The degree of a node $v_i$ is the number of edges $(i, j)$ (or $(j, i)$) containing that node. We can further distinguish the degree by specifying out degree (number of edges $(i, j)$) or the in-degree (number of edges $(j, i)$).

We define the neighborhood of a node $v_i$ as $N(v_i) = \{v_j \mid (j, i) \in E\}$ i.e. nodes with an edge...
going to node \( v_i \). In addition, each graph can have node, edge, or global attributes relevant to the task – for instance, each edge can be associated with a matching score in the sparse matching problem.

**Spectral Graph Theory**

Spectral graph theory will be useful for future discussion on graphs (for a more comprehensive overview refer to Gallier [38]). The adjacency matrix of a graph \( A(G) \) is a binary matrix where \((A(G))_{i,j} = 1 \) if \((i, j) \in \mathcal{E} \) and 0 otherwise. Alternatively \((A(G))_{i,j} = w_{ij} \) if the graph is ‘weighted’ (i.e. edges have a weight attribute \( w_{ij} \)). We assume positive weights \( w_{ij} \geq 0 \), thus we can interpret the weight as the strength of connection between two nodes. The degree matrix \( D(G) \) denotes \( \text{diag}(A(G)1_{|V|}) \).

Closely related to the adjacency matrix is the graph Laplacian, defined as \( L(G) = D(G) - A(G) \). It can be shown (see Gallier [38]) that for undirected graphs:

\[
x^T L(G)x = \sum_{i,j \in \mathcal{E}} w_{ij} (x_i - x_j)^2
\]

and, thus, it follows that \( L(G) \) is positive semi-definite and with at least one zero eigenvalue with eigenvector \( 1_{|V|} \). The graph Laplacian and adjacency matrices are thus rigorous ways of encoding the structure when analyzing graphs. In practice the symmetric normalized Laplacian \( L(G) = I_{|V|} - D(G)^{-1/2}A(G)D(G)^{-1/2} \) is often used for numerical stability reasons.

Often we will want to do feature aggregation over neighboring nodes (e.g. graph convolutions in the next section). An elegant way of expressing this is using matrix
multiplication with the adjacency matrix. So if we represent the nodes $\mathcal{V}$ as a matrix, which we denote $V$, we can get the neighborhood sums using the following formula:

$$V' = A(G) V \iff v'_i = \sum_{v_j \in N(v_i)} v_j$$  (3.2)

If we want to average by degree, we can represent it as

$$V' = D(G)^{-1} A(G) V \iff v'_i = \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} v_j$$  (3.3)

Variations on these neighborhood averages will come up quite frequently when we discuss graph convolutions.

With the major definitions reviewed, we can now explore how graphs are useful in a deep learning setting.

### 3.3.3 Regular Graph Structures and Convolutions

Most standard deep learning forms can be formulated as a graph problem. For instance, time series data can be formulated as a undirected graph with $n$ nodes and edges connecting node $v_i$ to $v_{i+1}$ for all $i$. Similarly, a two dimensional grid (like an image) can be formulated as a set of $n_x \cdot n_y$ nodes $\mathcal{V} = \{(x, y) | x \in [1, n_x], y \in [1, n_y] \}$. In the 2D case, there are several ways to formulate the edges, but the simplest is $E = \{((x, y), (x', y')) | x, x' \in [1, n_x], y, y' \in [1, n_y], |x - x'| \leq 1, |y - y'| \leq 1 \}$. This can generalize to 3D, 4D, or any higher dimension, and is a graph representation of the standard tensor structure used in deep learning. Thus, when trying to learn over an input structure, almost any input structure can be represented as a graph. While there is work
Figure 3.3: Diagram of the filter weight layout of a standard convolutional neural network and graph convolutional network. Different colors correspond to different weights, with light blue being zero. In (a), due to the regular structure of the grid, we can have a very expressive weight layout. In contrast, note in (b) how due to the lack of a regular structure, we can only give weights by distance in the graph (here 1-hop distance).

...on using higher order relations (3-way, 4-way, etc) they are not used as much in practice [91].

In these cases, the highly repetitive grid structure of the edges can be exploited for standardized filters (figure 3.3a). Thus the creation of standard convolution filters, which combine the grid structure with an assumption of locality (nodes nearby are more strongly correlated). These filters are inspired by convolution defined in signal processing. This exploitation of structure and locality made significant contributions to the success of deep neural networks referred to in the introduction (Chapter 1). The full expressiveness of deep networks comes from layering these linear convolutions ‘deeply’ with point-wise non-linearities. This allows expression of much more complicated functions by increasing the region affected by the filters with each passing layer. But we need a convolutional operator to express neighborhood locality structure.
In contrast, when our domain is an arbitrary graph, we cannot rely on a regular structure to inform how we make the filters. The only information that can be used are the node and edge annotations and the number of hops from one node to another to inform locality (figure 3.3b). Thus standard definitions of convolution do not apply in this setting, and new definitions must be found. Once we have a good definition of convolution for graphs, we can use that as the linear layers and combine with non-linearities as in regular deep networks. This lack of a regular structure and search for new definitions of convolution is what motivated the initial research on graph neural networks, covered in the next section.

### 3.3.4 Types of Graph Neural Networks

Formally, our goal is to find a class of linear operator over embeddings \( V \) and graph \( G \) with parameters \( \theta \). We can then layer them to create our graph neural network with point-wise non-linearity \( \sigma \):

\[
V_t = \sigma(f(V_t, G, \theta_t))
\]  

(3.4)
For brevity we shall assume the $\mathcal{G}$ implicitly in future equations. As discussed in the previous section, a convolutional type operator is ideal due to locality. Thus, most of these operators will either be explicitly trying to mimic convolutions or be an approximation of convolutions. We will now go over various definitions of these linear operators in the literature.

**Spectral Graph Neural Networks**

The earliest work on defining convolutions on a graph used the signal processing roots of convolution. The Fourier transform is fundamental in the study of convolution. It is a linear transformation of function space, in essence change of basis to the “frequency” domain with basis vectors $e^{2\pi x}$. The Fourier basis can be derived in many ways, but one way is that they are the eigenfunctions of the Laplacian operator $\nabla^2 f$ for functions $f$ in $\mathbb{R}^d$. Convolution can be defined as point-wise multiplication in the frequency domain i.e. taking the Fourier transform $\mathcal{F}$ of two signals $f$ and $g$, and taking their point-wise multiplication, then invert the Fourier Transform:

$$ (f * g)(t) = \mathcal{F}^{-1}\{\mathcal{F}[f] \cdot \mathcal{F}[g]\} $$

(3.5)

The first methods tried to define graph convolution analogously and define a Fourier basis as the eigenvectors of the graph Laplacian.

$$ L = U\Lambda U^T $$

(3.6)

In our analogy then, $U^T$ is equivalent $\mathcal{F}$ and $U$ is equivalent $\mathcal{F}^{-1}$, with the diagonal matrix $\Lambda$ being the Fourier coefficients of the function $g$ we want to convolve. Thus to
perform graph convolution, change a function into the graph Laplacian basis then take the point-wise multiplication with the learned function \( \hat{g}(\theta) \), equivalent to \( \mathcal{F}(g) \) in our analogy

\[
L = \mathbf{U} \Lambda \mathbf{U}^T
\]

\[
f(V; \theta) = \mathbf{U} \text{diag}(g(\theta)) \mathbf{U}^T V
\]

These early methods to learn neural networks over graphs were called spectral methods, due to their use of the graph Laplacian eigendecomposition. The first work to do this as in [16].

While this is an elegant way of generalizing convolutions, it suffers from several practical problems. First, a subset of the Laplacian eigenvectors need to be used for the transform to have a tractable learning problem. The eigenvectors of the Laplacian are not localized to any one part of the graph, and thus this approximation breaks locality. Second, the eigenvalue decomposition is not stable under different graph configurations, which means the filters need to be learned on a fixed graph. Should the graph input change from instance to instance, this method would not work. However in cases where one does have a fixed graph the results can be quite effective [16].

Due to the aforementioned limitations of early spectral methods, alternatives were developed that did not require explicit computation of the Laplacian eigenvectors. One method, ChebNet [29], proposed using degree \( K \) polynomials of the Laplacian to approximate the filter using learned features \( \theta \):

\[
g(\theta) = \sum_{k=1}^{K} \theta_k \Lambda^k
\]
This has the advantage of not requiring to compute the eigenvectors and thus can work on variable graphs. The filters are also guaranteed to be localized to $K$ hops. In fact, one need not compute the actual eigendecomposition to compute these filters:

\[
\begin{align*}
\mathbf{u} \left( \sum_{k=1}^{K} \theta_k \mathbf{A}^k \right) \mathbf{u}^T = & \sum_{k=1}^{K} \theta_k \left( \mathbf{u} \mathbf{A}^k \mathbf{u}^T \right) = \sum_{k=1}^{K} \theta_k \mathbf{L}^k \\
\mathbf{V}_t = & \sigma \left( \sum_{k=1}^{K} \theta_k \mathbf{L}^k \mathbf{V}_t \right)
\end{align*}
\]

(Kipf and Welling [74] propose only using degree 1 polynomial instead of a degree $K$ polynomial i.e. only using one pass per layer. They also use a modified graph Laplacian:

\[
\tilde{\mathbf{L}} = (\mathbf{I} + \mathbf{D})^{-1/2} (\mathbf{I} + \mathbf{A})(\mathbf{I} + \mathbf{D})^{-1/2}
\]

\[
\mathbf{V}_{t+1} = \sigma \left( \tilde{\mathbf{L}} \mathbf{V}_t \theta^T \right)
\]

The authors layer this in the standard deep network way to get wider information passing for GNN through the graph. This method can replicate standard CNNs with radially symmetric filters on a grid-like graph (see Figure 3.4)

While spectral methods’ matrix representations such as the graph Laplacian are useful for analysis, there are practical concerns of how to store graphs. As matrix representations increase quadratically with the number of nodes, other representations are more suited for practical applications. One memory efficient way to store nodes and edges would be with an adjacency list (storing the nodes each node is connected to in a list), or an edge list (storing all connected pairs). These sparse representations can do operations equivalent to matrix multiplication with the adjacency/Laplacian matrix without the memory overhead. The graph neural network implementations of Battaglia et al [5] and Fey and Lenssen [32] use the edge list representation.
Non-Spectral Graph Neural Networks

For larger scale networks, similar techniques can be used but require graph sub-sampling and learnable aggregators, as in the GraphSAGE algorithm [48]. One can additionally use graph pooling to shrink the graph if the task is to do graph-wide classification [143, 40]. From this point on, we simply express operators as operations on nodes and their neighborhoods:

$$v_i^{t+1} = \sigma \left( f \left( v_i^t, \rho_v \left( \{ v_j \}_{j \in N_i} \right) \right) \right), \quad \forall i \in V$$

(3.12)

where $\rho_v$ is an aggregation operator, which could be simply the mean, sub-sampled mean, element-wise max, or even an LSTM [48, 56].

This class of graph operators, while more expressive, means that our linear operator can no longer be expressed as a matrix multiplication of the graph adjacency matrix or Laplacian. For this reason, these types of algorithms are known as non-spectral methods. While they cannot be expressed in terms of spectral graph theory, they are closely related due to the aggregation of neighborhoods. Hence, they still respect locality, and can be treated as a convolution. You can see an example of information passing around a network in Figure 3.5.
**Edge Features in Graph Neural Networks**

The observant reader might notice above that previous methods are quite restrictive with respect to the treatment of edges. Some work has been done on methods dealing with learning edge weights and features. For instance, Gama et al [39] develop a framework for node-varying filters based on node degree and edge weight connection. Their follow up work [40] extends it to create multi-dimensional edge filters. Wang et al’s work [132] learn edge features for point cloud segmentation; since the nodes are the fixed 3D points, the edges dynamically change from layer to layer in the network based on the edge features of the previous layer.

There has been work recently on learning new edge weights, thus adjusting the graph structure. These allow us to reject false edges or predict missing edges in the graph. For instance, Graph Attention Networks [130] use attention mechanisms to reweigh different nodes in the graph. Fully describing attention is beyond the scope of this chapter, but attention in neural networks is effectively learned weights over inputs, in this case graph edges. This allows rejecting spurious edges in the graph. For predicting edges in the graph, graphs auto-encoders allow for global changes in the graph structure, such as in Kipf et al’s work [73].
Figure 3.6: Diagram of Graph Nets [5]. We have node attributes $v_i$, edge attributes $e_{ij}$, and global attributes $u$. In the first step, we aggregate information from the global attribute $u$ and nodes $v_i$, $v_j$ to their edges $e_{ij}$. In the next step, we aggregate information from the global attribute $u$ and the edges $e_{ij}$ to nodes they point to $v_i$. In final step, we aggregate information from all edges $e_{ij}$ and all nodes $v_j$ to the global attribute $u$. We can repeat this with different weights at each step to build our graph network.

**Graph Networks**

More recent work has also tried learning edge features beyond just a single weight per edge [112]. Battaglia et al [5] formulate a general Graph Network framework. They learn node features $v_i$, edge features $e_{ij}$, and global features $u$ over the directed graph:

\[
\begin{align*}
    e_{ij}^{t+1} &= f_e(v_i^t, v_j^t, u^t) & \forall (i,j) \in E \\
    v_i^{t+1} &= f_v(\rho_e(\{e_{ji}\}_{(j,i) \in N_i}), u) & \forall i \in V \\
    u_i^{t+1} &= f_u(\rho_v(\{v_i\}_{i \in V}), \rho_e(\{e_{ij}\}_{(i,j) \in E}))
\end{align*}
\]  

(3.13) (3.14) (3.15)

with the $f$ being neural networks, typically multi-layer perceptrons, and the $\rho$ being aggregation functions, similar to GraphSAGE. You can see this illustrated graphically in Figure 3.6. This is the method we use for this chapter.
Figure 3.7: (a) An illustration of the idea of the universe of features. Each feature in each image corresponds to a 3D point in the scene. We can construct cycle consistent embeddings of the features by mapping each one to the one-hot vector of its corresponding 3D point. While there can be many features, there are fewer 3D points and thus this corresponds to a low rank factorization of the correspondence matrix. Best viewed in color. (b) Visualization of the learned embeddings. On the left we have the raw outputs, which are difficult to interpret. In the center, we rotated the features to best match the ground truth for a more interpretable visualization (see the end of Section 3.4.3). On the right, we have the ground truth embeddings, given as indicator vectors for which feature in the world the points correspond to. For the optimally rotated embedding we can see that the true embedding structure is recovered (with some noise).

3.4 Method

Our goal is to learn optimal features that capture multiple image views by filtering out noisy feature matches. The input to our algorithm is a set of features and noisy correspondences, and the output is a new set of features where the pairwise similarities of these features correspond to the true matches. An outline of our approach can be seen in Figure 3.1. We do this by training the new set of feature embeddings to be cycle consistent. We formulate this problem in terms of the correspondence graph of the features. Graphs $G = (V, E)$ have a set of vertices $V$ and of directed edges $E \subseteq V \times V$. For a vertex $v \in V$ we use $N^h(v)$ to denote the $h$-hop neighbors of $v$, with the superscript left out for 1-hop
neighbors. Similarly $\mathcal{E}(v)$ is used to denote the edges associated with $v$. To denote the vertices connected to an edge $e \in \mathcal{E}$ we write $e(v_1, v_2)$.

### 3.4.1 Correspondence Graph

We assume there is an initial set of feature matches represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with an associated adjacency matrix $A$. The graph is constructed from putative correspondences of image features across images, typically constructed using feature descriptor distance (e.g. SIFT feature distance). While there are many interesting methods for computing these putative correspondences [120, 142], we do not explore them in this chapter. Typically putative correspondences are matched probabilistically, meaning a feature in one image matches to many features in another. The ambiguity in the matches could come from repeated structures in the scene, insufficiently informative low-level feature descriptors, or just an error in the matching algorithm. Filtering out these noisy matches is our primary learning goal.

Each vertex of the graph $v \in \mathcal{V}$ is an image feature, corresponding to some ground truth 3D point $p(v)$. Each edge $e = (v_1, v_2) \in \mathcal{E}$ is a potential correspondence. Associated with each vertex $v$ is an embedding $f_v \in \mathbb{R}^m$, which can include the visual feature descriptor, position, scale, orientation, etc. Similarly, each edge $e$ has an associated feature $f_e \in \mathbb{R}^p$ (for this method, initially just the weight of the feature association). We use these features as the initialization for our learning algorithm.

In the absence of noise or outliers, this graph would have a connected component for each visible point in the world, all mutually disjoint. Without noise, vertices $v$ would only match with other vertices $v'$ that correspond to the same 3D point in the scene. Since features in this case represent unique locations in the scene, no points in the same image would have edges $e$ between them. Mathematically, this can be expressed as
e = (v_1, v_2) \in E \implies P(v_1) = P(v_2). \text{ In the noisy case we expect this structure to be corrupted, i.e. there are some edges } e = (v_1, v_2) \in E \text{ such that } P(v_1) \neq P(v_2). \text{ Thus we need to prune the erroneous edges.} 

However, standard CNNs cannot operate on this general graph structure. Thus we cannot use standard convolutional nets to learn features for this task. Instead we use graph networks to learn feature representations on this space, which we describe in the next section.

### 3.4.2 Graph Neural Networks for Feature Matching

As input to our method we are given a graph \( G = (V, E) \) with the features described in Section 3.4.1: \( f_v \ \forall v \in V \) and \( f_e \ \forall e \in E \). As with any neural network, GNNs have layered outputs. We describe the output of layer \( k \) as \( f_v^{(k)} \in \mathbb{R}^{m_k} \ \forall v \in V \) and \( f_e^{(k)} \in \mathbb{R}^{p_k} \ \forall e \in E \), with the initial embeddings denoted \( f_v^{(0)} = f_v \) and \( f_e^{(0)} = f_e \). To aid future analysis, we will represent the features as matrices, denoting the vertex embedding matrix as \( F_V^{(k)} \) and the edge embedding matrix as \( F_E^{(k)} \). If a superscript is not specified then it refers to the final output of the network.

First we describe older methods of GNNs to give context, then we describe the method we use in this work. Many older methods assume we have the adjacency matrix \( A \) of the graph known a-priori [16], and can encode graph convolutions using the eigenvectors of \( A \) (these are known as spectral methods). However, we do not have this luxury, as the correspondence structure changes from image set to image set, and thus we use non-spectral Graph Neural Networks. Newer models use non-spectral methods, which often ultimately amount to transforming each node with learned weights then averaging each node’s representation with its neighbors, known as a message pass [74, 29, 41, 39]. Some works such as [40] use pooling operations on the vertices to make the graph smaller.
and thus aid computation, but as we need labels on every vertex of the original graph, we cannot use this. Most GNNs used in these works can be expressed mathematically as:

\[
\tilde{f}_{v}^{(k+1)} = \sigma \left( b^{(k)} + W^{(k)}_0 f_v^{(k)} + \sum_{h=0}^{H} \sum_{v' \in \mathcal{N}^h(v)} f_{e(v,v')} W^{(k)}_{h,v'} f_{v'}^{(k)} \right)
\]

The weights/biases \( W^{(k)}_{h,v'} \), \( b^{(k)} \) are all learned, with no learning done on the edge weights \( f_{e(v,v')} \). Note that this is just averages over \( h \)-hop neighborhoods, where the weights on the edges remain static through the computation. Given that we are trying to prune edges, we add features over edges to learn which ones to prune and which to keep such as in [112]. Comparing to graph attention networks [130], this allows us to learn more flexible features on the edges.

In this work we use the method and implementation described in Battaglia et al [5]. Battaglia et al [5] uses message passing between node features as well as edge features, which the model can use to prune unnecessary or erroneous edges. Therefore there is intermediate processing on the edges before information is passed to the vertices.

Mathematically, this is expressed as:

\[
\tilde{f}_{e(v_1,v_2)}^{(k+1)} = \sigma \left( a^{(k)} + U^{(k)}_0 f_{e}^{(k)} + U^{(k)}_1 f_{v_1}^{(k)} + U^{(k)}_2 f_{v_2}^{(k)} \right) \quad (3.16)
\]

\[
\tilde{f}_{v}^{(k+1)} = \sigma \left( b^{(k)} + W^{(k)}_0 f_v^{(k)} + \sum_{e \in \mathcal{E}(v)} W^{(k)}_1 f_e^{(k+1)} \right) \quad (3.17)
\]

Here the learned weights are denoted \( W \) and \( U \), and the biases \( a^{(k)} \) and \( b^{(k)} \). In [5], they allow for more sophisticated aggregation functions, but in this work we simply use the mean function. Each one of these steps we refer to here as a message pass, and it is
analogously equivalent to an iteration in a distributed graph based optimization method. Between each of the message passes, we further process the features using MLPs.

3.4.3 Cycle Consistency

Let $M$ be the noiseless set of matches between our features, with $M_{ij}$ being the partial permutation representing the matches between image $i$ and image $j$. If the pairwise matches are globally consistent, then for all $i, j, k$:

$$M_{ij} = M_{ik}M_{kj}$$

In other words, the matches between two images stay the same no matter what path is taken to get there. This constraint is known as cycle consistency, and has been used in a number of works to optimize for global consistency [156, 131, 79]. Stated in this form, there are $O(n^3)$ cycle consistency constraints to check. A more elegant way to represent cycle consistency is to first create a ‘universe’ of features that all images match to (see Figure 3.7a). Then, one can match the $i^{th}$ set of features to the universe using a ground-truth matching matrix $X_i$. Then the cycle consistency constraint becomes:

$$M_{ij} = X_iX_j^T$$

This reduces the number of our constraints from $O(n^3)$ to $O(n^2)$. This was shown to be equivalent to the original definition of cycle consistency (Equation 3.18) in [62]. We try to learn vertex embeddings $F_V$ to approximate $X$ - in other words the final embedding should be an encoding of the universe of features. As we do not have the ground truth
matches $M$, we approximate it using the noisy adjacency matrix $A$ of our correspondence graph. Thus our loss would be

$$\mathcal{L}(A, F_V) = D(A, F_V F_V^\top)$$

(3.20)

Here $D$ could be an $L_2$ loss, $L_1$ loss, or many others. In this work, we use the $L_1$ loss. Note that because of this formulation, we can determine our embeddings only up to a rotation, as $F_V R (F_V R)^\top = F_V R R^\top F_V = F_V F_V^\top$. Thus when visualizing embeddings, we rotate them to make them more interpretable (see Figure 3.7b).

### 3.4.4 Geometric Consistency Loss

In order to use geometric information, more traditional optimization based methods require the geometric information at inference time, while with learning approaches we can use it to speed up training while not needing it at inference time. Thus geometric consistency losses are one distinct advantage of our method over more traditional optimization based approaches. We use the epipolar constraint, the simplest way to add a geometric consistency loss. The epipolar constraint describes how the positions of features in different images corresponding to the same point should be related. An illustration of this is provided in Figure 3.8a, showing how this loss can help reject erroneous points. Given a relative pose $(R_{ij}, T_{ij})$ between two cameras $i$ and $j$ (transforms $j$ to $i$) the epipolar constraint on corresponding feature locations $X_i$ and $X_j$: $X_i^\top [T_{ij} \times R_{ij}] X_j = 0$. In this work we use the two pose epipolar constraint [126]:

$$X_i^\top R_i^\top [T_j - T_i] \times R_j X_j = 0$$

(3.21)
Illustrated here is an example of how the geometric loss is computed for one feature.

Figure 3.8: (a) Errors are computed via absolute distance from the epipolar line, as expressed by Equation 3.21 via the epipolar constraint. The epipolar line is the line of projection of the feature in the first image, projected into to the second. The distance to this line on the second image indicates how likely that point is to correspond geometrically to the original feature. There can be false positives along the projected line, as shown by the square feature in the figure, but other points will be eliminated, such as the hexagonal feature. (b) Training curves with and without Geometric Training loss, described in Equation 3.23. The geometric training loss improves testing performance. Note how training with geometric consistency losses decreases the convergence time of the network. Best viewed in color.

The constraint assumes that the $X_k$ are calibrated (i.e. the camera intrinsics are known). Given our vertex embeddings matrix $f_v$, we can formulate a loss between all cameras $i$ and $j$ (the vertices associated with camera $i$ denoted $V(i)$):

$$
\mathcal{L}_{ij,\text{geom}}(F_V) = \sum_{v \in V(i), u \in V(j)} |(f_v \cdot f_u) X_v^\top R_i^\top [T_j - T_i] \times R_j X_u|
$$

(3.22)
<table>
<thead>
<tr>
<th>Method</th>
<th>Same Point Similarities</th>
<th>Different Point Similarities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>1.000 ± 0.0000</td>
<td>0.0000 ± 0.0000</td>
</tr>
<tr>
<td>Initialization Baseline</td>
<td>0.511 ± 0.0168</td>
<td>0.2560 ± 0.2060</td>
</tr>
<tr>
<td>5 Views, Noiseless</td>
<td>1.000 ± 0.0004</td>
<td>0.1220 ± 0.1670</td>
</tr>
<tr>
<td>6 Views, Added Noise</td>
<td>0.984 ± 0.0031</td>
<td>0.0746 ± 0.1570</td>
</tr>
<tr>
<td>3 Views, 5% Outliers</td>
<td>0.929 ± 0.1790</td>
<td>0.1410 ± 0.1480</td>
</tr>
<tr>
<td>3 Views, 10% Outliers</td>
<td>0.927 ± 0.1790</td>
<td>0.1400 ± 0.1510</td>
</tr>
</tbody>
</table>

Table 3.1: Results for unsupervised training on synthetic data under various noise conditions. The table plots out the weights (mean and standard deviation) of the edges reconstructed by our model, for true positive matches and true negative ones. This shows under various noise conditions that our architecture can still recover the original connectivity structure of the matching graph.

For our purposes, since we use low rank embeddings $F_v$, the loss would read (where $c(k)$ is the appropriate camera for point index $k$):

$$
\mathcal{L}_{geom}(F_v) = \text{tr}(G^T F_v F_v^T) = \sum_{k,l} (F_v)_k \cdot (F_v)_l^T (G)_{kl}
$$

$$(G)_{kl} = \begin{vmatrix} X_k^T R_{c(k)}^T [T_{c(l)} - T_{c(k)}] x R_{c(l)}^T X_l 
\end{vmatrix}
$$

3.5 EXPERIMENTS

3.5.1 Synthetic Graph Dataset

We first test our method on synthetically generated data as a simple proof of concept. As these were simpler datasets, we use the simpler edge-feature free model of [74]. To generate the data, we generate $p$ points, each with its own randomly generated descriptor. To create the graph, we generate random permutation matrices, with a noise applied to it
after it is generated. We initialize the input descriptors using the synthetically generated
ground truth descriptor, plus some added Gaussian noise. No geometric losses were added
during training for these experiments. However, the method was robust in testing with
different noise functions and parameters. The normalized noisy input descriptors are
our baseline - they correlate with the true values but do not preserve the structure well.
However, the GNN recovered the true structure very well, as shown in Table 3.1, showing
the appropriate edge similarities. With this simple test on synthetic data passed, we now
move to more challenging datasets.

### 3.5.2 Rome 16K Graph Dataset

We use the Rome16K dataset [81] to test our algorithm in real world settings. Rome16K
consists of 16 thousand images of various historical sites in Rome extracted from Flickr,
along with the 3D structure of the sites provided by bundle adjustment. While not a
standard dataset to test cycle consistency, most standard datasets have tens or hundreds
images, not enough to train a GNN on. Rome16K is typically used to test bundle adjustment
methods. Therefore, to use our method, we extract 6-tuples and 10-tuples of images
with overlap of 80 points or more to test our algorithm, with the points established as
corresponding in the given bundle adjustment output. For the initial embedding we use the
original 128 dimensional SIFT descriptors, normalized to have unit $L_2$ norm, the calibrated
x-y position, the orientation, and log scale of the SIFT feature. To calibrate we use the focal
lengths provided in Rome16K’s data, and we assume the image center is in the center of
the image (as none is provided by Rome16K). To construct the graph, we take each feature
as a vertex and create edges to the 5 nearest SIFT descriptors for the other images.

All experiments were run with a 12 layer GNN with the ReLU non-linearity and skip
connections. The feature vector lengths were $32, 64, 128, 256, 512, 512, 512, 512, 512$,
Figure 3.9: Plot of the losses of the baselines at different iteration numbers. The line shows the mean of the graph while the translucent coloring shows the 25th to 75th percentiles. The ROC AUC curves remain fairly consistent while the L1 loss goes noticeably down after more iterations. Our method compares to 35-45 iterations of MatchALS, while only having 8 message passes. PGDDS performs better than us in L1 but we perform similarly in the ROC AUC metric. These results still hold even when we change domains to the Graffiti dataset (see 3.5.3).

1024, 1024, with skip connections between layers 1 and 6, 6 and 12, and 1 and 12. All were trained with the Adam optimizer [72] and a learning rate $10^{-4}$ The network was implemented in Tensorflow [1], version 1.11.

For these experiments we train with the L1 norm and geometric consistency losses. We evaluate on a test set using the ground truth adjacency matrix, which we compute from the bundle adjustment given by the Rome16K dataset. However, we do not train with the ground truth adjacency matrix, only with a noisy version of the adjacency matrix. We also add the geometric loss (3.23) which helps improve testing performance (see Figure 3.8b). We use the L1 and ROC AUC metrics to measure performance. For this method to work, we need the dimension of the embedding to be at least the number of unique points in
the scene. Picking the correct number is difficult a-priori, and is a problem with all cycle consistency based methods. Here we use the ground truth dimension of the embedding to test both our method and the baselines.

The network was implemented using the code provided by [5] using Tensorflow 1.11 [1]. Our network has 16 layers, with 8 message passing operations placed every other layer. All layers were simple Multi-layer Perceptron’s, with no batch norm. The network was trained with the Adam optimizer [72] with a learning rate of $10^{-4}$, with an exponentially decaying learning rate. We incorporate skip connections between the input, 6th, and 12th layers (all possible pairs).

We compare our method to spectral and optimization based baselines with different maximum iteration cutoffs. Figure 3.9 illustrates this by plotting the means of various metrics and their 25th and 75th percentiles, with Table 3.2 giving the exact numbers. Our network, though only using 8 message passes, has comparable accuracy to MatchALS [156] run 35 to 45 iterations, with an equivalent message passing step at each phase. Although our method does not outperform the Projected Gradient Descent - Doubly Stochastic (PGDDS) [79] method, we perform comparably to them in the ROC AUC metric.

3.5.3 Computing Geometric Loss

To compute the Geometric Loss during training we use the given rotations and translations from Rome16K to compute the relative poses. In practice, one could just use the relative poses computed. We build up an intermediate Essential Matrix representations $P, Q \in \mathbb{R}^{3pv \times 3}$, with $v$ being the number of views and $p$ being the number of points. The matrix $P$ is defined by the block matrix representation $P_i = [T_{c(i)}] \times R_{c(i)} X_i$. The matrix $Q$ is defined
by the block matrix representation $Q_i = R_{c(i)}X_i$. We build the Essential matrices using $E = Q^TP + P^TQ$, which has 3 by 3 blocks

$$E_{ij} = (R_{c(i)}X_i)^T[T_{c(j)}]xR_{c(j)}X_j + ([T_{c(i)}]xR_{c(i)}X_i)^TR_{c(j)}X_j$$

$$= X_i^TR_{c(i)}^T[T_{c(j)}]xR_{c(j)}X_j - X_i^TR_{c(i)}^T[T_{c(i)}]xR_{c(j)}X_j$$

$$= X_i^TR_{c(i)}^T[T_{c(j)} - T_{c(i)}]xR_{c(j)}X_j$$

Note that this is the same as Equation 3.21 and thus we have our relative pairwise constraints.

**Graffiti Dataset**

We run our trained model on the more Graffiti Dataset from the Affine Covariant Regions dataset (formatted to be able to be input to our model properly). The Graffiti Dataset is the most common benchmark used in feature matching algorithms (e.g. [79, 156]). The results are shown in Figure 3.9 in the rightmost figure. As the graffiti dataset is very small (only 6 views total), we were not able to train on it. We randomly permute the intra-image order of the features to add some variance - by design the GNN outputs the same result each time, while the optimization methods have a very small amount of variance. The transferred results of Graffiti are similar to the test error of Rome16K - smaller $L_1$ error and comparable ROC error. This shows that the GNNs trained in Rome16K generalize similarly to the optimization based methods.
3.6 CONCLUSION

We have shown a novel method for training feature matching using GNNs, using an unsupervised cycle consistency loss and geometric consistency losses. We have demonstrated end-to-end trainable GNNs have comparable performance to the traditional optimization-based baselines.
<table>
<thead>
<tr>
<th>Method (6 Views)</th>
<th>L₁</th>
<th>L₂</th>
<th>AUC ROC</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchALS 15 Iter.</td>
<td>0.101 ± 0.008</td>
<td>0.022 ± 0.004</td>
<td>0.918 ± 0.073</td>
<td>0.074 ± 0.008</td>
</tr>
<tr>
<td>MatchALS 35 Iter.</td>
<td>0.046 ± 0.016</td>
<td>0.010 ± 0.005</td>
<td>0.910 ± 0.072</td>
<td>0.139 ± 0.041</td>
</tr>
<tr>
<td>MatchALS 50 Iter.</td>
<td>0.029 ± 0.017</td>
<td>0.008 ± 0.005</td>
<td>0.905 ± 0.068</td>
<td>0.260 ± 0.048</td>
</tr>
<tr>
<td>PGDDS 15 Iter.</td>
<td>0.017 ± 0.002</td>
<td>0.007 ± 0.001</td>
<td>0.918 ± 0.087</td>
<td>0.796 ± 0.147</td>
</tr>
<tr>
<td>PGDDS 25 Iter.</td>
<td>0.016 ± 0.002</td>
<td>0.007 ± 0.002</td>
<td>0.919 ± 0.087</td>
<td>1.670 ± 0.328</td>
</tr>
<tr>
<td>PGDDS 50 Iter.</td>
<td>0.015 ± 0.002</td>
<td>0.006 ± 0.002</td>
<td>0.920 ± 0.087</td>
<td>3.363 ± 0.528</td>
</tr>
<tr>
<td>Spectral</td>
<td>0.073 ± 0.006</td>
<td>0.027 ± 0.003</td>
<td>0.921 ± 0.083</td>
<td>0.036 ± 0.005</td>
</tr>
<tr>
<td><strong>GNN (ours)</strong></td>
<td><strong>0.044 ± 0.005</strong></td>
<td><strong>0.031 ± 0.005</strong></td>
<td><strong>0.872 ± 0.081</strong></td>
<td><strong>0.765 ± 0.046</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method (10 Views)</th>
<th>L₁</th>
<th>L₂</th>
<th>AUC ROC</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchALS 15 Iter.</td>
<td>0.114 ± 0.008</td>
<td>0.028 ± 0.004</td>
<td>0.915 ± 0.051</td>
<td>0.142 ± 0.009</td>
</tr>
<tr>
<td>MatchALS 35 Iter.</td>
<td>0.065 ± 0.009</td>
<td>0.013 ± 0.003</td>
<td>0.907 ± 0.053</td>
<td>0.355 ± 0.073</td>
</tr>
<tr>
<td>MatchALS 50 Iter.</td>
<td>0.045 ± 0.012</td>
<td>0.011 ± 0.004</td>
<td>0.914 ± 0.051</td>
<td>0.455 ± 0.022</td>
</tr>
<tr>
<td>PGDDS 15 Iter.</td>
<td>0.017 ± 0.001</td>
<td>0.008 ± 0.001</td>
<td>0.903 ± 0.061</td>
<td>1.225 ± 0.159</td>
</tr>
<tr>
<td>PGDDS 25 Iter.</td>
<td>0.016 ± 0.001</td>
<td>0.007 ± 0.001</td>
<td>0.904 ± 0.061</td>
<td>2.637 ± 0.357</td>
</tr>
<tr>
<td>PGDDS 50 Iter.</td>
<td>0.016 ± 0.001</td>
<td>0.007 ± 0.001</td>
<td>0.905 ± 0.061</td>
<td>6.116 ± 1.009</td>
</tr>
<tr>
<td>Spectral</td>
<td>0.073 ± 0.005</td>
<td>0.029 ± 0.002</td>
<td>0.912 ± 0.057</td>
<td>0.081 ± 0.021</td>
</tr>
<tr>
<td><strong>GNN (ours)</strong></td>
<td><strong>0.053 ± 0.006</strong></td>
<td><strong>0.035 ± 0.005</strong></td>
<td><strong>0.872 ± 0.061</strong></td>
<td><strong>2.438 ± 0.070</strong></td>
</tr>
</tbody>
</table>

Table 3.2: Results on Rome16K Correspondence graphs, showing the mean and standard deviation of the L₁ and L₂. Our method was not trained on ground truth correspondences but using unsupervised methods and geometric side losses. Thus we test against ground truth correspondence graph adjacency matrices computed from the bundle adjustment output. Our method performs better than 35 iteration of the MatchALS [156] method, but does not perform as well as 50 iterations. We perform better than a simple eigenvalue based method [98]. Note that we perform much better in L₁ performance rather than L₂, as we optimized the network weights using an L₁ loss.
The general problem of matching in the presence of outliers, while old and well studied, is still an active area of research \[33, 115, 86, 149, 136\]. As discussed in Chapter 1, many applications require dealing with robust matching such as two-view image matching, multi-view matching, point cloud alignment, pose graph optimization, etc. Some of the earlier applications for robust matching were in multi-view geometry for matching features in multiple images \[51\]. However, these applications’ performance suffer greatly when inputs deviate from the expected model. When matches fit the desired model (called “inliers”) up to random white noise, most matching problems have fairly straightforward solutions. In most practical settings, there are matches that deviate arbitrarily from the expected model (“outliers”), which are much harder to model. Brute force matching or Branch-and-bound methods lead to exponential running times, so faster algorithms are needed. However, beyond the most basic settings, robust fitting in the presence of outliers is NP-hard \[22\], and in fact inapproximable \[128\].

However, while a fast and general solution to the matching problem is impossible, there have been many algorithmic paradigms developed to find good approximate solutions. Many of these have been reviewed in Chapter 1. One paradigm is the well known Random Sample Consensus (RANSAC) \[33\], which takes random subsets and a minimal solver to find the largest inlier set probabilistically. An alternative paradigm, robust optimization,
tries to minimize a cost function accounting for large deviations from outliers [57, 145], with some more recent work being able to handle much larger outlier rates [128, 136].

Many recent works on robust matching have focused on using deep learning [148, 46, 52, 142, 13]. Deep learning uses powerful function approximators to model the distribution of data in practical, real-world contexts, giving more accurate solutions in that context. Examples include two-view matching [142] and point cloud matching [25]. In these examples among others, leveraging training data to distinguish inlier and outlier data clearly benefits practical performance.

Both deep learning and robust optimization have their complementary strengths, and combining them one can draw from best of both methods. For instance, standard deep learning techniques don’t provide much in the way of guarantees for hard constraints, and thus most work either projects the output to the valid solution space or only uses soft constraints. Drawing from the optimization literature, recent work has shown ways of enforcing constraints over the expected output of Deep Networks in using alternative training paradigms [30, 21, 93], specifically using primal-dual training techniques. These techniques allow for a combination of the advantages of deep learning and robust optimization with constraint satisfaction.

In this work we hope to combine the work from the robust optimization literature and deep learning literature to create a general framework for robust matching. It can be used to enhance performance in existing deep learning frameworks or to improve performance in robust learning frameworks. We leverage the aforementioned primal-dual training techniques to learn more robust matching estimators. Our contributions are:

- A Lagrangian primal-dual training framework for robust matching problems in a semi-supervised setting
• Theoretical results proving that the outlier-rejection problem can satisfy the outlier-rejection constraints.

• Experiments demonstrating the method in a variety of settings.

4.2 BACKGROUND

4.2.1 Robust Matching

We have reviewed the matching problem in detail in Chapter 1. Given the importance of the robust matching problem, many practical algorithms have been developed [33, 65, 145, 67, 141], and are relevant to the subject in this chapter.

Also reviewed in Chapter 1, robust optimization methods attempt to minimize the errors of data points generated by inliers while limiting the effect of outliers on the result. Such methods have been used very successfully in various domains of computer vision [45, 7]. One approach is to find a convex relaxation of the original matching problem [89]. While there is no guarantee the solution to relaxed problem will be close to the original, this approach can achieve good results and often can be used as an initialization for a more challenging optimization problem. Constraints can be used to add robustness to the optimization problem as well [11, Ch. 6.4]. In particular, employing Lagrangian Duality Theory, we can find the global solution for many constrained convex optimization problems. We will cover this in more detail in Section 4.2.3.

Another robust optimization paradigm is the lifted kernel approach, which optimizes robust kernels more easily than direct optimization by ‘lifting’ the problem by adding weight dimensions to the optimization [145, 147, 161]. Building off of lifted kernel ap-
proaches, some recent work has strived to handle much larger outlier rates with theoretical guarantees. Tzoumas et al [128] used an adaptive trimming algorithm (ADAPT) to select the inliers, with a per-instance bound of sub-optimality. Yang et al [138] created a robust point cloud matching solver by creating invariant measures with adaptive voting to select inliers. Their later work [136] uses gradually increasing non-convex loss functions to down-weight outliers. These works inspired the robust loss functions of this chapter.

4.2.2 Robust Optimization with Deep Learning

Here we focus on robust matching and deep neural networks, rather than general deep learning in matching gone over in Chapter 1. While deep learning has been used for generic outlier detection [20], most work in deep learning focuses on solving more narrowly-scoped problems. Choy et al [25] solve the particular problem point cloud matching using deep learning to learn good features for inlier detection. Yi et al [142] use DNNs for two-view matching and inlier detection, but focus on RANSAC rather than constraint satisfaction. Brachmann and Carsten [13] expand on this and use reinforcement learning techniques to improve the inlier detection on an initialized network – our work assumes no such pre-training, or could be used as an alternative pre-training to this network. Zhu et al [158] similarly use deep learning for improving robust estimation via RANSAC for disparity and optical flow estimation. Hua et al [60] use a differential keypoint estimator for robust 6DOF object pose estimation, trying to avoid the outlier problem by learning robust keypoints.

Our work focuses on building a general framework for many matching tasks, rather than tackling individual sub-problems. Chalapathy et al [20] overview deep learning techniques used for anomaly detection, but none are used in the context of matching. Lathuiliére et al [78] create a general framework by using a Gaussian-Uniform mixture model to perform EM estimation with Deep Networks for learning robust regression or
classification. However their goal is to learn a robust deep model rather learn weights for a robust regression.

4.2.3 Lagrangian Duality

To contextualize the mathematics covered in Section 4.3, we cover very briefly the rich field of Lagrangian duality. For a more complete overview, please refer to Boyd and Vandenberghe [11]. Consider a standard minimization problem with constraints, which is called the primal problem:

$$\min_{x} f(x) \text{ subject to } g_i(x) \leq 0, \ i = 1, \ldots, m$$  \hspace{1cm} (4.1)$$

We denote the set of x that satisfy the constraints as $C = \{x \mid g(x) \leq 0 \forall i = 1, \ldots, m\}$. The minimal value of the primal problem is called $P^*$. The constraints can cause some difficulty in the optimization, as it must be ensured that x is feasible i.e. satisfies the constraints. To optimize this using standard techniques like gradient descent, one would have to find a feasible initial point and project to the feasible set each step. One can relax this and simply penalize violation of the constraints with positive coefficients $\lambda_i$:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \text{ subject to } \lambda_i \geq 0, \ i = 1, \ldots, m$$  \hspace{1cm} (4.2)$$

This is known as the Lagrangian of the original problem 4.1. For a given set of $\lambda$, we can minimize this as an unconstrained minimization problem. We can express this as a function of $\lambda$.

$$d(\lambda) = \min_{x} \left( f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \right)$$  \hspace{1cm} (4.3)$$

This is known as the dual of the original primal problem.
It is not hard to show that the dual is a lower bound to the primal. Consider a feasible point \( \tilde{x} \in \mathcal{C} \):

\[
d(\lambda) = \min_x \left( f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \right)
\]

\[
\leq f(\tilde{x}) + \sum_{i=1}^{m} \lambda_i g_i(\tilde{x})
\]

\[
\leq f(\tilde{x})
\]

Thus, for any \( \lambda \), the dual is a lower bound to a feasible \( x \):

\[
d(\lambda) \leq f(x), \quad \forall x \in \mathcal{C}, \lambda \geq 0 \tag{4.4}
\]

In particular, maximizing the dual \( d \) with respect to \( \lambda \) can give us a lower bound to the optimal feasible value of primal \( f \):

\[
D^* = \left( \max_{\lambda \geq 0} d(\lambda) \right) \leq \left( \min_{x \in \mathcal{C}} f(x) \right) = P^* \tag{4.5}
\]

The difference between the optimal primal and dual values \( P^* - D^* \) is known as the duality gap. In general, the duality gap is greater than zero. However, constrained convex optimization problems that are strictly feasible have zero duality gap [11]. Typically these problems have well defined forms for the primal and dual and both are feasible to compute. In those cases, we can find an exact optimal solution and verify we are at the solution by inspecting the duality gap.

This is typically done with convex solvers. While going over all the different types of convex solvers is beyond the scope of this overview, we will go over primal-dual solvers. Primal-dual solvers can converge to the optimal solution very quickly if the problem is in the right form. A very popular example of this is the Alternating Direction Method of
Multipliers, which assumes that the function you are optimizing is of the form \( f_1(x) + f_2(x) \). Instead of optimizing directly, it optimizes with respect to \( x \) and dummy variables \( z \) and adds the constraint \( x - z = 0 \), and uses alternating primal and dual minimization to solve the problem. However, in this work, we will be using deep neural networks, which are highly non-convex and thus must use the general form of primal-dual optimization.

Primal-dual solvers optimize the duality gap by alternating maximizing the dual and minimizing the primal:

\[
\max_{\lambda \geq 0} \min_x \mathcal{L}(x, \lambda) = \max_{\lambda \geq 0} \min_x f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)
\]

This can be done using simple gradient descent with learning rates \( \alpha_\lambda \) and \( \alpha_x \):

\[
\begin{align*}
\lambda_{t+1} &= [\lambda_t + \alpha_\lambda \nabla_\lambda \mathcal{L}(x_t, \lambda_t)]_+ = [\lambda_t + \alpha_\lambda g(x_t)]_+ \\
x_{t+1} &= x_t + \alpha_x \nabla_x \mathcal{L}(x_t, \lambda_t) = x_t + \alpha_x \left( \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) \right) 
\end{align*}
\]

(4.6)

Here \( g(x_t) = (g_1(x_t), \ldots, g_m(x_t))^T \), the vector of the constraint violations, with \([ \cdot ]_+ = \max(0, \cdot)\) enforcing \( \lambda_i \geq 0 \). In lay terms, this means we minimize \( x \) while allowing violations of the constraints, but increase the penalty terms \( \lambda_i \) of constraints that are violated. The goal is to reach a saddle point, a minima for \( x \) and maxima for \( \lambda \).

4.2.4 Deep Learning with Constraints

In deep learning, constraints are mostly enforced either softly using penalty terms or directly using the architecture (e.g. softmax, sigmoid). In some recent works, and the approach taken here, a Lagrangian dual formulation (covered in the previous section) is used to enforce constraints on the expected output of deep neural networks. One recent
work by Nandwani et al [93] formulates an optimization framework using a hinge function to learn hard constraints, and proves that the method converges to a local saddle point. Here we draw on the work of Eisen et al [30] and Chamon et al [21], which avoid the use of the hinge function for better theoretical guarantees on the duality gap of the problem.

In particular, their works specifically focus on supervised classification problems over continuous functions $\Phi$:

\[
P^* = \min_{\phi \in \Phi} \mathbb{E}_{(x, y) \in \mathcal{D}} [f_0(\phi(x), y)]
\]

subject to $\mathbb{E}_{(x, y) \in \mathcal{D}} [f_k(\phi(x), y)] \leq 0, \ \forall k = 1, \ldots, K. \quad (4.7)$

Thus they formulate the Lagrangian and dual:

\[
D^* = \max_{\lambda \geq 0} \min_{\phi \in \Phi} \mathbb{E}_{(x, y) \in \mathcal{D}} [f_0(\phi(x), y)] + \sum_{k=1}^{K} \lambda_k \mathbb{E}_{(x, y) \in \mathcal{D}} [f_k(\phi(x), y)]
\]

In Eisen et al [30] they use Equation 4.6 to optimize a deep neural network to approximate $D^*$.

Chamon et al [21] show that the duality gap $P^* - D^*$ is zero assuming an infinite capacity network $\phi$ and infinite data. We prove similar results in Section 4.3.3. For function approximators $\phi$ that approximate any function with less than $\epsilon$ error and with $N$ samples (i.e. in more practical settings), they define the empirical duality gap:

\[
D_{\epsilon,N} = \max_{\lambda \geq 0} \min_{\phi \in \Phi} \sum_{i=1}^{N} f_0(\phi(x_i, \theta), y_i) + \sum_{k=1}^{K} \lambda_k \sum_{i=1}^{N} f_k(\phi(x_i, \theta), y_i)
\]

Specifically, they with probability $1 - \delta$ bound the empirical duality gap by:

\[
|D_{\epsilon,N} - P^*| \leq \|1 + \lambda_\epsilon\| \mathbb{E} - 2\sqrt{N^{-1} \left[1 + \log \left(4\delta^{-1}(2N)^{d_{\text{VC}}}ight)\right]} \quad (4.8)
\]
To obtain this bound they assume the functions are L-Lipschitz, with $d_{VC}$ being the VC dimension of the problem space.

4.3 METHODS

4.3.1 Trimmed Least Squares

In this work, we formulate outlier rejection specifically as a minimally trimmed least squares problem (similar to [128, 136]). We are given data $D = \{(x_k, I_k, y_k)\}_{k=1,...,n}$. Here $(x_k, y_k)$ would be information we have a good model for, and $I_k$ would be any additional information that could be useful for our task (for example images or local features). The collection of all $x_k$ is denoted $X$, similarly for $Y$, and $I$. For notational simplicity, we will not write out $I$ in the discussions below but note that it is valid to add it wherever the $X$ is present.

Our goal is to find the parameters $W$ of our model to predict the relationship between $X$ and $Y$, with true model $W^*$. The error of this model is denoted $h(x_k, y_k, W)$ (for simplicity, it is assumed $h$ does not vary with $k$). However, we assume much of the measurement pairs $(x_k, y_k)$ are outliers and do not fit our model of the data. If we knew the ground truth model parameters $W^*$, the true inliers could be found by thresholding ("trimming") the errors of each measurement pair by $\epsilon$. As we need to find both the inliers and the parameters, there is an additional need to regress selection variables $s$ to determine inliers.
This optimization problem is known as trimmed least squares, and can be written as follows (with hyper-parameter $\epsilon$):

$$\max_{\{s_k\}_{k=1}^{m}, W} \sum_{k=1}^{m} s_k$$

subject to

$$\sum_{k=1}^{n} s_k \| h(x_k, y_k, W) \|^2 \leq \epsilon$$

$$s_k \in \{0, 1\}, \forall k \in 1, \ldots, n$$

As stated in [128], this problem is extremely difficult to solve directly with any guarantees. While one cannot find the global optima for this problem, it is possible to leverage the representational power of deep neural networks to find good solutions for data seen in practice.

4.3.2 Deep Learning for Outlier Classification

One way to approach this would be to use full supervision to predict the outliers given a dataset of regression examples with ground truth labels for the selection variables $s_{gt}$, denoted $D = \{D^{(i)} | D^{(i)} = (X^{(i)}, Y^{(i)}, s_{gt}^{(i)}) = \{(x_k^{(i)}, y_k^{(i)}, s_{gt,k}) \}_{k=1, \ldots, n_i}^{i=1, \ldots, N}\}$. For brevity, we will use $E_D[f(X, Y, S_{gt})]$ to denote $(1/N) \sum_{D_i \in D}[f(X^{(i)}, Y^{(i)}, S_{gt}^{(i)})]$. This notation has obvious connection to the theory presented in Section 4.2.4. We are approximating the optimization of continuous functions over expected values of the loss functions in Equation 4.7 with deep neural networks and sums. The expressiveness of deep networks makes them good approximations for continuous functions, and the law of large numbers gives us good approximations for the expected values.
The problem then becomes a standard classification problem with $s_k$ implemented using DNNs with a sigmoid as its final layer:

$$
\min_\theta \mathbb{E}_D \left[ \sum_{k=1}^n \mathcal{H}(s_k(X, Y, \theta), s_{gt,k}) \right]
$$

with $\mathcal{H}$ denoted the binary cross entropy classification loss.

While a valid solution to the problem, this formulation does not account for the difference in errors between false positives and false negatives for the final product in a natural way. As the goal of the classification is to minimize the expected error $h(x_k, y_k, W)$ of the inliers, we can include the trimmed least squares formulation into the loss to account for this. Additionally, for the case where not all the data is labeled, we can introduce an additional variable for each regression example $L(i) \in \{0, 1\}$ denoting if $s^{(i)}$ is valid. Given the additional regression losses, even unlabeled examples can still get a learning signal. The new formulation becomes (with hyper-parameters $\alpha$, $\beta$, $\epsilon$):

$$
\min_\theta \mathbb{E}_D \left[ \alpha L \sum_{k=1}^n \mathcal{H}(s_k, s_{gt,k}) - \beta \sum_{k=1}^m s_k \right] \\
\text{s.t. } \mathbb{E}_D \left[ \sum_{k=1}^{n_i} s_k \|h(x_k, y_k, W)\|^2 \right] \leq \epsilon
$$

With $s_k = s_k(X, Y, \theta)$ and $W = W(X, Y, \theta)$ for brevity.

Combining this into a single loss using the Lagrangian, with additional hyper-parameter $\lambda$, the problem becomes:

$$
\min_\theta \mathbb{E}_D \left[ \alpha L \sum_{k=1}^{n_i} \mathcal{H}(s_k, s_{gt,k}) - \beta \sum_{k=1}^m s_k \right] + \lambda \mathbb{E}_D \left[ \sum_{k=1}^{n_i} s_k \|h(x_k, y_k, W)\|^2 - \epsilon \right]
$$
4.3.3 Theoretical Duality Gap

In this work we modify Equation 4.11 to fit the framework of what is presented in Eisen et al [30] discussed in Section 4.2.4. Namely given a dataset \( D \), we can formulate a general optimization problem, solvable by a Neural Network with parameters \( \theta \):

\[
\min_{\theta} \mathbb{E}_{x \in D} [f(x, \theta)] \quad \text{subject to} \quad \mathbb{E}_{x \in D} [g(x, \theta)] \leq 0
\]

Using the Lagrangian of this problem (enforcing \( \lambda \geq 0 \)) we have:

\[
\max_{\lambda \geq 0} \min_{\theta} \mathcal{L}(\theta, \lambda) = \mathbb{E}_{x \in D} [f(x, \theta)] + \lambda \sum_{k} \mathbb{E}_{x \in D} [g_k(x, \theta)] \tag{4.12}
\]

From here we can optimize by alternating minimization and maximization for the \( x \) and \( \lambda \) respectively. Formulating it this way, inspired by Eisen et al [30] and Chamon et al [21], we can prove the following results.

**Theorem 1.** Let \( \Phi \) be the space of piecewise-continuous functions and let us consider the perturbation function

\[
P^\ast(\xi) = \min_{\phi \in \Phi} \mathbb{E}_{(x, y) \in D} [f_0(\phi(x), y)]
\]

subject to \( \mathbb{E}_{(x, y) \in D} [f_k(\phi(x), y)] \leq \xi, \quad \forall k = 1, \ldots, K. \tag{4.13}
\]

If the distribution of the data is non-atomic and \( f_k \) are bounded for all \( k = 0, \ldots, K \), then \( P^\ast = D^\ast \).

Here, non-atomic means the joint distribution of the data does not have any single point with non-zero mass. With this formulation, the network trained by this solution has theoretical guarantees for satisfying the given constraints in expectation (though not for every instance).
In our problem, \( x \) would be the problem instance \( X \) and \( Y \), \( y \) would correspond to our labels \( s_{gt} \), and we optimize over functions directly rather than DNN model parameters \( \theta \).

Before we prove this theorem, first let us discuss what it means for this framework. Equation 4.12 is similar to Equation 4.11 with the key difference of maximizing \( \lambda \). Thus, Equation 4.11 becomes:

\[
\max_{\lambda \geq 0, \theta} \min \mathbb{E}_D \left[ \alpha \mathbb{L} \sum_{k=1}^{n_i} s_k - \beta \sum_{k=1}^{m} s_k \right] + \lambda \mathbb{E}_D \left[ \sum_{k=1}^{n_i} s_k \| h(x_k, y_k, W) \|_2^2 - \epsilon \right]
\]

(4.14)

Thus we can satisfy in expectation the outlier constraints in expectation by solving this min-max problem, provided we have a network with sufficient capacity and a feasible problem.

Now we prove Theorem 1. First we will need some preliminary theorems.

**Preliminary theorems**

The proof of Theorem 1 requires the following results.

**Theorem 2.** If the perturbation function associated to (4.13) is convex then \( P^* = D^* \).

**Proof.** See e.g. [9, Theorem 2.142] \( \square \)

**Theorem 3.** Consider non-atomic measures \( w_1, \ldots, w_n \) on the Borel field \( \mathcal{B} \) of subsets of a space \( \mathbb{R}^n \) and define the vector measure \( w(E) := [w_1(E), \ldots, w_n(E)]^T \). The range \( \mathcal{W} := \{w(E) : E \in \mathcal{B}\} \) of the measure \( w \) is convex.

**Proof.** See e.g. [82, 123]. \( \square \)

**Lemma 1.** Let \( \mathcal{P} := \{p \in \mathbb{R}^{K+1} \mid \exists \phi \in \Phi, \mathbb{E}_{x,y \sim \mathbb{D}} [f(\phi(x), y)] = p\} \) is convex.
Proof. Let \( p_1, p_2 \in \mathcal{P} \) and \( \phi_1, \phi_2 \in \Phi \) such that \( E_{x,y \in \mathcal{D}} [f(\phi_i(x), x)] = p_i \). Such \( \phi_1, \phi_2 \) exists since \( p_1, p_2 \in \mathcal{P} \). We next set focus to prove that for any \( \alpha \in (0, 1) \) it follows that \( \alpha p_1 + (1 - \alpha) p_2 \in \mathcal{P} \). Let \( E \subset \mathcal{D} \) and define the following vector

\[
\begin{align*}
w(E) &= \left[ \int_E f(\phi_1(x), x) dm_\mathcal{D}, \int_E f(\phi_2(x), x) dm_\mathcal{D} \right]^T.
\end{align*}
\]  

(4.15)

Since \( f \) is bounded and \( m_\mathcal{D} \) is non-atomic, it follows that \( w(E) \) is a non-atomic measure. Hence, from Theorem 3 it follows that for any \( \alpha \in (0, 1) \) we have that the vector

\[
w_0 = \alpha w(\mathcal{D}) + (1 - \alpha) w(\emptyset) \in W.
\]  

(4.16)

Therefore, there exists a set \( E_0 \subset \mathcal{D} \) such that \( w(E_0) = w_0 \). Moreover, notice that \( w(\emptyset) = 0 \) and that \( w(\mathcal{D}) = [p_1, p_2]^T \) by definition of the set \( \mathcal{P} \) and the measure \( w \). Thus, there exists \( E_0 \) such that \( W(E_0) = \alpha [p_1, p_2]^T \). Hence, it holds that

\[
\int_{E_0} f(\phi_1(x), x) dm_\mathcal{D} = \alpha p_1.
\]  

(4.17)

Let \( E_0^c \) be the complement of \( E_0 \). Following an analogous reasoning it holds that

\[
\int_{E_0^c} f(\phi_2(x), x) dm_\mathcal{D} = (1 - \alpha) p_2.
\]  

(4.18)

Define \( \phi_\alpha \) as the follows

\[
\phi_\alpha(x) = \begin{cases} 
\phi_1(x) & \text{if } (x, y) \in E_0 \\
\phi_2(x) & \text{if } (x, y) \in E_0^c.
\end{cases}
\]  

(4.19)
and notice that by construction it satisfies that $E_{(x,y) \sim m_D} [f(\phi_\alpha(x), y)] = \alpha p_1 + (1 - \alpha) p_2$. Therefore by definition of the set $P$ it follows that for any $\alpha$, $\alpha p_1 + (1 - \alpha) p_2 \in P$. This completes the proof of the result.

\[ \Box \]

**Proof of Theorem 1**

Let $\xi_1, \xi_2 \in \mathbb{R}^K$ and let $\phi_1, \phi_2 \in \Phi$ be the arguments that solve (4.13) for $\xi_1$ and $\xi_2$ respectively. Notice that $\phi_1, \phi_2$ generate elements $p_1, p_2 \in P$, the set defined in Lemma 1. Since $P$ is convex it follows that for any $\alpha \in (0,1)$, there exists $\phi_\alpha \in \Phi$ such that $E_{x,y \in D} [f(\phi_\alpha(x), y)] = \alpha p_1 + (1 - \alpha) p_2$.

We next focus in showing that $\phi_\alpha$ is feasible for the problem (4.13) with $\xi = \alpha \xi_1 + (1 - \alpha) \xi_2$. Notice that by definition we have that

$$E_{x,y \in D} [f(\phi_\alpha(x), y)] = \alpha p_1 + (1 - \alpha) p_2 \leq \alpha \xi_1 + (1 - \alpha) \xi_2,$$  \quad (4.20)

where the last inequality follows from the fact that $\phi_1, \phi_2$ are feasible for the perturbations $\xi_1, \xi_2$ respectively. Since $\phi_\alpha$ is feasible for the perturbation $\alpha \xi_1 + (1 - \alpha) \xi_2$, by definition of minimum it follows that

$$P^*(\alpha \xi_1 + (1 - \alpha) \xi_2) \geq E_{x,y \in D} [f_0(\phi_\alpha(x), y)] = \alpha P^*(\xi_1) + (1 - \alpha) P^*(\xi_2),$$  \quad (4.21)

where the equality follows form the definition of the set $P$ and the second equality from the definition of the perturbation function. The inequality in (4.21) proves that the perturbation function is indeed convex. The proof is completed by invoking Theorem 2.
Discussion

Ideally we would have theoretical results show that when we use a large but finite amount of samples to train this dual problem, we have small error from the ideal. Chamon et al [21] defined the empirical duality gap $|D^{\ast}_{\text{emp}} - P^{\ast}|$ as the gap from the ideal when moving to samples. However, their results require a pure classification loss, and would need to be extended to apply to this problem which combines classification and regression.

However, we can still use Equation 4.14 to achieve better results and reduce the duality gap of our solution closer to the theoretical optimal. On a practical level, this means optimizing the $\lambda$ in an alternating fashion with the network parameters $\theta$. If the constraints are easily satisfied, then the $\lambda$ will decrease to zero, while if they are unsatisfied then the $\lambda$ will increase to ensure the network loss focuses on them more. The specifics of how to do this are discussed in the next section.

4.4 EXPERIMENTAL RESULTS

4.4.1 Implementation details

Architectures

We represent the $f$ from the previous section using a DenseNet style architecture [61]. In each layer of DenseNet, the network takes a linear transformation of the concatenation of the outputs of all previous layers followed by a non-linearity. We use a DenseNet style sub-module to encode features for the $X$ and $Y$ inputs to learn the selection weights. In a least squares setting, we do not have any pairwise relations between the points, and thus we use Pointnet [103, 104] style feature aggregation. Pointnet uses a commutative function
Figure 4.1: Diagram for Dense-Point-Net (DPN) module for architectures solving least squares problem with outliers. The input to the module is \( N \) points (e.g. point clouds, 2D image features), each represented by a \( D \)-dimensional feature embedding. At each layer, the point features are aggregated to create a global feature, which is input to the subsequent layers. Layer \( \ell \) computes is features using a linear transform to output dimension \( K_\ell \) of all previous layers followed by a standard ReLU.

on all the point features of intermediate layers to obtain global features for subsequent layers. For this work, we use the mean as the aggregation function. Combining these two we obtain the sub-module which is the basis for all the architectures we use (see Figure 4.1 for more details). We also test on a PointNet++ architecture [104]. More details architectures used in the experiments in the following sections are described in Figure 4.2 and Table 4.1.

**Hyper-parameter selection**

Hyper-parameter tuning is important to any learning algorithm. While our method avoids the need for tuning for regularization weights on auxiliary losses, the learning rate for the dual optimizer needs to be picked. In our experiments we found that a dual optimizer learning rate 2-4 times smaller than the primal learning rate worked well for most of our experiments. Another parameter that needs to be tuned is the \( \epsilon \) from Equation 4.14. We found that an adaptable \( \epsilon \) works best, with an initialization at 1-5 times the mean of the initial untrained network errors. Then we decrease it over time using exponential decay,
4.4 EXPERIMENTAL RESULTS

Figure 4.2: Architecture diagrams for the different architectures tested below. The Standard architecture has independent networks operating on the $X$ and $Y$ variables. The Shared $XY$ architecture has networks operating on the $X$ and $Y$ variables share parameters. The Combined $XY$ architecture one network operating on the concatenated $X$ and $Y$ variables. Ultimately, the Standard architecture performed best.
<table>
<thead>
<tr>
<th>Name</th>
<th>Arch. Type</th>
<th>Layer Sizes</th>
<th>Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Standard</td>
<td>[64]</td>
<td>140,417</td>
</tr>
<tr>
<td>Medium</td>
<td>Standard</td>
<td>[256, 128]</td>
<td>932,609</td>
</tr>
<tr>
<td>Big</td>
<td>Standard</td>
<td>[256, 128, 128, 64]</td>
<td>1,656,961</td>
</tr>
<tr>
<td>Shared XY</td>
<td>Shared XY</td>
<td>[256, 128]</td>
<td>863,617</td>
</tr>
<tr>
<td>Combined XY</td>
<td>Combined XY</td>
<td>[256, 128]</td>
<td>472,705</td>
</tr>
<tr>
<td>Pointnet</td>
<td>Pointnet++</td>
<td>N/A</td>
<td>1,116,033</td>
</tr>
</tbody>
</table>

Table 4.1: Different properties of architectures used for experiments. Please refer to Figure 4.2 for details on the architecture layouts. Note that the runtime is inversely proportional to the number of parameters, except for Pointnet which is considerably slower due to the aggregation complexity.

decreasing it when the dual constraints have become negative for a certain number of iterations (20 for our experiments). We do not allow the value of $\epsilon$ to become zero, instead capping it at a small positive value approximately 100 times smaller than the initial value; in practice this value is never reached.

For the initial value of the dual value $\lambda$, we do a hyper-parameter sweep for each experiment (see later sections). We will see in the later experiments that with the primal-dual optimizer the initial value does not affect the results a great deal.

4.4.2 Robust Least Squares Experiments

For least squares, we have the data $x, y \in \mathbb{R}^d$ and $h(x, y, W) = W \cdot x - y$. We use synthetic data for the initial experiments. Formally, each regression problem is defined by $(W, \{x_k, y_k, s_k \cdot L\}_{i=1}^n)$. Please refer to Tables 4.2 and 4.3 for additional details on the data generation process.
In the following sections, we hope to (1) test the viability of the primal-dual formulation of outlier rejection and (2) find the comparative advantage of the framework compared to standard methods.

**Easily Distinguished Outliers**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Domain</th>
<th>Value Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>( \sim \mathcal{U}(S^2) )</td>
<td>( \mathbb{R}^d )</td>
<td>( c = 0.5 )</td>
</tr>
<tr>
<td>( \phi )</td>
<td>( \sim (\mathcal{U}[0, 1] \cdot \mathcal{U}[c, 1 + c])/(1 + c) )</td>
<td>( [0, 1] )</td>
<td>( 0 \leq c \leq 1 )</td>
</tr>
<tr>
<td>( W )</td>
<td>( \expm(r \cdot \phi \cdot \pi) )</td>
<td>( \mathbb{R}^{d \times d} )</td>
<td>( )</td>
</tr>
<tr>
<td>( L )</td>
<td>Label indicator</td>
<td>( {0, 1} )</td>
<td>Total determined by experiment</td>
</tr>
<tr>
<td>( s_k )</td>
<td>( \sim \text{Bernoulli}(p_{\text{inlier}}) )</td>
<td>( {0, 1} )</td>
<td>( p_{\text{inlier}} ) determined by experiment</td>
</tr>
<tr>
<td>( x_k )</td>
<td>( \sim \mathcal{N}({\mu}_x, {\Sigma}_x) )</td>
<td>( \mathbb{R}^d )</td>
<td>( \mu_x = 0, {\Sigma}_x = I )</td>
</tr>
<tr>
<td>( n_k )</td>
<td>( \sim \mathcal{N}(0, {\Sigma}_{\text{inlier}}) )</td>
<td>( \mathbb{R}^d )</td>
<td>( {\Sigma}_{\text{in}} = 0.4I )</td>
</tr>
<tr>
<td>( z_k )</td>
<td>( \sim \mathcal{N}(0, {\Sigma}_{\text{outlier}}) )</td>
<td>( \mathbb{R}^d )</td>
<td>( {\Sigma}_{\text{out}} = 10I )</td>
</tr>
<tr>
<td>( y_k )</td>
<td>( s_k(W x_k + n_k) + (1 - s_k) z_k )</td>
<td>( \mathbb{R}^d )</td>
<td>( )</td>
</tr>
</tbody>
</table>

**Table 4.2:** Definitions for the linear least squares with outliers data generation process, along with the specific values used. We used dimension \( d = 3 \)

We begin with simple least squares experiments. In this case we design an experiment where the outliers are easily distinguished from the inliers using the marginal distribution alone. Specifically, notice notice that the value used for \( {\Sigma}_{\text{in}} \) in Table 4.2 is much smaller than \( {\Sigma}_{\text{out}} \), and thus the outlier distribution is quite distinct from the inlier distribution.

In Figure 4.3, we use Equation 4.11 (\( \alpha = 1, \beta = 0, \lambda = 0.5 \)) to optimize a deep network with various architectures each for 8192 iterations, with the same hyper-parameters (ADAM Optimizer learning rate \( 4 \cdot 10^{-3} \)). In the experiments ran, \( p_{\text{inlier}} \) was tested at 70\%, 50\%, and 10\% (displayed as percent outliers), and \( N \) was tested at 1000, 2000, 4000, and 8000 points. We test a variety of different architectures, discussed in Section 4.4.1, with details
Figure 4.3: Results from various architectures and data generation parameters (x-axis) vs MSE for the final model (y-axis). All models trained using full supervision, and no primal-dual optimization but a fixed hyper-parameter $\lambda = 0.5$. The GNC [136] baseline beat the learning models in this case, though all models perform fairly well.

shown in Figure 4.2 and Table 4.1. As you can see all methods do close to perfect with the GNC [136] baseline doing the best.

This shows that (1) this is an easy problem and can be solved without much difficulty and (2) deep learning techniques are not useful in this case as standard methods work very well. Thus we need to find a harder problem for deep learning to be of use.

**Marginally Indistinguishable Outliers**

Given the ease of the previous experiments, the next batch of experiments was made much harder by making the marginal distribution of the outliers identical to the inliers. Due to the isotropic nature of the outliers, there are no clusters of inliers larger than the labelled inliers, but the labelled inliers are still challenging to find. The distribution is defined (similar to before) by $(W, T, \{x_k, y_k, s_k \cdot L\}_{k=1}^n, L)$ with the specific generation process and
parameters given in Table 4.3. Thus the network could only distinguish inliers and outliers by the joint distribution.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Domain</th>
<th>Value Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$\sim U(S^2)$</td>
<td>$\mathbb{R}^d$</td>
<td></td>
</tr>
<tr>
<td>$\phi$</td>
<td>$(U[0,1] \cdot U[c,1+c])/(1+c)$</td>
<td>$[0,1]$</td>
<td>$c = 0.5$</td>
</tr>
<tr>
<td>$W$</td>
<td>$\exp(r \cdot \phi \cdot \pi)$</td>
<td>$\mathbb{R}^{d \times d}$</td>
<td>$\mu_A = 0, \Sigma_A = I$</td>
</tr>
<tr>
<td>$T$</td>
<td>$\sim N(\mu_T, \Sigma_T)$</td>
<td>$\mathbb{R}^d$</td>
<td>$\mu_T = 0, \Sigma_T = 0.3I$</td>
</tr>
<tr>
<td>$L$</td>
<td>Label indicator</td>
<td>${0,1}$</td>
<td>Total determined by experiment</td>
</tr>
<tr>
<td>$s_k$</td>
<td>$\sim \text{Bernoulli}(p_{\text{inlier}})$</td>
<td>${0,1}$</td>
<td>$p_{\text{inlier}}$ determined by experiment</td>
</tr>
<tr>
<td>$x_k$</td>
<td>$\sim N(\mu_x, \Sigma_x)$</td>
<td>$\mathbb{R}^d$</td>
<td>$\mu_x = 0, \Sigma_x = I$, $k \in [1,N]$</td>
</tr>
<tr>
<td>$n_k$</td>
<td>$\sim N(0, \Sigma_{\text{inlier}})$</td>
<td>$\mathbb{R}^d$</td>
<td>$\Sigma_{\text{in}} = 0.4I$</td>
</tr>
<tr>
<td>$z_k$</td>
<td>$\sim N(0, \Sigma_{\text{outlier}})$</td>
<td>$\mathbb{R}^d$</td>
<td>$\Sigma_{\text{out}} = I$</td>
</tr>
<tr>
<td>$y_k$</td>
<td>$W(s_i x_k + (1-s_k)z_k) + T + n_k$</td>
<td>$\mathbb{R}^d$</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Similar to Table 4.2. We used dimension $d = 3$ for all experiments.

We run the same experiment set up from the previous section. Namely, similar to the previous experiments, we train using no primal-dual optimization, only with the classification and weighted constraint losses (Equation 4.11, $\alpha = 1$, $\beta = 0$, $\lambda = 0.5$), to distinguish architecture capacity from the optimization method. We test the same architectures from Figure 4.2 and Table 4.1. For the PointNet++ architecture [104], we used their provided implementation. Each architecture was trained for 8192 iterations, with the same hyper-parameters (ADAM Optimizer learning rate $4 \cdot 10^{-3}$).

The final results are shown in Figure 4.4. The rightmost bar in each grouping is the naive MSE with no outlier rejection i.e. chance. The leftmost bar gives the MSE that perfect model would give i.e. ground truth. For comparison across architectures, the standard architecture performed best. Unsurprisingly the model improved proportionally with model size (for exact model size see Table 4.1) Thus for all future experiments we use the ‘Large’ standard architecture.
Figure 4.4: Results from various architectures and data generation parameters (x-axis) vs test MSE or final model (y-axis). All models trained using full supervision, and no primal-dual optimization but a fixed hyper-parameter $\lambda = 0.5$. For lower percentages of outliers, the GNC \[136\] baseline performs near perfect, but for high percentages of outliers, the learning performs better. Note both the MSE from the GT estimate and the “Naive” estimate (no modeling of outliers) are shown for comparison.

The GNC baseline is very close to the ground truth in all cases except with the proportion of outliers is very high (90%). The number of points for each problem instance seems not to affect the outcome. This it seems the comparative advantage of the deep neural network is very high outlier rates with outliers that are difficult to distinguish marginally from the inliers.

4.4.3 Robust Least Squares Semi-supervised learning

Now that we have established where the comparative advantage of our method lies in the previous, we from here on will use the data generation process specified by Table 4.3 with parameters $p_{\text{inlier}} = 0.1$ and $N = 4000$ with 8192 data points per dataset (a data point being
one instance of the robust least squares problem. We test the performance of the model in a semi-supervised setting as measured by average MSE per hyper-parameter configuration. More specifically, given a fixed amount of data we test how well our training performs with different percentages of the data being labelled. We train with the primal-dual optimizer and the loss specified in Equation 4.14.

**Semi-supervised Learning**

![Graphs showing semi-supervised experiments](image)

(a) Trained with full supervision (with limited training examples) and no primal-dual optimization  
(b) Semi-supervised training without primal-dual optimization, and no auxiliary loss  
(c) Semi-supervised training with primal-dual optimization, and no auxiliary loss

**Figure 4.5**: Results for semi-supervised experiments, with 90% outliers (see Table 4.3). Training with the primal-dual optimization with a semi-supervised approach gets consistently lower and lower variance MSE loss compared to full supervision with fewer examples. Thus the primal-dual optimization is less sensitive to initialization and get better performance. In these experiments, we set the hyper-parameters from Equation 4.14 as $\alpha = 1$ and $\beta = 0$.

The results are shown in Figures 4.5 and 4.6. For Figure 4.5, we show a box plot, where each column represents the number of labelled data (logarithmically scaled). The box
Figure 4.6: Full distribution of test set residuals across all levels of supervision for one set of hyper-parameters. The distributions of the GNC [136] baseline and our model (DNN) differ. Our method get more points closer to the ground truth error, but when incorrect it has a higher variance of errors. GNC, on the other hand, has fairly consistent errors across all test points and clusters around chance. You can see that the more data is obtained, the better the model is at preventing large errors.

represents the average MSE of different initializations of dual variable $\lambda$. We also add a new baseline ADAPT [128], as it more a difficult comparison for this task.

For this robust least squares task, semi-supervised learning beats the GNC [136] baseline in all cases. The ADAPT [128] baseline does quite well in this task, and does better than semi-supervised training with all but high amounts of training data. On the left you see higher variance of test results based on hyper-parameter initialization for fully supervised learning without primal dual training. With primal dual training, shown on the right, we see a large reduction in variance and consistently lower means across all numbers of labeled examples. In these tests we set the hyper-parameter $\beta = 0$, as any positive value gave worse results (see the next section for more details).

Looking at the distribution of errors in Figure 4.6, the error characteristics of our method differ considerably from the GNC baseline. The GNC baseline has very consistent results, but none do much better than chance. Whereas our method (labeled DNN) can achieve close to ground truth results, but has a higher variance of errors. As one would expect, the more labels on trains with, the larger the reduction in variance. This figure only uses one
hyper-parameter configuration, but other configurations are very similar. To see the scatter plots for all configurations, see Figure 4.11.

**Unsupervised Learning**

![Figure 4.7: Hyper-parameter sweep over various architectures and learning rates. No experiment did better than chance.](image)

In Equation 4.14 we have hyper-parameters $\alpha$ and $\beta$ determining how much the loss should be focused on the supervised loss ($\alpha$) or unsupervised regularizer ($\beta$). Here we test different proportions to see empirically how well they work.

In Figure 4.7 we test using a purely unsupervised loss (i.e. $\alpha = 0$ and $\beta = 1$) using the marginally indistinguishable outliers. However, none of the results do better than chance, and all do worse than the GNC baseline. Even with a very large hyper-parameter sweep, this problem seems unable to be solved using unsupervised methods, as shown in Figure 4.7. Alternative hyper-parameters and auxiliary unsupervised losses did not perform as
well, as shown in Figure 4.8. Thus, it appears in the cases where deep learning might be useful, the problem is too difficult to learn in an unsupervised fashion.

To see if the unsupervised regularizer helps at all, we test semi-supervised framework with the configuration $\alpha = 1$ and $\beta = 0.1$. In Figure 4.8, shown in the same format as Figure 4.5 from Section 4.4.3. While when enough labeling the model can beat the GNC baseline, it doesn’t beat ADAPT and thus is not as viable.

4.4.4 Robust Two-view Matching

For least squares, we have the data $x_1, x_2 \in \mathbb{R}^2$ with $x = [x_1^T, 1] \otimes [x_2^T, 1]$ (denoting the Kronecker product) and $y = 0$ in all cases. The error model is $h(x, y, W) = x \cdot W$, thus the solution is finding the homogeneous solution to the equation. One can see the derivation in [51]. Here we run experiments on synthetic data again with the parameters specified in Table 4.4, in this case using a naive KNN feature matching to find the matches. This gives us equivalent outlier rates of 90%.

Similar to the robust least squares case, in semi-supervised training, robust two view matching gives more stable training and is less sensitive to parameter initialization. We use the same ‘standard’ architecture as the robust least squares matching case. Subfigure 4.9a is essentially the same algorithm as in Yi et al [142], who assume full supervision. Comparing it to Subfigure 4.9b, we can see that in the semi-supervised setting the primal-dual optimization is very important for stability and a good final testing MSE. In this application, GNC and ADAPT baselines do not perform well due to the more non-convex nature of the two view matching problem compared to least squares.

We found (unsurprisingly given the literature) that the network performed much better when given both the $(x, y)$ coordinates and features used for matching, as seen in Figure 4.10. This is one advantage of learning methods over traditional optimization baselines, in
Table 4.4: Definitions for two view image matching problem data generation process, along with the specific values used. In the experiments, we used $N = 500$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Domain</th>
<th>Value Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_k$</td>
<td>$\sim \mathcal{U}[-s_X, s_X]^3 + T_d$</td>
<td>$\mathbb{R}^3$</td>
<td>$s_X = 5, T_d = (0,0,12)^T, k \in [1,N]$</td>
</tr>
<tr>
<td>$r_c$</td>
<td>$\sim \mathcal{U}(S^2)$</td>
<td>$\mathbb{R}^3$</td>
<td></td>
</tr>
<tr>
<td>$\phi_c$</td>
<td>$\mathcal{U}[0,1] \cdot \mathcal{U}[c, 1+c]/(1+c)$</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td></td>
</tr>
<tr>
<td>$R_c$</td>
<td>$\exp(r \cdot \phi \cdot \tau_c)$</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td>$\tau_c = \pi/2$</td>
</tr>
<tr>
<td>$r_n$</td>
<td>$\sim \mathcal{U}(S^2)$</td>
<td>$\mathbb{R}^3$</td>
<td></td>
</tr>
<tr>
<td>$\phi_n$</td>
<td>$\sim \mathcal{U}[0,1]$</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td>$\tau_n = \pi/12$</td>
</tr>
<tr>
<td>$R_n$</td>
<td>$\exp(r \cdot \phi \cdot \tau_n)$</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td>Label indicator</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td>Total determined by experiment</td>
</tr>
<tr>
<td>$R$</td>
<td>$R_c \cdot R_n$</td>
<td>$\mathbb{R}^3 \times 3$</td>
<td></td>
</tr>
<tr>
<td>$T_n$</td>
<td>$\sim \mathcal{N}(0, \Sigma_{T_n})$</td>
<td>$\mathbb{R}^3$</td>
<td>$\Sigma_{T_n} = 0.08 \cdot I_3$</td>
</tr>
<tr>
<td>$T$</td>
<td>$(I - R_c)T_d + T_n$</td>
<td>$\mathbb{R}^3$</td>
<td>$\Sigma_{T_n} = 0.08 \cdot I_3$</td>
</tr>
<tr>
<td>$Y_k$</td>
<td>$R \cdot X_k + T$</td>
<td>$\mathbb{R}^3$</td>
<td></td>
</tr>
<tr>
<td>$x_k$</td>
<td>$(X_{k,1}/X_{k,3}, X_{k,2}/X_{k,3})$</td>
<td>$\mathbb{R}^2$</td>
<td></td>
</tr>
<tr>
<td>$y_k$</td>
<td>$(Y_{k,1}/Y_{k,3}, Y_{k,2}/Y_{k,3})$</td>
<td>$\mathbb{R}^2$</td>
<td></td>
</tr>
<tr>
<td>$F_k$</td>
<td>$\sim \mathcal{U}[-1,1]^{d_F}$</td>
<td>$\mathbb{R}^{d_F}$</td>
<td>$d_F = 16$</td>
</tr>
<tr>
<td>$F_{x,k}$</td>
<td>$F_k + \mathcal{N}(0, \Sigma_F)$</td>
<td>$\mathbb{R}^{d_F}$</td>
<td>$\Sigma_F = 0.5 \cdot I_{d_F}$</td>
</tr>
<tr>
<td>$F_{y,k}$</td>
<td>$F_k + \mathcal{N}(0, \Sigma_F)$</td>
<td>$\mathbb{R}^{d_F}$</td>
<td></td>
</tr>
<tr>
<td>$(x_i, y_j)$</td>
<td>$j \in \text{KNN}([F_{y,t}]<em>{t=1,\ldots,N}, F</em>{x,k})$</td>
<td>$\mathbb{R}^6$</td>
<td>$K = 8$</td>
</tr>
</tbody>
</table>

that they can incorporate relevant features in the matching that do not have necessarily have any geometric information.

4.5 CONCLUSION

We have introduced a framework training a network to learn robust matching in general matching settings. Using trimmed least squares as an inspiration, we formulated
a constrained learning problem for outlier rejection. We then propose to solve it using
primal-dual optimization techniques to optimize for expected constraint satisfaction. We
prove the theoretical duality gap for such expected constraint satisfaction problem is
zero. We ran experiments for various applications, showing the usefulness of the method
in semi-supervised training regime. Applying this technique to more practical settings,
finding better initialization techniques, and theoretical work showing the empirical duality
gap is small remain the subject for future work.
Figure 4.8: Results for semi-supervised experiments with additional losses ($\beta = 0.1$ from Equation 4.14). $L_{\text{max}}(S) = -\sum k s_k$, the same as in Equation 4.14, while $L_{\text{max}}(S)$ replaces it with $|\sum k s_k - p_{\text{out}}|$ with $p_{\text{out}}$ being the true outlier rate. Both of these losses did not outperform the $\beta = 0$ case, shown in Figure 4.5.
(a) Semi-supervised training and no primal-dual optimization  
(b) Semi-supervised training with primal-dual optimization

Figure 4.9: Results for two view matching semi-supervised experiments. Training with the primal-dual optimization with a semi-supervised approach gets consistently lower and lower variance MSE loss compared to training without it. This is a replication of the results from Figure 4.5 for the two view case.
(a) Semi-supervised training without matching features

(b) Semi-supervised training with matching features

**Figure 4.10:** Results for two view matching semi-supervised experiments. Training with the primal-dual optimization with a semi-supervised approach gets consistently lower and lower variance MSE loss compared to training without it. This is a replication of the results from Figure 4.5 for the two view case.
Figure 4.11: Full distribution of test set residuals across all runs, similar to figure (ref)
We have discussed matching across a wide variety of different fields and applications. We discussed the history and traditional techniques of matching in Chapter 1. Then we looked at robust matching in the continuous domain in Chapter 2, using a novel weighing scheme to reject outliers. We shifted in Chapter 3 to wide baseline matching, but used multiple images and unsupervised losses to train a Graph Neural Network to solve the multi-image matching problem. In Chapter 4 how to explicitly add in outlier rejection into DNN training, combining Lagrangian duality theory and deep learning. There is a large breadth of applications to the matching, and this dissertation only covers a small part of it. However, by learning better representations to facilitate matching, we can improve performance and generalization across many of those applications.

There are many steps forward we could take with the works in the previous chapters. For Chapter 2, we could use more intelligent sampling methods for the translation to improve performance. For the work in Chapter 3, we hope to investigate more robust losses for better outlier rejection, and using higher order geometric constraints, such as the trifocal tensor, as additional loss terms. We can also extend this to distributed settings where we can train for matching images from multiple distributed agents. Finally, for Chapter 4, we hope to gain better theoretical insights for the duality gap in empirical settings. We also hope to apply the method in more realistic datasets, and perhaps a wider variety of settings. We hope to add these extentions in future work to be able to handle robust matching in more difficult settings.
LIST OF TABLES

Table 1.1  Table of the different types of matching problems 3
Table 3.1  Results for unsupervised training on synthetic data under various noise conditions. The table plots out the weights (mean and standard deviation) of the edges reconstructed by our model, for true positive matches and true negative ones. This shows under various noise conditions that our architecture can still recover the original connectivity structure of the matching graph. 75
Table 3.2  Results on Romen6K Correspondence graphs, showing the mean and standard deviation of the $L_1$ and $L_2$. Our method was not trained on ground truth correspondences but using unsupervised methods and geometric side losses. Thus we test against ground truth correspondence graph adjacency matrices computed from the bundle adjustment output. Our method performs better than 35 iteration of the MatchALS [156] method, but does not perform as well as 50 iterations. We perform better than a simple eigenvalue based method [98]. Note that we perform much better in $L_1$ performance rather than $L_2$, as we optimized the network weights using an $L_1$ loss. 81
Table 4.1  Different properties of architectures used for experiments. Please refer to Figure 4.2 for details on the architecture layouts. Note that the runtime is inversely proportional to the number of parameters, except for Pointnet which is considerably slower due to the aggregation complexity. 100
Table 4.2  Definitions for the linear least squares with outliers data generation process, along with the specific values used. We used dimension $d = 3$ 101
Table 4.3  Similar to Table 4.2. We used dimension $d = 3$ for all experiments. 103
Table 4.4  Definitions for two view image matching problem data generation process, along with the specific values used. In the experiments, we used $N = 500$. 109

116
| Figure 1.1 | Illustration of the many variations and applications of the matching problem. Matching can be applied to low level geometric problems as in two-view and multi-view feature matching or optical flow. Alternatively, matching can be applied to semantic matching for high level analysis of images. All this can come together for applications such as semantic SLAM. Figures taken from [64, 142, 79, 131, 10]. |
| Figure 1.2 | (a) The first deep learning submission to the ImageNet [109] competition, AlexNet [76], dramatically improved results compared to the non-Deep Learning counterparts, nearly by a factor of 2. All winners of ImageNet since then have used deep Learning. (b) The percentage of papers with deep learning or related techniques (e.g, LSTMs) explicitly in the title has dramatically increased in recent years. This is not including papers using deep learning without mentioning it in the title. (c) The number of papers submitted to the major Computer Vision conferences has been rising steadily since the increase in popularity in deep learning. |
Figure 2.1  Schematic depiction of the ERL method for egomotion estimation from noisy flow fields. Figure best viewed in color. (A) Example optical flow field from two frames of KITTI odometry (sequence 5, images 2358-2359). Note the outliers on the grass in the lower right part of the image and scattered throughout the flow field. (B) We evaluate the flow field under M models with translation parameters sampled uniformly over the unit hemisphere. The residuals for the flow field under three counterfactual models are shown. Each black point indicates the translation direction used. Residuals are scaled to [0,1] for visualization. (C) We estimate the likelihood of each observed residual under each of the models by fitting a Laplacian distribution to each set of residuals. The final confidence weight for each flow vector is estimated as the expected value of the residual likelihood over the set of counterfactual models. Likelihood distributions are shown for the three models above. (D) The weighted flow field is used to make a final estimate of the true egomotion parameters. The black point indicates the translation direction estimated using ERL and the green point indicates ground truth. The unweighted estimate of translation is not visible as it is outside of the image bounds.

Figure 2.2  A 2D line-fitting problem demonstrating how ERL weights inliers and outliers. Inliers are generated as \( y_i \approx 2x_i + 1 \) with Gaussian noise. Each data points is colored according to its estimated confidence weight.

Figure 2.3  Robust methods recover the error surface of the outlier-free flow field. (A) Example optical flow field from two frames of KITTI odometry (sequence 10, images 14-15). Note the prominent outliers indicated by the yellow box. Error surfaces on this flow field for (A) the raw method (Equation 2.5) with all flow vectors, (B) with outliers removed by hand, and (C) with confidence weights estimated by ERL or (D) the lifted kernel. The green point is the true translational velocity and the black point the method’s estimate. Blue: low error. Red: high error. Translation components are given in calibrated coordinates.

Figure 2.4  Median translational and rotational errors on the full KITTI odometry dataset for our methods and baselines.

Figure 2.5  Log likelihoods of Laplacian and Gaussian fits to the optical flow error, Gaussian likelihood on the x-axis and Laplacian likelihood on the y-axis. Laplacian fits are consistently better than Gaussian fits.
Figure 2.6  Median translational and rotational errors on the full KITTI odometry dataset for ERL with two candidate distributions. 46

Figure 2.7  Distributions of errors. (a) Full distribution of translational velocity errors. (b) Full distribution of rotational velocity errors. 47

Figure 2.8  Translation error as a function of percent outliers on synthetic data for our robust methods and two baseline continuous egomotion methods. 48

Figure 3.1  An illustration of the approach of this work. The Graph Neural Neural Network (GNN) [5] takes as input the graph of matches and then outputs a low rank embedding of the adjacency matrix of the graph. The GNN operates on an embedding over the vertices of the graph. In the figure, the GNN vertex embeddings are represented by different colors. The final embedding is used to construct a pairwise similarity matrix, which we train to be a low dimensional cycle-consistent representation of the graph adjacency matrix, thus pruning the erroneous matches. We train the network using a reconstruction loss on the similarity matrix with the noisy adjacency matrix, and thus do not need ground truth matches. In addition, we can use geometric consistency information, such as epipolar constraints, to assist training the network. 52

Figure 3.2  Examples of graphs in learning problems. 57

Figure 3.3  Diagram of the filter weight layout of a standard convolutional neural network and graph convolutional network. Different colors correspond to different weights, with light blue being zero. In (a), due to the regular structure of the grid, we can have a very expressive weight layout. In contrast, note in (b) how due to the lack of a regular structure, we can only give weights by distance in the graph (here 1-hop distance). 60

Figure 3.4  Diagram of graph convolutions on a grid structure, visually showing that they are equivalent to standard grid convolutions (but with radially symmetric filters) 61

Figure 3.5  Diagram of information (shown as the purple outline on the nodes) being passed in a message passing like algorithm 65

Figure 3.6  Diagram of Graph Nets [5]. We have node attributes $v_i$, edge attributes $e_{ij}$, and global attributes $u$. In the first step, we aggregate information from the global attribute $u$ and nodes $v_i$, $v_j$ to their edges $e_{ij}$. In the next step, we aggregate information from the global attribute $u$ and the edges $e_{ij}$ to nodes they point to $v_j$. In final step, we aggregate information from all edges $e_{ij}$ and all nodes $v_j$ to the global attribute $u$. We can repeat this with different weights at each step to build our graph network. 67
Figure 3.7 (a) An illustration of the idea of the universe of features. Each feature in each image corresponds to a 3D point in the scene. We can construct cycle consistent embeddings of the features by mapping each one to the one-hot vector of its corresponding 3D point. While there can be many features, there are fewer 3D points and thus this corresponds to a low rank factorization of the correspondence matrix. Best viewed in color. (b) Visualization of the learned embeddings. On the left we have the raw outputs, which are difficult to interpret. In the center, we rotated the features to best match the ground truth for a more interpretable visualization (see the end of Section 3.4.3). On the right, we have the ground truth embeddings, given as indicator vectors for which feature in the world the points correspond to. For the optimally rotated embedding we can see that the true embedding structure is recovered (with some noise).

Figure 3.8 (a) Errors are computed via absolute distance from the epipolar line, as expressed by Equation 3.21 via the epipolar constraint. The epipolar line is the line of projection of the feature in the first image, projected into the second. The distance to this line on the second image indicates how likely that point is to correspond geometrically to the original feature. There can be false positives along the projected line, as shown by the square feature in the figure, but other points will be eliminated, such as the hexagonal feature. (b) Training curves with and without Geometric Training loss, described in Equation 3.23. The geometric training loss improves testing performance. Note how training with geometric consistency losses decreases the convergence time of the network. Best viewed in color.

Figure 3.9 Plot of the losses of the baselines at different iteration numbers. The line shows the mean of the graph while the translucent coloring shows the 25th to 75th percentiles. The ROC AUC curves remain fairly consistent while the L1 loss goes noticeably down after more iterations. Our method compares to 35-45 iterations of MatchALS, while only having 8 message passes. PGDDS performs better than us in L1 but we perform similarly in the ROC AUC metric. These results still hold even when we change domains to the Graffiti dataset (see 3.5.3).
Figure 4.1 Diagram for Dense-Point-Net (DPN) module for architectures solving least squares problem with outliers. The input to the module is N points (e.g. point clouds, 2D image features), each represented by a D-dimensional feature embedding. At each layer, the point features are aggregated to create a global feature, which is input to the subsequent layers. Layer ℓ computes features using a linear transform to output dimension K_ℓ of all previous layers followed by a standard ReLU.

98

Figure 4.2 Architecture diagrams for the different architectures tested below. The Standard architecture has independent networks operating on the X and Y variables. The Shared XY architecture has networks operating on the X and Y variables share parameters. The Combined XY architecture one network operating on the concatenated X and Y variables. Ultimately, the Standard architecture performed best.

99

Figure 4.3 Results from various architectures and data generation parameters (x-axis) vs MSE for the final model (y-axis). All models trained using full supervision, and no primal-dual optimization but a fixed hyper-parameter λ = 0.5. The GNC [136] baseline beat the learning models in this case, though all models perform fairly well.

102

Figure 4.4 Results from various architectures and data generation parameters (x-axis) vs test MSE or final model (y-axis). All models trained using full supervision, and no primal-dual optimization but a fixed hyper-parameter λ = 0.5. For lower percentages of outliers, the GNC [136] baseline performs near perfect, but for high percentages of outliers, the learning performs better. Note both the MSE from the GT estimate and the “Naive” estimate (no modeling of outliers) are shown for comparison.

104

Figure 4.5 Results for semi-supervised experiments, with 90% outliers (see Table 4.3). Training with the primal-dual optimization with a semi-supervised approach gets consistently lower and lower variance MSE loss compared to full supervision with fewer examples. Thus the primal-dual optimization is less sensitive to initialization and get better performance. In these experiments, we set the hyper-parameters from Equation 4.14 as α = 1 and β = 0.

105
Full distribution of test set residuals across all levels of supervision for one set of hyper-parameters. The distributions of the GNC [136] baseline and our model (DNN) differ. Our method gets more points closer to the ground truth error, but when incorrect it has a higher variance of errors. GNC, on the other hand, has fairly consistent errors across all test points and clusters around chance. You can see that the more data is obtained, the better the model is at preventing large errors.

Hyper-parameter sweep over various architectures and learning rates. No experiment did better than chance.

Results for semi-supervised experiments with additional losses ($\beta = 0.1$ from Equation 4.14). $L_{\text{max}}(S) = - \sum_k s_k$, the same as in Equation 4.14, while $L_{\text{max}}(S)$ replaces it with $|\sum_k s_k - p_{\text{out}}|$ with $p_{\text{out}}$ being the true outlier rate. Both of these losses did not outperform the $\beta = 0$ case, shown in Figure 4.5.

Results for two view matching semi-supervised experiments. Training with the primal-dual optimization with a semi-supervised approach gets consistently lower and lower variance MSE loss compared to training without it. This is a replication of the results from Figure 4.5 for the two view case.

Full distribution of test set residuals across all runs, similar to figure (ref)
BIBLIOGRAPHY


