State Estimation, Control, And Planning For A Quadrotor Team

Kartik Mohta
University of Pennsylvania, kartikmohta@gmail.com

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

Part of the Robotics Commons

Recommended Citation
Mohta, Kartik, "State Estimation, Control, And Planning For A Quadrotor Team" (2018). Publicly Accessible Penn Dissertations. 3360.
https://repository.upenn.edu/edissertations/3360

This paper is posted at ScholarlyCommons, https://repository.upenn.edu/edissertations/3360
For more information, please contact repository@pobox.upenn.edu.
State Estimation, Control, And Planning For A Quadrotor Team

Abstract
Teams of aerial robots have the potential to become effective information gathering instruments, specially in the fields of infrastructure inspection, surveillance, and environmental monitoring due to their ability to operate in environments which may be difficult or impassable for ground robots. This dissertation addresses the problem of developing a team of autonomous quadrotors that can be quickly deployed and controlled by a single human operator.

First, we describe the system design and algorithms for robust single robot autonomy and demonstrate it with experiments in a variety of environments. The developed robot uses onboard cameras for state estimation and is capable of autonomous navigation through natural and man-made environments at high speeds.

Next, we develop the system architecture and components to allow a single operator to deploy, control, and monitor a team of robots.

In order to focus on the multi-robot coordination problem, the robots used for this part utilized GPS for state estimation and required known maps of the environment for obstacle avoidance, which limited the usage of the system to known outdoor environments.

Finally, we combine the robots from the first part with this multi-robot framework. The main challenge here is that due to using onboard sensing for state estimation, each robot has a different reference frame for its state estimates, typically with the origin at the starting pose of the robot while the high-level planner for the team requires all the robot state estimates to be in a common reference frame.

We propose a method that uses the relative position/bearing measurements of nearby robots detected using the onboard camera to solve this problem. The complication in this method is that these relative measurements do not contain any identity information since all the robots in the team are equivalent and look the same. In order to fuse these unlabeled measurements and estimate the states of the robots in a common reference frame, we used the Coupled Probabilistic Data Association Filter (CPDAF) and developed an approximation to reduce its computational complexity enabling online estimation with a team of up to 15 robots.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Electrical & Systems Engineering

First Advisor
Vijay Kumar

Keywords
control, planning, quadrotor, relative localization, state estimation
STATE ESTIMATION, CONTROL, AND PLANNING FOR A QUADROTOR TEAM

Kartik Mohta

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2018

Supervisor of Dissertation

Vijay Kumar, Nemirovsky Family Dean and Professor of Mechanical Engineering and Applied Mechanics

Graduate Group Chairperson

Victor Preciado, Associate Professor of Electrical and Systems Engineering

Dissertation Committee

Alejandro Ribeiro, Professor of Electrical and Systems Engineering

Vijay Kumar, Nemirovsky Family Dean and Professor of Mechanical Engineering and Applied Mechanics

Camillo J. Taylor, Professor of Computer and Information Science

Kostas Daniilidis, Ruth Yalom Stone Professor of Computer and Information Science

Nikolay Atanasov, Assistant Professor of Electrical and Computer Engineering
ACKNOWLEDGEMENTS

A Ph.D. is rarely the output of a single person working alone, it requires a team of people to provide the necessary support to make it successful. I have been very fortunate during my journey through this process to have had the company which made it not just possible but also enjoyable.

First and foremost, I want to thank my advisor, Prof. Vijay Kumar, who gave me the opportunity to pursue this path. His vast knowledge and experience in robotics have been particularly helpful for guidance whenever I was unsure about the direction in my research. The resources available in the lab allowed me to work on things I would not have thought possible before joining the lab, and I consider myself lucky to have gotten this opportunity. I also want to thank my committee members for their time and constructive feedback about the topics to focus on for my research.

During my time at Penn, I have had an exceptional group of people for company in the lab. The lab environment has been friendly but also academically stimulating, with somebody always available for discussing the minute details of any technical subject.

The FLA project has been a large part of my thesis work, and many people were involved in making it a success. I specifically want to thank Sikang, Mike, Yash, Ke, Bernd, Jeremy, and Shreyas who formed the core team without whom the project would not have been possible. Special thanks to CJ for being there with us on all those trips and providing just the right amount of guidance.

I cannot understate the support I have received from my parents in my life. They have always encouraged me to pursue the path that I enjoy without worrying about the sacrifices they have had to make. None of this would have been possible without them.

Finally, I want to thank my wife, Vandana, for her constant support through this long journey. It would have been hard to finish this without her persistent encouragement.
ABSTRACT

STATE ESTIMATION, CONTROL, AND PLANNING FOR A QUADROTOR TEAM

Kartik Mohta
Vijay Kumar

Teams of aerial robots have the potential to become effective information gathering instruments, specially in the fields of infrastructure inspection, surveillance, and environmental monitoring due to their ability to operate in environments which may be difficult or impassable for ground robots. This dissertation addresses the problem of developing a team of autonomous quadrotors that can be quickly deployed and controlled by a single human operator.

First, we describe the system design and algorithms for robust single robot autonomy and demonstrate it with experiments in a variety of environments. The developed robot uses onboard cameras for state estimation and is capable of autonomous navigation through natural and man-made environments at high speeds.

Next, we develop the system architecture and components to allow a single operator to deploy, control, and monitor a team of robots. In order to focus on the multi-robot coordination problem, the robots used for this part utilized GPS for state estimation and required known maps of the environment for obstacle avoidance, which limited the usage of the system to known outdoor environments.

Finally, we combine the robots from the first part with this multi-robot framework. The main challenge here is that due to using onboard sensing for state estimation, each robot has a different reference frame for its state estimates, typically with the origin at the starting pose of the robot while the high-level planner for the team requires all the robot state estimates to be in a common reference frame. We propose a method that uses the relative position/bearing measurements of nearby robots detected using the onboard camera to solve
this problem. The complication in this method is that these relative measurements do not contain any identity information since all the robots in the team are equivalent and look the same. In order to fuse these unlabeled measurements and estimate the states of the robots in a common reference frame, we used the Coupled Probabilistic Data Association Filter (CPDAF) and developed an approximation to reduce its computational complexity enabling online estimation with a team of up to 15 robots.
# Contents

Acknowledgements ii

Abstract iii

List of Tables vii

List of Figures x

1 Introduction 1

1.1 Motivation ........................................... 1
1.2 Problem Statement .................................... 3
1.3 Main Contributions ................................... 4

2 Related Work 6

2.1 State Estimation ....................................... 6
2.2 Control .................................................. 13
2.3 Mapping .................................................. 14
2.4 Trajectory Planning .................................... 15
2.5 Multi-robot UAV systems ............................ 17

3 Preliminaries 22

3.1 Notation .................................................. 22
3.2 Reference Frames ...................................... 24
3.3 Quadrotor Dynamics ................................... 25

4 Single Robot Autonomy 27

4.1 Sensing, Computation and Communication ............. 28
4.2 Software Architecture .................................. 31
4.3 State Estimation ....................................... 32
4.4 Control .................................................. 42
4.5 Mapping and Planning .................................. 45
4.6 Experimental Results ................................... 49

5 Multi-Robot Autonomy – A Centralized Paradigm with External Infrastructure 54

5.1 System Architecture .................................... 54
5.2 Robot Hardware .......................................... 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Software</td>
<td>58</td>
</tr>
<tr>
<td>5.4 Estimation and Control</td>
<td>58</td>
</tr>
<tr>
<td>5.5 Planning</td>
<td>62</td>
</tr>
<tr>
<td>5.6 Communication and Supervision</td>
<td>63</td>
</tr>
<tr>
<td>5.7 Aggregation of visual imagery</td>
<td>64</td>
</tr>
<tr>
<td>5.8 3D Reconstruction</td>
<td>64</td>
</tr>
<tr>
<td>5.9 Experimental Results</td>
<td>66</td>
</tr>
<tr>
<td>6 Multi-Robot Autonomy – Towards a Decentralized</td>
<td>71</td>
</tr>
<tr>
<td>External Infrastructure</td>
<td></td>
</tr>
<tr>
<td>6.1 Need for a common reference frame</td>
<td>71</td>
</tr>
<tr>
<td>6.2 Using absolute/indirect measurements</td>
<td>72</td>
</tr>
<tr>
<td>6.3 Using relative measurements</td>
<td>73</td>
</tr>
<tr>
<td>6.4 Using unlabeled relative measurements</td>
<td>73</td>
</tr>
<tr>
<td>6.5 Problem formulation</td>
<td>74</td>
</tr>
<tr>
<td>6.6 Approach</td>
<td>77</td>
</tr>
<tr>
<td>6.7 Experiments</td>
<td>86</td>
</tr>
<tr>
<td>6.8 Extension: Bearing only relative measurements</td>
<td>92</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>95</td>
</tr>
<tr>
<td>A Multi-target tracking</td>
<td>97</td>
</tr>
<tr>
<td>A.1 Joint Probabilistic Data Association</td>
<td>97</td>
</tr>
<tr>
<td>A.2 Multi Hypothesis Tracking</td>
<td>103</td>
</tr>
<tr>
<td>A.3 Random Finite Sets</td>
<td>109</td>
</tr>
<tr>
<td>Bibliography</td>
<td>110</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Comparison of sensors used for state estimation. . . . . . . . . . . . . . . . . . 7
2.2 Advantages and disadvantages of different visual odometry configurations. . . 11

4.1 Approximate CPU usage of the components of the system. . . . . . . . . . . 50

5.1 Mean error between the desired position and estimated position in the hori-
zontal plane for each robot during the six robot experiment . . . . . . . . . 70

A.1 The possible HOMHT global hypotheses generated from the example shown
in Figure A.1. $\emptyset$ represents that the target is not associated with any mea-
surement. $T_3$, $T_4$ and $T_5$ are potential new targets (see Figure A.2). . . . . 105
List of Figures

3.1 Rotation of frame $A$ to frame $B$ using the Z-Y-X intrinsic Tait-Bryan angles convention. .......................................................... 24
3.2 The forces and moments generated by the individual motors of a quadrotor. The vectors $b_1$, $b_2$, $b_3$ and $w_1$, $w_2$, $w_3$ are the basis vectors for the body frame and world frame respectively. .................................................................. 26

4.1 Our mapping solution consisting of a 2D lidar mounted on a nodding gimbal. 30
4.2 A high level block diagram of our system architecture. ......................... 31
4.3 A comparison of the accuracy and computational efficiency our VIO system with various open source packages on the openly available EuRoC dataset. Our method fails on the V2_03 dataset due to significantly different exposure times on the two cameras in some parts of the dataset. ......................... 35
4.4 Sequence of images showing how our auto-exposure algorithm synchronously changes the exposure of both the left and right cameras as the robot goes from a dimly lit indoor to a bright outdoor environment. The actual exposure times change from 10 ms in the first pair of images to 0.01 ms in the last. . . . 37
4.5 Data flow diagram of the UKF used on the robot. ................................. 38
4.6 The forces on the robot when it is moving towards the right with a velocity $v$. Note that the modelled drag is in the x-y plane in the robot body frame, which may not be exactly opposite to the direction of velocity. ............. 45
4.7 The set of positions reached from the origin after two steps of expansion using nine constant acceleration primitives at each step. Yellow lines represent the first expansion while the green curves represent the second. We can see the discretization in the set of positions reached by this set of primitives as it expands. Note that the primitives do not necessarily end at zero velocity, so the state of the robot at most of these positions has non-zero velocity. ........ 47
4.8 Effect of initial state on the planned trajectory. The green dot is the start position and the red dot is the goal. (a) shows the output of a path based planner which does not take into account the dynamics of the robot, (b) shows the output of the motion primitives based planner where the initial state is at rest, and (c) shows the trajectory from the motion primitives based planner when the robot has an initial velocity in the upward direction. 48

4.9 The effect of the refinement on the trajectory generated from the graph search. The original trajectory from the search based planner with constant acceleration motion primitives is shown in orange while the refined trajectory is shown in blue. It can be seen that refined trajectory is smoother but the difference in positions between the original and the refined trajectories is very small. 48

4.10 Plots of the velocity and tracking errors during a high speed flight without drag compensation where the robot was commanded to follow a 300 m straight line trajectory with a maximum speed of 17.5 m/s. 51

4.11 Improvement in the position tracking when using the controller with drag compensation. The robot is commanded to follow a 300 m straight line trajectory at a speed of 15 m/s. 52

4.12 A series of images from the experiment showing the progress of the robot starting from the forest environment and ending inside the building. 53

4.13 An overhead map showing the full run for the indoor + outdoor test. The starting position of the robot is marked with a green circle while the goal position is marked in red. The path followed by the robot is shown in blue. 53

5.1 Decentralized robot motion planning and control along with a centralized model for human interaction and task assignment. 55

5.2 The KMel kQuad500 quadrotor equipped with a u-blox NEO-6T GPS module, a Matrix Vision mvBlueFOX camera, a Ubiquity Networks Bullet M2 and a Odroid-U2 quad-core ARM single board computer. 57

5.3 The KMel NanoHex equipped with an Odroid-U2 quad-core ARM single board computer, a downward facing mvBlueFOX camera, a GPS module and a Ubiquity Networks Bullet M2. The Bullet-M2 is mounted on the bottom of the robot and is not visible in the picture. 58

5.4 The estimation and control systems running on each robot. 59

5.5 A mosaic being created using the images from three robots. The robot positions can be seen by the red circles in the image. 65

5.6 The 3D reconstruction pipeline running on the base station for each robot. 66

5.7 The 3D reconstruction of a 20 m x 20 m outdoor scene using the images from one robot. The colors represent the height, with increasing height going from red to green. 66

5.8 UKF estimates during a representative hovering experiment in an open area. Ground truth from an OptiTrack motion capture system, which was set up specifically for this experiment, are shown for reference. The position tracking errors had a standard deviation of 0.158 m in the horizontal direction and 0.386 m in the vertical direction. 67

5.9 An outdoor experiment with six robots. 68
5.10 A series of snapshots of the user interface while running an experiment with six robots. 69

6.1 Number of joint association events as a function of the number of targets and number of measurements. Note that the Z-axis has log scale. 84

6.2 Number of joint association events with gated measurements. Note that the Z-axis has log scale. 85

6.3 Comparison of the computation time required for the CPDAF update step with and without gating. Note that the Y-axis has log scale. 86

6.4 An examples of the tree of association events for three targets and three measurements after gating. In this example, the gated measurements for the three targets are \{1, 2\}, \{1, 2, 3\}, and \{3\} respectively. Note that one measurement can only be associated with a single target for a valid joint association, hence some of the branches in the tree are invalid. The set of valid joint associations in this case is \{(1,2,3), (2,1,3)\}. 88

6.5 Detection of other robots in the image from the onboard camera of one of the robots during a flight test. 89

6.6 Computing the distance of the robot from the image size. \(w_r\) is the real robot width, \(w_i\) is the robot width in the image, \(f\) is the camera focal length, and \(Z\) is the distance along the optical axis of the robot from the camera. 90

6.7 Comparison of the position and yaw of the three robots in the common reference frame estimated using our approach during a flight test with ground truth from a motion capture system. 91

6.8 Estimates of the position and yaw of the three robots in the common reference frame using our approach with unlabeled relative bearing measurements during a flight test. 94

A.1 An example showing the predicted measurements, \(\hat{z}_1\) and \(\hat{z}_2\), from two targets and three observations, \(z_1\), \(z_2\) and \(z_3\), with ellipsoids representing the measurement validation gates. 104

A.2 TOMHT target trees generated by the example shown in Figure A.1. \(\emptyset\) represents that none of the measurements correspond to that track. \(T_3\), \(T_4\) and \(T_5\) are potential new targets. 105

A.3 TOMHT target trees generated as sets of measurements at different time steps are processed. Assume that at time step \(k - 1\) we have two resolved target tracks, \(T_1\) and \(T_2\). At time step \(k\), we receive the measurement set \(\{z_1, z_2\}\) where both \(z_1\) and \(z_2\) fall within the validation gate of \(T_1\) whereas only \(z_2\) is inside the validation gate of \(T_2\). At time step \(k + 1\), we receive a single measurement \(z_3\) which falls within the validation gates of only a few tracks as shown in the figure. \(\emptyset\) represents that none of the measurements correspond to that track. \(T_3\), \(T_4\) and \(T_5\) are potential new targets. 108
Chapter 1

Introduction

1.1 Motivation

1.1.1 Micro Aerial Vehicles

Micro aerial vehicles (MAVs) in general and multirotor aircrafts specifically have received lot of interest recently since they are highly versatile flying platforms with a simple mechanical design. One of the main challenges with these flying robots has been to make them autonomous such that there is minimal to no human effort involved in flying them. We need robust solutions to the components of the autonomy system, namely state estimation, control, mapping, and planning, to enable the use of these systems in practical situations. This capability of autonomous flight can enable a variety of applications such as disaster response, infrastructure inspection, and agricultural monitoring which currently require a lot of direct human involvement and labor.

Ground robots have been used for some of these tasks for a long time but aerial robots have significant benefits over them in these applications. For monitoring or search and rescue applications, aerial robots can fly high and get a top-down view of the area providing a quick overview of the situation below. This ability to move in 3D allows aerial robots to navigate
terrains which are tough or even impossible for ground robots to cross, such as during a disaster response situation. Similarly, in a lot of infrastructure inspection applications, it is impossible to use ground robots due to the nature of the structure to be inspected. Due to their small size, these aerial vehicles can also fly through narrow gaps to enter a damaged structure, which may not have an easily accessible entrance for ground robots, and collect sensor data while flying inside. These advantages have made MAVs the platform of choice for such applications. However MAVs have one big disadvantage compared to ground robots and that is their limited operating time. This is especially true for multirotors which typically have a flight time of less than 30 min.

1.1.2 Teams of robots

The limited operating time of MAVs makes their use infeasible in applications where we need them to operate in large environments. We can increase the operating time of the MAV by using larger batteries but this reduces the payload capacity of the vehicle requiring the use of smaller and less capable sensors and computation onboard the robot. Another way to solve this problem is by using multiple vehicles such that the task can be split up among the team. This splitting up of the task among the team means that the requirements from individual members of the team can be lowered hence it allows each robot in the team to be made smaller and simpler. Using multiple robots together means that portions of the task can be done in parallel and the task can be completed in a shorter time. A team of robots working together can also accomplish tasks that would be infeasible for a single robot to do, such as carrying loads which are too heavy for a single robot. Moreover, a system with multiple robots is more robust than one with a single robot since the failure of one member of the team can be accommodated by redistributing the task among the remaining members while it would lead to a complete failure of the task in the case of a single robot.

There are many advantages of using a team of robots, but realizing a multi-robot system requires careful selection of system architecture and design of algorithms. Many times, the solutions that are developed for a single robot need to be extended or redesigned to be able
to handle the requirements of the multi-robot system. This is especially true for the state estimation and planning components of the system, where we need to explicitly handle the multi-robot case.

1.2 Problem Statement

The main objective of this thesis is to develop an autonomous team of aerial robots that,

- is able to navigate in GPS-denied environments,
- only uses local sensing for state estimation but is able to localize with respect to a common reference frame,
- can be commanded by a single operator through a simple interface to accomplish coordinated missions.

We achieve these goal in two steps. First, we focus on single robot autonomy enabling fully autonomous navigation of the robot in obstacle-rich environments. Once we have a robust solution for single robot autonomy, we address the problems of system design, state estimation and planning for a team of robots.

In order to create an autonomous team of robots, we first need to solve the problem of autonomy for a single robot. This means that we need to develop solutions for state estimation, control, and planning to enable the robot to autonomously navigate through cluttered environments. In addition to this choice of algorithms, the autonomy system also requires the proper selection of the sensors and computation that are used on the robot. This hardware selection is especially important since we want to keep the total weight of the computational payload low to maximize the operating time of each robot.

The autonomy system that we develop for single robots relies only on local sensing for state estimation due to its robustness but the problem with local sensing is that each robot has a different reference frame for its state estimate. We require coordination among the robots
to perform co-operative tasks, and an important requirement for such coordination in a centralized architecture is that the robots need to be able to localize themselves in a common reference frame. Without a common reference frame, the robots cannot have a consistent interpretation of the commands from the planner and cannot perform any coordinated task. Thus, there is a need to establish a common reference frame for the team of robots even though each robot only uses local sensing for state estimation.

Finally, in order to achieve the co-ordination, the multi-robot system requires a planner which takes the goals provided by the user, assigns them to the individual robots in an optimal way and generates smooth collision-free trajectories for each of the robots to follow. In this work, we assume that the robots in the team are interchangeable, which means all of them have the same capability and hence can be assigned to any of the goals given by the user.

1.3 Main Contributions

In this thesis, we present robust solutions for each of the components required for autonomy in a homogeneous multi-quadrotor team, with a primary focus on state estimation, and demonstrate their integration on an experimental testbed.

First we describe the four components, namely, state estimation, control, mapping, and planning, that are essential for an autonomous system and provide robust solutions to each of them for a MAV [73]. The main challenge with autonomy for a MAV is that the sensing and computation onboard the robot is limited due to limited payload capacity, hence we require algorithms that are able to run at high rates with limited computation. We integrate these on a robot and demonstrate its ability to navigate through unknown obstacle-rich environments at high speeds. We show experimental results for this system in a variety of environments ranging from forests to a multi-floor building [71, 82].

Next, we develop the system architecture and components to allow a single operator to deploy, control, and monitor a team of robots [72]. In order to focus on the multi-robot
coordination problem, the robots used for this part utilize GPS for state estimation and require known maps of the environment for obstacle avoidance, thus limiting the usage of the system to known outdoor environments. We demonstrate a planner for the team that takes in the set of goals for the team, handles the assignment of the goals to the robots and then plans trajectories for the robots while avoiding collisions with obstacles in the environment as well as other robots.

Finally, we combine the robots from the first part with this multi-robot framework. The main challenge in using the robots, which relying only on local sensing for state estimation, in the multi-robot scenario lies in state estimation. Since the robots only use local sensing for state estimation, the estimate is in a robot-specific frame whereas for coordination among the team we need to be able to localize the robots in a common reference frame. We propose a solution to this problem, without relying on any external features, by using the relative position/bearing measurements of the nearby robots detected in the onboard camera. The issue with applying this method for our team is that the robots in the team are interchangeable and look the same, hence the relative measurements do not contain any identity information. We use the Coupled Probabilistic Data Association Filter (CPDAF) for solving this anonymous measurement problem and develop an approximation to reduce its computational complexity enabling online estimation with a team of up to 15 robots.
Chapter 2

Related Work

The basic functionality of a mobile robot is to navigate from its current location to a designated goal. To perform this task autonomously requires a navigation system that is composed of the standard building blocks of state estimation, control, mapping and planning. Each of these blocks builds on top of the previous ones in order to construct the full navigation system. For example, the controller requires a working state estimator while the planner requires a working state estimator, controller and mapping system.

2.1 State Estimation

For mobile robots, the state primarily consists of the robot’s position, orientation, and velocity with respect to an inertial reference frame. It is the first component required for autonomous navigation. Even with the limited sensing and computation available, the state estimation system for a MAV needs to estimate the full 6 DoF pose, 3D linear and angular velocities of the robot.

2.1.1 Sensors

Aerial vehicles do not have any direct source of odometry such as the wheel odometry which is available on most ground robots. Instead, they use an IMU which consists of a 3D gyroscope...
and 3D accelerometer and measures the angular velocities and linear accelerations of the vehicle. By integrating these measurements, we can get a rough estimate of the velocities, position and orientation of the vehicle but this estimate suffers from drift due to the noise and time-varying biases in the IMU measurements. This is similar to using the wheel odometry of a ground robot to estimate its position and heading. Hence we require additional sensors to accurately estimate the state of the vehicle. Typical sensors which are used for this purpose on aerial vehicles are motion capture systems, GPS, Lidar and cameras. Each of these sensors has its advantages and disadvantages.

**Motion Capture**

A motion capture (MoCap) system consists of a set of fixed cameras observing the object to be tracked typically with a set of reflective markers attached to the object. The system works by observing the markers on the object from multiple cameras and then finding the best pose of the object which matches the camera observations. The main advantage of MoCap systems is that they provide a low latency and highly accurate estimate of the pose of the object at high rates. This allows the estimates from the system to be used as ground truth and hence these have been very successfully used for controls research. The big disadvantage of these systems is that it requires an elaborate and time-consuming process for setting up a new system. This has led to these systems almost always being used only in a dedicated indoor lab environment. These systems are also quite expensive and since multiple cameras need to see the object being tracked at every instance, the tracked volume for most systems is usually quite small. The performance of the system is dependent on accurately knowing the camera locations in the workspace hence the calibration procedure to determine the camera
locations needs to be run periodically. Due to all these disadvantages, the use of motion
capture systems is slowly declining in mobile robotics and especially aerial vehicle research.

**GPS**

GPS works by localizing the receiver using signals from a network of satellites orbiting the
earth. The main advantage of GPS is that it can provide a global position of the vehicle
almost anywhere on earth. In addition to position estimates, GPS receivers can also compute
the velocity of the vehicle using Doppler shifts in the received signals. The main disadvantage
of GPS is that it requires a clear view of the sky in order to receive the signals from the
satellites, hence can only reliably work outdoors. Even outdoors, in certain environments
such as urban canyons or near large metallic structures, the GPS quality can be very poor. In
addition to these problems, the accuracy of the GPS position estimates is at best around 1 m
which limits its use in applications that require high position accuracy. This accuracy can be
improved using technologies like DGPS and RTK-GPS. These can improve the positioning
accuracy to somewhere between 2–20 cm but they require a continuous data stream from a
separate base station which has its absolute position on the earth known with a high degree
of accuracy. The main problem with this is that obtaining the accurate position of the base
station requires averaging its GPS position for around 12–18 hours.

**Lidar**

Lidar sensors use spinning lasers with detectors to measure the distances to objects around
them. Most of the current lidar sensors only operate in 2D, i.e. they only provide the distance
measurements to points in a plane as the laser spins in a circle, though 3D lidar sensors
are becoming more popular. Lidar sensors are used for pose estimation by matching the
current scan (a full circle of readings) to previous scans or a map. Once motion of the
robot is computed, the map can be extended by adding the new scan into the map. The
main advantage of lidar sensors is that they are long-range and accurate sensors providing
data that can be easily processed for pose estimation and mapping. Since lidar sensors are
active sensors, they can be used in places without any external illumination allowing their
use in places where it can be hard for humans to go. Thus these sensors can be carried by
the vehicle and used for localization without requiring any external infrastructure. Their
big disadvantage is that these sensors are large and bulky, especially the 3D ones, which
restricts their use on small aerial vehicles. Moreover, using the smaller 2D lidar sensors for
aerial vehicles require certain assumptions to be made about the environment in order to
localize the vehicle in an unknown map, for example, the popular 2.5D assumption about the
environment looking the same in any horizontal cross-section between the floor and ceiling.
This prevents their use in unknown and unstructured environments.

Cameras

Cameras capture the amount of light falling on their sensors and output images representing
this light intensity. As the robot with a camera moves, the scene captured by the camera
changes and using the images, visual odometry algorithms are able to infer the motion of the
robot. These algorithms typically work by first extracting salient features in the image, such
as corners and lines, and then tracking their motion across frames as the camera moves. Using
the motion of these features, we can estimate the motion of the camera between these
frames.

Cameras have become very popular in robotics due to their small size and rich information
output. With a mass of only a few grams, cameras can be used on the smallest of the robots.
The main disadvantage of cameras is that they require a significant amount of processing to
calculate the motion from the images. This is a declining concern since the processing power
available on small processors has been increasing with time, mainly due to the popularity of
smartphones, and we already have credit-card size boards which can run most of the current
visual odometry algorithms.

There has been a lot of research in recent times on visual and visual-inertial odometry for
MAVs with a variety of proposed algorithms [11, 27, 46, 53, 74, 76]. The algorithms can be
classified based on the number of cameras required into three groups: monocular, stereo, or
multi-camera. Note that by multi-camera we mean a system with more than two cameras
arranged such that the cameras are looking in different directions with each camera’s field of view overlapping at least partially with another camera. There are also algorithms using depth cameras but these cameras only work well in small indoor environments so we do not consider them. An overview of the advantages and disadvantages of the algorithms is shown in Table 2.2. The mechanical complexity is due to the requirement of keeping the cameras and IMU rigidly attached such that the camera-to-camera and camera-to-IMU transformations are constant during operation. The monocular camera has an obvious advantage in this regard and the complexity increases as the number of cameras in the system increases.

In terms of software complexity, stereo methods have the advantage that we can get the depth of features in the image directly by stereo matching instead of relying on structure-from-motion type techniques required for monocular or multi-camera methods. In some cases, using the stereo setup as two independent monocular cameras with a known constraint between them is advantageous [95], where the software complexity is similar to the monocular case. Compared to the monocular and stereo methods, the complexity involved in handling the feature tracking and matching in a multi-camera method is significantly higher which increases the software complexity. The main advantage of multi-camera methods is their increased robustness due to the ability to track features in different directions preventing the loss of features tracking in case there is an untextured surface in the field of view of one of the cameras. This situation can lead to complete failure for a monocular or stereo method. Monocular algorithms have the additional problem of requiring an initialization process during which the estimates are either not available or are not reliable. In comparison, the stereo and multi-camera algorithms can be initialized using a single frame making them much more robust in case the algorithm needs to be reinitialized while flying, for example, if there is a sudden rotation leading to motion blur and all the tracked features being lost. Thus in terms of robustness the multi-camera methods rank the highest, followed by stereo, and the monocular methods are lowest.

Instead of the camera being used by itself, we can utilize the IMU on the robot to aid the
camera in estimating the motion of the robot. There are two types of methods for combining the information from the cameras and the IMU, namely *loosely-coupled* and *tightly-coupled*. In a *loosely-coupled* approach, the vision module provides pose estimates which are combined with the IMU measurements in a separate sensor fusion module whereas a *tightly-coupled* VIO system combines the measurements from both camera and IMU in a single estimator. There has been a lot of research in recent times on *tightly-coupled* VIO systems [11, 46, 52, 59, 60, 74, 81] and the *tightly-coupled* methods have been shown to have better accuracy than *loosely-coupled* ones [59].

The *tightly-coupled* VIO methods are broadly divided into two types, *filter based* [11, 46, 52, 74] and *optimization based* [59, 60, 81]. Filter based methods are typically based on the EKF framework where the IMU is used for prediction and camera measurements used for the state update. In comparison, optimization based methods explicitly solve the non-linear VIO problem and hence avoid the linearization errors that occur in the filtering framework. Thus, optimization based methods generally produce more accurate results compared to filter based methods but with the disadvantage of requiring higher computational resources than filter based methods.

Recently, there has been interesting research on directly using the image intensities without extracting any features for estimating the camera’s motion [21, 23, 105]. These methods have been shown to be more robust than feature based methods, especially in low texture environments, since they are able to use more information from the image by utilizing all the gradients in the image. The main disadvantage of these direct methods is that they require much higher computational resources compared to indirect (feature-based) methods, thus

<table>
<thead>
<tr>
<th></th>
<th>Monocular</th>
<th>Stereo</th>
<th>Multi-camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanical</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Medium</td>
<td>Low/Medium</td>
<td>High</td>
</tr>
<tr>
<td>complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 2.2: Advantages and disadvantages of different visual odometry configurations.
limiting their use on platforms with tight computational constraints.

### 2.1.2 Sensor fusion

Robots typically have multiple sensors, in some cases because some sensors only provide partial information about the state to be estimated and sometimes even if each sensor is sufficient to estimate the complete state, to improve the quality of the estimates or to provide robustness against failures of one of the sensors. The information provided by each of the sensors needs to be merged into a single estimate of the state of the robot. This task of sensor fusion can be accomplished using recursive bayesian filtering methods, such as Kalman filters, or factor graph based methods.

Kalman filters are very popular due to their ease of use and good performance. Even though the Kalman filter was originally proposed only for linear systems [49], new extensions such as the Extended Kalman filter (EKF) [45] and Unscented Kalman filter (UKF) [47], have been made for applying them to non-linear systems. All of them maintain an estimate of the current state of the system and also the uncertainty of this estimate, represented as a covariance matrix. They work in a two-step process. The first is called the prediction step where an approximate model of the system is used to estimate the next state of the system, given the current state estimate and any external inputs to the system. Once a measurement from any of the sensors is received, the second step, called the update step, of the Kalman filter is run which updates the estimated state using the new measurement with a weight determined by the uncertainty of the measurement and the previous state estimate. The Kalman filters are recursive in nature, only requiring the previous state estimate and current measurement, leading to constant time updates and allowing them to run real-time on devices with very low computational power. The problem with using the EKF or the UKF for non-linear systems is that they fix the previous state estimate and only update the current estimate when the measurement is received leading to non-optimal estimates even in the presence of Gaussian noise.

Factor graphs in comparison create an optimization problem involving the estimated state at
each time step and the measurements received from the sensors. The states at each time step are related through a transition function which can be used to predict the next state of the system given the current state and the inputs to the system. The measurements at each time step are related to the states by a measurement function which outputs an expected measurement given the estimated state. Thus the cost function for the optimization consists of terms containing the difference between the predicted states at each time step and the estimated state at that time, and the difference between the measurements received from the sensors and the predicted measurements from the estimated states. Solving this non-linear optimization problem leads to the optimal estimates for the state at each time step. The main disadvantage of factor graphs is that the computational requirement slowly increases as the graph grows in size over time. There have been some advances in methods for incremental optimization of factor graphs, such as iSAM2 [48], which reduce the computational complexity by only solving a part of the complete optimization problem and providing a good approximate solution.

2.2 Control

The dynamics of the quadrotor are nonlinear due to the rotational degrees of freedom. In the control design for these robots, special care has to be taken in order to take this nonlinearity into account in order to utilize the full dynamics of the robot. Most early works in control design for quadrotors [13–15, 25, 38] used the small angle approximation for linearizing the dynamics around the hover state and applied controllers developed from linear system theory, such as PID, backstepping, and LQR controllers, to stabilize the simplified system. Due to the small angle assumption, these controllers are not able to handle large orientation errors and have large tracking errors for aggressive trajectories. More advanced control methods take the non-linear dynamics into account and are able to stabilize the robot even from states which are far from the hover state. A nonlinear controller using quaternions (instead of Euler angles) was developed in [34] where the quadrotor was commanded to follow velocity commands. [58] defined an orientation error metric directly in the SO(3) space and proposed
a globally asymptotic controller that can stabilize the quadrotor from large position and orientation errors.

2.3 Mapping

Once the robot is able to localize itself, the next step in navigation is to construct a map of the environment. This needs to be a dense map for the planner to be able to guarantee safety. The sensor data is typically in the form of a pointcloud but accumulating the pointclouds from each sensor reading to create the map would lead to a huge number of points in the map. This map requires sparsification in order to be feasible for planner to use. With pointclouds one common way to do this is by using a filter that overlays a grid on top of the pointcloud and merges the points lying inside each grid cell by replacing the points with a single point at their centroid.

The map represented as a set of points has the limitation that it only indicates the occupied parts of the environment and does not explicitly label the free and unknown areas which are useful for planners. The simplest representation for the map which provides this is a uniform grid where each grid cell denotes whether that location is free, occupied or unknown. Instead of this three state map, a probabilistic representation can be used where each cell stores the probability of being occupied. This probabilistic representation is useful when merging multiple sensor readings into the map since it is able to take the noise in the sensor into account when merging the new sensor readings. The main disadvantage of the uniform grid representation is that the memory requirement grows very quickly as the map size increases. Octree-based solutions provide an improvement over the uniform grid map in terms of the memory required for representing the maps which is particularly useful when dealing with large outdoor maps. Octomap [39] is a recent implementation of this idea that has become quite popular in the robotics community.
2.4 Trajectory Planning

In order to move through a map with obstacles, the robot needs to plan a collision-free trajectory that satisfies its dynamics. A piecewise linear trajectory that a path planner provides is not ideal for a multirotor MAVs since they are fourth order systems and require trajectories that are continuous in jerk, the third derivative of position, to be able to follow them. For following a piecewise linear trajectory, the robot would need to come to a stop at points where the direction changes since the trajectory has infinite curvature at these points. Even with sampling based methods such as RRT*\[50\], planning in the full state space of the robot is not feasible due to the 12-dimensional nonlinear dynamics. Thus we require a trajectory generation algorithm which is able to efficiently generate smooth (up to the third derivative) trajectories while avoiding obstacles in the map.

2.4.1 Optimization based

The earliest work on trajectory generation for quadrotors formulated the problem as a Quadratic Program (QP) \[69\]. This formulation represents the trajectory as a piecewise polynomial function with the cost function defined as the integral of the squared norm of the snap along the trajectory. Linear constraints are added to enforce continuity of the derivatives at the boundaries of the polynomial segments as well as to make the trajectory pass through specific points or corridors. This work was mainly about generating trajectories and did not take the obstacles in the environment into account.

In order to handle obstacles, \[68\] proposed a Mixed Integer Quadratic program (MIQP) based approach where integer constants are used to enforce collision avoidance. This formulation was able to plan collision-free trajectories for a team of heterogeneous robots in an environment with obstacles but the main disadvantage of this work was the extremely slow run time due to solving a large MIQP problem. Another problem with this method is that it requires the map to be known before planning, which is unrealistic in practical applications where the robot builds up the map as it moves through the environment and gathers sensor data.
In [85], the authors extended the original QP based trajectory generation work by improving the numerical stability of the solution, specially in cases with large number of segments, and coupled it with a RRT* based path planner in order to generate smooth, collision-free trajectories. Their method used waypoints from the RRT* path and generated a trajectory passing through these waypoints. This did not guarantee that the trajectory would be collision-free since the it could curve between waypoints and go through nearby obstacles. In order to solve this, they add new waypoints in between the previous waypoints to force the trajectory to be close to the path generated by RRT*. In environments with a high obstacle density, this process needs to be repeated multiple times eventually leading to a trajectory which hugs the planned path closely.

A new approach for trajectory for MAVs was proposed in [20] where the authors constructed a convex decomposition of the free space to enforce convex constraints on the trajectory segments in order to guarantee collision avoidance. Their approach requires solving a mixed-integer convex optimization and while it is much faster than the MIQP solution proposed in [68], it is too slow to run online as the robot is flying. Simpler approaches to constructing such Safe Flight Corridors (SFC) were proposed in [16, 63] which led to significantly faster computation and allowed these methods to be used online as the robot moves through the environment. These methods first use a graph search based planner on a discretized representation of the environment to find a path to the goal and then construct the SFC around this path. After generating the SFC, a QP is constructed with to keep the trajectory inside it while also setting limits on the higher derivatives (velocity, acceleration, and so on) of the trajectory to make it dynamically feasible for the robot.

2.4.2 Search based

Due to their fast run times (less than 0.5 s), the SFC based methods can be used to continuously replan as the robot moves through the environment and builds up a map, but as the robot’s speed is increased these planners start failing. This is mainly due to the path from a path planner being used to initialize the SFC construction, which implicitly assumes
that the robot is stationary at the start. Since the initial velocity is not taken into account when generating the path, it can lead to the SFC being small in the direction of initial velocity and requiring large, unfeasible control inputs to stay within the SFC, especially in cases when the robot replans while moving at a high speed.

In order to solve this issue, [62] developed a method to directly search for a valid trajectory instead of the two step process used in the QP approach of first finding a path and then generating the trajectory using the SFC around the path. Instead of using the prior path, this method uses short-duration constant-input motion primitives, introduced in [75], to directly explore the space of trajectories. The motion primitives are generated by discretizing the input space and applying the discretized inputs to the current state of the robot. This in turn induces a discretization on the state space and allows the use a graph search algorithm to find safe, dynamically feasible and optimal trajectories. In order to check for collisions, points on the short trajectories generated by each of the primitives are sampled and the ones that pass through an occupied node in the graph are rejected. In addition, each of the short trajectories can be checked for the maximum velocity, acceleration and so on, in order to enforce the dynamic limits of the robot. Thus, during the graph search, only the neighbors which are reachable from the current state are added to the search queue while guaranteeing safety and feasibility. This allows us to find a trajectory which is globally optimal (up to a resolution), safe and feasible for the robot to follow.

2.5 Multi-robot UAV systems

The study of teams of aerial robots is the main focus of our work. The earliest works using teams of UAVs for demonstrating co-ordination and control used small fixed wing aircraft for their experiments [42, 101]. However, these fixed-wing aircrafts are expensive, complex, and require large open spaces to operate. Multirotor UAVs, in comparison, are simple, inexpensive and able to operate in small environments. This has led to their increasing popularity when studying multi-robot systems.
[37] demonstrated one of the first hardware testbeds using a team of multirotor UAVs. This was an outdoor system consisting of up to four UAVs, with the robots using GPS and IMU for state estimation.

Motion capture systems provide accurate, low latency and high frequency estimates of the position and orientation of the robots only requiring the addition of small reflective markers on them. [41, 65, 70] were among the first works using motion capture systems for multi-robot systems and it has now become the most popular choice for state estimation in multi-robot research.

Compared to motion capture system and GPS, the use of vision for state estimation in a swarm of aerial vehicles has been less common. The sFly project [1, 91] was one of the first to demonstrate a team of aerial vehicles flying using only vision-based state estimation. Recently, [115] demonstrated formation flying and transitions between formations with a team of 12 quadrotors each using vision and IMU as their only source of estimation.

There has been a steady increase in the number of robots flown simultaneously through the last few years. [106] demonstrated flocking behavior and formation flight with a team of 10 robots using GPS as the primary sensor for estimation. A team of more than 30 small quadrotors was flown as a demonstration during a TED talk [19] using ultra-wideband radios for localization which require much lower infrastructure compared to setting up a motion capture system. [80] describes a system architecture for state estimation, control, and planning for a swarm and demonstrated it using a team of 49 miniature quadrotors using a motion capture system for state estimation. Recently, Intel conducted a light show with 2018 quadrotors [43], making it the largest swarm of aerial vehicles ever flown. The robots used GPS for localization and are controlled through commands sent from a centralized planner.

2.5.1 Planning

The correct method of planning for a team of robots by planning in the joint state space of the robots quickly becomes infeasible as the number of robots in the team increases since
its computational complexity grows exponentially with the number of robots. This has led to development of many approximations that reduce the computational complexity while trading in either completeness, optimality, or both to achieve that.

The simplest type of approximation are reactive planners which ignore the presence of the other robots unless any of them gets close to the robot and when that happens, the planner computes a local perturbation to avoid the collision and return to the planned trajectory. Examples of this approach include the Velocity Obstacles [26], ORCA [9], and Generalized Reciprocal Collision Avoidance [6] methods.

Another approximation that is commonly employed [7, 8] is to assign a priority to the robots and sequentially plan for the robots where the higher priority robots ignore the lower priority ones when planning their trajectories while the lower priority robots assume that the higher priority robots are moving obstacles when generating their own motion plans. Unfortunately, this method loses both optimality and completeness in order to gain computational efficiency.

In comparison, [56] demonstrated trajectory generation and control for a team of 20 robots by extending the Mixed-Integer Quadratic Program (MIQP) formulation developed in [68]. The primary approximation used in their method is the discretization in time to generate a finite set of constraints for collision avoidance. The main disadvantage of this method is that it is very computationally intensive, requiring multiple minutes to find the solution.

In a multi-robot system with homogeneous robots, the assignment of the goals to the robots is flexible. This flexibility can be directly exploited to generate collision-free motion plans for the robots in obstacle-free environments [103] or combined with a prioritization scheme for environments with obstacles [104].

2.5.2 Localization

A multi-robot system requires accurate localization of the individual robots in order to perform any high-level task. Ideally, we want the localization of all the robots in a common reference frame such that a centralized planner may be used for the team.
Using sensors such as GPS, motion capture systems or ultra-wideband radios for localization solves this problem since they directly provide state estimates in a common reference frame. Another similar method is to add fiducials, such as AprilTags [77], to the environment which can be used by the robots to localize themselves with respect to and hence infer a common reference frame. The main disadvantage of these methods is that they depend on external infrastructure in the form of either GPS signals, setting up a camera system for motion capture, or modifications to the environment to add the fiducials which may not be work for certain applications.

If the robots use local sensing, such as vision or lidar, for state estimation, then each robot would have a different reference frame with the origin typically being at the starting location of the robot. There are multiple methods to solve the problem of establishing a common reference frame across all the robots. The simplest solution is to start the robots at known locations such that we can add the initial offsets to the individual robot states to obtain the estimates in a common reference frame. This was the method used in [115] for a team of 12 robots each using vision for state estimation. In many cases, this method is not desired or practically difficult since it requires careful initial placement of the robots.

Typical methods for localization when using vision or lidar sensors, require the creation of a (potentially sparse) map of the environment. One way for the robots to localize themselves in a common reference frame is by sharing their respective maps with other robots or a common ground station where matches in the maps from different robots can be used to estimate the relative offsets in the robot state estimates. This approach has been demonstrated for robots using monocular cameras [28, 91] as well as RGBD cameras [64].

Another solution to this problem of establishing a common reference frame is to detect other robots and localize using relative measurements of the other robots location. This is also known as cooperative localization and has been studied extensively [30, 87, 96]. The most common approach for this has been to put fiducials, such as AprilTags [77], circular patterns [54], or colored markers on each of the robots and detect them using cameras on the other
robots. The main disadvantage of this approach is that we need a unique fiducial for each robot in the team, with sufficient difference between the fiducials to be able to identify the robot’s identity from distant measurements.

In comparison, in this thesis, we describe algorithms to estimate the positions of the robots in a team using unlabeled relative position measurements between them, which allows us to remove the requirement of having unique fiducials for each of the robots in the team. There have been only a few works [31, 32, 97] using the unlabeled relative measurements for multi-robot localization which have shown promising results. These approaches rely on specific geometric constraints involved in the relative position or bearing measurements and are non-trivial to extend to different sensing modalities. In comparison, our work uses methods developed for generic sensor models allowing it to be easily applied to other types of sensors.
Chapter 3

Preliminaries

3.1 Notation

We first describe the convention for the mathematical notation used in this thesis.

Unless otherwise noted specifically, we use

- lowercase italics for scalars \((a, b, c)\)
- lowercase bold italic for vectors \((x, y, z)\)
- capital italic for matrices \((M, N, P)\),

and similarly for scalar, vector, or matrix-valued functions.

We use \(0_{n \times m}\) to represent a \(n \times m\) matrix of zeros and the short-hand \(0_n\) when \(m = n\). Similarly, \(I_n\) is used to denote the \(n \times n\) identity matrix.

We represent the Gaussian probability distribution as \(N(\mu, \Sigma)\), where \(\mu\) is the mean, and \(\Sigma\) is the covariance. A random variable following this Gaussian probability distribution would be denoted as \(v \sim N(\mu, \Sigma)\).
The rotation matrix that takes the co-ordinates of a vector expressed in frame $B$, $^Bv$, and transforms them into co-ordinates expressed in frame $A$, $^Av$, is written as $^AR_B$, i.e. $^Av = ^AR_B^Bv$. This matrix, $^AR_B$, also represents the orientation of frame $B$ in frame $A$, meaning that the columns of the matrix are the co-ordinates of the $X$, $Y$ and $Z$ axes of frame $B$ expressed in frame $A$.

We denote three rotation matrices corresponding to the rotations about the $X$, $Y$ and $Z$ axes as,

\[
R_x(\phi) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\phi) & -\sin(\phi) \\
0 & \sin(\phi) & \cos(\phi)
\end{bmatrix},
\]
\[
R_y(\theta) = \begin{bmatrix}
\cos(\theta) & 0 & \sin(\theta) \\
0 & 1 & 0 \\
-\sin(\theta) & 0 & \cos(\theta)
\end{bmatrix},
\]
\[
R_z(\psi) = \begin{bmatrix}
\cos(\psi) & -\sin(\psi) & 0 \\
\sin(\psi) & \cos(\psi) & 0 \\
0 & 0 & 1
\end{bmatrix}.
\]

We adopt the Z-Y-X intrinsic Tait-Bryan angles convention when we use the Euler angles representation for rotations. This means that if $(\phi, \theta, \psi)$ are the Euler angles for the rotation representing the orientation of frame $B$ in frame $A$, then to go from frame $A$ to $B$, we first rotate $A$ by the angle $\psi$ about its $Z$-axis, then we rotate by $\theta$ about the new $Y$-axis after the first rotation, and finally we rotate by $\phi$ about the new $X$-axis after the second rotation. An example of this rotation is shown in Figure 3.1. Thus the complete rotation matrix corresponding to Euler angles $(\phi, \theta, \psi)$ can be written as by $^A R_B(\phi, \theta, \psi) = R_z(\psi)R_y(\theta)R_x(\phi)$.

We use $[\cdot]_\times$ to denote the operator that converts a vector in $\mathbb{R}^3$ to a $3 \times 3$ skew-symmetric matrix that is used to represent the cross product as a matrix multiplication, i.e.,

\[
x \times y = [x]_\times y.
\]
Given a vector \( \mathbf{x} \in \mathbb{R}^3 \), this matrix is constructed as follows,

\[
\mathbf{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \implies [\mathbf{x}]_x = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}.
\]

### 3.2 Reference Frames

On a typical robot, we have many sensors, each having its own reference frame. Here we define some of the common frames that would be referred to in this thesis.

The *world frame*, denoted by \( \mathcal{W} \), is defined as the inertial frame that is used as a common
reference frame for the whole system. We assume that it is fixed to the earth, neglecting the small effects of the rotation of the earth, with the $Z$ axis pointing upwards opposite to the direction of gravity.

The *robot frame* or the *body frame*, denoted by $\mathcal{B}$, is a frame rigidly attached to body of the robot and is used to represent the position and orientation of the robot. In this thesis, unless explicitly mentioned, when we refer to the robot/body frame, we are referring to the frame corresponding to the IMU of the robot.

For robots having a camera, we define a *camera frame*, denoted by $\mathcal{C}$, such that the $Z$ axis of this frame points along the optical axis of the camera, the $X$ axis is aligned with the horizontal axis of the camera sensor, and the $Y$ axis is aligned with the vertical axis.

### 3.3 Quadrotor Dynamics

The platform of choice for most of the experiments in this work is a quadrotor. Here we provide a brief summary of the dynamics of the quadrotor which is used for the development of state estimation and control algorithms for the robot.

The control input for the robot is the individual motor speeds, but we can convert it into individual motor force and moment using a simplified aerodynamic model. Figure 3.2 shows the forces and moments generated by the motors of the quadrotor. The force produced by each motor is given by $f_i = k_f \omega_i^2 \mathbf{b}_3$ while the moment produced is $\tau_i = (-1)^{(i-1)} k_m \omega_i^2 \mathbf{b}_3$ where $\omega_i$ is the angular speed of the $i^{th}$ motor, and $k_f$ and $k_m$ are propeller specific constants. We assume that the motor speeds only produce thrust in the positive $\mathbf{b}_3$ direction.
Figure 3.2: The forces and moments generated by the individual motors of a quadrotor. The vectors $b_1$, $b_2$, $b_3$ and $w_1$, $w_2$, $w_3$ are the basis vectors for the body frame and world frame respectively.

The robot follows the rigid body dynamics,

\[
W \dot{p} = W v, \\
W \dot{v} = -g + \frac{1}{m} W f, \\
W R_B = W R_B [B \omega]_\times, \\
B \dot{\omega} = I^{-1} \left( B \tau - B \omega \times I \omega \right),
\]

where $W p$ and $W v$ are the position and velocity of the robot expressed in the world frame, $g = [0 \ 0 \ 9.81]^T$ is the acceleration due to gravity, $m$ is the mass of the robot, $W f = \sum_i f_i$ is the total force generated by the motors expressed in the world frame, $W R_B$ is the orientation of the robot in the world frame, $B \omega$ is the angular velocity of the robot expressed in the body frame, $I$ is the moment of inertia of the robot, and $B \tau$ is the total moment generated by the motors expressed in the body frame. The total moment, $B \tau$, is computed as

\[
B \tau = \begin{bmatrix}
      l \cdot (\|f_2\| - \|f_4\|) \\
      l \cdot (\|f_3\| - \|f_1\|) \\
      \|\tau_1\| - \|\tau_2\| + \|\tau_3\| - \|\tau_4\|
    \end{bmatrix},
\]

where $l$ is the length of the arms of the robot.
Chapter 4

Single Robot Autonomy

The main challenge for a navigation system for micro aerial vehicles (MAVs) is the limited sensing and computation capabilities due to the constraints on the payload capacity. Ground robots typically have a payload capacities of a few kilograms while a MAV typically has a payload capacity of up to 1 kg with the smaller ones only able to carry up to 0.2 kg. Compared to ground robots, the feedback loops on a MAV need to run at a much higher rate due to the fast and inherently unstable dynamics of the system. Thus, a careful selection of sensing, computation, and algorithms has to be made in order to create a successful navigation system for MAVs.

In this chapter, we provide a detailed description of a quadrotor system that is able to navigate to a goal at high speeds in fully unknown and cluttered 3D environments while using only onboard sensing and computation for state estimation, control, mapping and planning. The motivation for this problem comes from the DARPA Fast Lightweight Autonomy (FLA) program\(^1\).

Our initial system consists of a DJI F450 frame with the DJI E600 propulsion system upon which we mount our sensing and computation payload. This consists of a stereo camera

\(^1\)http://www.darpa.mil/program/fast-lightweight-autonomy
synced with an industrial grade IMU and a laser based height sensor for state estimation, a nodding Hokuyo lidar for mapping, an Intel NUC i7 kit for handling all the high-level computation, and a Pixhawk autopilot for low-level interfacing and control. We explain the selection of the components of our sensing and computation payload in the following section.

4.1 Sensing, Computation and Communication

As mentioned earlier, the main challenge in creating such small, completely autonomous MAVs is due to the size and weight constraints imposed on the payload carried by these platforms. This restricts the kinds of sensors and computation that can be carried by the robot and requires careful consideration when choosing the components to be used for a particular application. Also, since the goal is fast flight, we want to keep the weight as low as possible in order to allow the robot to accelerate, decelerate and change directions quickly.

The two tasks that the robot has to perform which require proper sensor selection are state estimation and mapping. The two solutions for state estimation for MAVs are either vision based or lidar based. For unstructured 3D environments, the vision based systems have been more successful that lidar based ones, so we decided on using cameras as our primary state estimation sensor.

From Table 2.2 it is clear that a multi-camera setup would be the most preferred for visual odometry but the software complexity is still a hurdle in terms of real-world usage. Even though monocular algorithms have received a lot of research attention in the last few years and have improved to a level that they can be used as the only source of odometry for a MAV system, their robustness is still lower than stereo methods. One more advantage of using stereo cameras is that in the extreme case that stereo matching is not possible due to features being too far away, we can use the input from only one of the cameras from the stereo pair and treat it as a monocular camera setup. Thus we chose to use a stereo camera combined with an IMU as our main sensor for state estimation on the robot. In addition to these, we added a downward pointing lidar in order to improve the height estimate.
We require good time synchronization among the cameras and between the cameras and the IMU to achieve good estimation performance when using multiple cameras for VIO. This is done by triggering all the sensors using an external pulse. This pulse can be generated by a separate micro-controller but it is common to have it generated by the IMU which is already part of the system. In our case, we configured the IMU to generate a pulse when it samples data and use this to trigger the capture from the cameras in order to have this time synchronization. The trigger pulses from the IMU to the cameras are set up such that a pulse is sent once every few IMU samples so that the camera rate is an integer fraction of the IMU rate. This is done to keep the camera frame rate in a reasonable range (between 10–50 Hz) even when the IMU is sampling at a high rate (more than 100 Hz).

The situation for mapping is a bit different. Current vision based dense mapping algorithms are either not accurate enough or too computationally expensive to run in real time, so lidar or depth camera based mapping is still the preferred choice for MAVs. The problem with depth cameras is that their sensing range is quite short, typically around 5 m, hence they are not suitable for high-speed flight. Thus we chose to use a lidar for mapping, but in order to keep our weight low, we decided to use a Hokuyo 2D lidar (Hokuyo UTM-30LX) instead of a heavy 3D lidar. The issue with using a 2D lidar is that we require a 3D map for planning, so we decided to mount the 2D lidar on a one degree of freedom nodding gimbal as shown in Figure 4.1. This allows us to tilt the 2D lidar up and down which combined with the 270° field of view of the lidar provides a good 3D map of the area around the robot. Our mapping solution, including the gimbal, has a total mass of around 250 g which is much lower than the lightest 3D lidar available on the market, the Velodyne Puck LITE, which weighs around 600 g.

In order to handle all the computations for estimation, control, mapping and planning onboard the robot, we selected the Intel NUC i7 computer. This single board computer is based on the Intel i7-5557U processor and supports up to 16GB of RAM and an M.2 SSD for storage. This provides sufficient computing power to run our full software stack on the robot.
without overloading the CPU and also gives us ample amount of storage for recording sensor
data for long flights. While the robot is flying, we need to have a communication link in order
to monitor the status of the various modules running on the robot. We wanted a link that
has good bandwidth, so that during development we can stream the sensor data back to the
base station, but also good range so that we do not loose the link when running long range
(up to 200 m) experiments. In addition, since we use ROS as our software framework, having
a wireless link that behaves like a wireless local area network was preferred in order to be able
to use the standard ROS message transport mechanism. Based on these requirements, we
selected the Ubiquity Networks Picostation M2 for the robot side and the Nanostation M2 for
the base station. These are high power wireless radios that incorporate Ubiquity Networks’
proprietary airMAX protocol, which improves latency and reliability for long range wireless
links compared to the 802.11 protocol, which was designed mainly for indoor use. The
Picostation is the smaller and lighter of the two, weighing at around 50 g (after taking off
the outer plastic case) compared to 400 g for the Nanostation. This lower weight comes with
the compromise of lower transmit power and lower bandwidth, but the performance was
sufficient for our purpose, providing a bandwidth of more than 50 Mbps up to distances of
200 m.
4.2 Software Architecture

Figure 4.2 shows a high level block diagram of our system illustrating the different components and how they are connected to each other. The software components in our system can be grouped under four categories: Estimation, Control, Mapping and Planning. Each of these is in turn separated into smaller parts, and we use ROS as the framework for all the high level software running on the robot. ROS is chosen because it provides a natural way to separate each component into its own package allowing distributed development and ease of testing and debugging. A ROS system can be thought of as a computational graph consisting of a peer-to-peer network of nodes processing and passing data among them. One convenient feature of this system is that the nodes can be run on different computers, since the message passing uses the TCP transport, which allows us to run a subset of the nodes on the robot while the remaining can be run on a workstation computer making it easier to analyze and debug problems leading to a faster development phase. We also benefit from the whole ROS ecosystem of tools and utilities that have been developed in order to perform routine but useful tasks when developing a system such as tools for logging and playing back the messages passed between nodes or tools to visualize the data being sent between nodes.
4.3 State Estimation

In order to perform any navigation task, the robot needs to know its position and velocity. In addition, for controlling a flying robot, we need to know the orientation of the robot with respect to gravity. Thus, the state estimation task for our robot consists of estimating the position, orientation, and velocity of the robot with respect to an inertial reference frame.

4.3.1 Visual Inertial Odometry

The stereo cameras are the main source of state estimates for our system. This requires a VIO algorithm that is accurate and efficient. Given the different kinds of environments that we want the robot to operate in, the VIO algorithm needs to be robust as well. We decided to use a tightly-coupled VIO system, which combines the measurements from both camera and IMU in a single estimator, instead of a loosely-coupled approach, where the vision module provides pose estimates which are combined with the IMU measurements in a separate sensor fusion module, since tightly-coupled methods have been shown to have better accuracy than loosely-coupled ones [59].

Unfortunately, only a few of these algorithms have efficient open-source implementations available and most of them only work with monocular cameras. Very few algorithms are designed specifically for stereo or multi-camera setups mainly due to higher computational cost of feature detection and matching across the cameras. Hence we developed our own stereo VIO system based on the MSCKF algorithm [74] while incorporating improvements proposed in [36]. A filter based approach was chosen due to its computational efficiency compared to optimization based methods. This is because we need to run our full navigation stack on the onboard computer of our aerial robot, hence computational efficiency of each of the algorithms running on the robot was an important factor for us.

Our stereo VIO algorithm uses FAST features and tracks them across time using KLT on the left camera frames. In order to perform stereo matching, we again use KLT with FAST features from the left camera image to the right camera image. Descriptor based methods
have shown to perform better than KLT in terms of tracking and matching accuracy but in our experiments, we found that they have much higher computational requirements with only a small gain in accuracy making them unfavorable to run in the limited computational budget we have. We use a two step outlier rejection strategy consisting of a 2-point RANSAC during temporal tracking and a circular matching between the previous and current stereo image pairs during the stereo matching.

In addition to the components of the state described in the original MSCKF[74], we add the relative transform between the left camera in the IMU frame to the VIO state. This transform can be estimated online and hence requires only an approximate guess as the starting value. We only add the transform for left camera to the state since we assume that the stereo extrinsics between the left and right cameras are known.

The prediction step of the filter using the IMU measurements is the same as the MSCKF algorithm. The difference is that instead of the monocular measurement model used in the original MSCKF algorithm, we use a stereo measurement model:

$$z = \begin{bmatrix} u_L \\ v_L \\ u_R \\ v_R \end{bmatrix} = \begin{bmatrix} 1 \\ \begin{bmatrix} L & 0 & 2 \\ 0 & L & 2 \\ 0 & 0 & Z \end{bmatrix} I_{2\times2} \\ \begin{bmatrix} 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} I_{2\times2} \end{bmatrix} \begin{bmatrix} L_x \\ L_y \\ L_z \\ R_x \\ R_y \\ R_z \end{bmatrix}$$

where \((u_L, v_L)\) and \((u_R, v_R)\) are the normalized image co-ordinates while \((L_x, L_y, L_z)\) and \((R_x, R_y, R_z)\) are the 3D position of a feature in the left and right cameras respectively.

Given the 3D position of a feature in the world frame, \(Wp_j\), which can be triangulated using the history of estimated camera poses by the least squares method shown in [74], we can compute the 3D position of the feature in the left and right cameras as,
\[ Lp_j = \begin{bmatrix} L_X \\ L_Y \\ L_Z \end{bmatrix} = C \left( \frac{L}{q} \right) \left( Wp_j - Wp_L \right) \]

\[ Rp_j = \begin{bmatrix} R_X \\ R_Y \\ R_Z \end{bmatrix} = C \left( \frac{R}{q} \right) \left( Lp_j - Lp_R \right) \]

where \( C(q) \) represents the rotation matrix corresponding to the quaternion \( q \), the terms \( Lq_W \) and \( Wp_L \) represent the orientation of the world frame in the left camera frame and position of the left camera in the world frame respectively which are computed from elements of the state, and the terms \( Rp_R \) and \( Lp_R \) are from the known stereo extrinsics.

As mentioned earlier, the main difference in our filter compared to the original MSCKF [74] is that we use a stereo measurement model instead of the monocular one. In this measurement model, the location of the feature in the left image is computed similar to the original filter but to compute the location of the feature in the right image, we use the known stereo extrinsics to project the feature location from the left camera frame to the right camera frame. This enforces the constraint of the known stereo extrinsics and provides additional information to the filter [78]. More details about our implementation can be found in [99].

A comparison of our algorithm against some open-source tightly-coupled VIO algorithms, namely OKVIS\(^3\) [60], ROVIO\(^4\) [11] and VINS-Mono\(^5\) [81], on the EuRoC dataset is shown in Figure 4.3. From the figure, we can see that our method provides a good compromise between estimation accuracy and low computational requirement.

On our robot, we run the IMU at a rate of 200 Hz and the cameras are triggered once every five IMU samples leading to a frame rate of 40 Hz. The VIO takes these inputs and outputs odometry at the camera frame rate (40 Hz).

\(^2\)The development of the VIO system was primarily led by Ke Sun and done in collaboration with Bernd Pfrommer and Yash Mulgaonkar.

\(^3\)https://github.com/ethz-asl/okvis

\(^4\)https://github.com/ethz-asl/rovio

\(^5\)https://github.com/HKUST-Aerial-Robotics/VINS-Mono
Figure 4.3: A comparison of the accuracy and computational efficiency our VIO system with various open source packages on the openly available EuRoC dataset. Our method fails on the V2_03 dataset due to significantly different exposure times on the two cameras in some parts of the dataset.

**Exposure Control**

The fast feature detector used in our VIO system relies on the difference between pixel intensities in order to determine the location of the features. In situations when the scene is either too dark or bright for the current shutter time, the lack of contrast in the image results in insufficient number of detected features which leads to a degradation of the VIO performance. Therefore, it is desired that camera shutter time be automatically adjusted in order to maintain sufficient image contrast.

This problem is easily solved in monocular systems by just turning on the built-in auto-exposure algorithm that most cameras provide however in our case, with stereo cameras, we cannot let the two cameras run their own internal auto-exposure independently. This is because we use the KLT algorithm for feature matching across the stereo images, which assumes brightness constancy between the images. This means that the neighborhood of each feature should have the same brightness in both the images. Hence we want the exposure
changes to be synced for both the left and right cameras. In fact, the failure of our algorithm in one of the EuRoC datasets (V2_03) is caused by different exposures on the left and right cameras for some portions of the dataset.

In addition, we wanted to have control over the region of interest (ROI) used for determining the average image brightness instead of using the full image. This is because when flying with the cameras facing forward, the top part of the image usually contains the sky or the ceiling which has significantly different brightness compared to the bottom of the image where most of the good features are present. By using only the bottom 70% of the image for calculating the desired shutter time, we get better contrast in the region of the image where we expect good features.

We use the pixel values, which are in the range $0 - 255$, as a measure of brightness. To keep the average image brightness at a target value, $b_{targ}$, we employ the iterative controller from [61] where the exposure time for the next frame, $\Delta t_{k+1}$, is adjusted based on the current exposure time, $\Delta t_k$, and current brightness, $b_k$, as follows:

$$\Delta t_{k+1} = \Delta t_k \cdot \frac{b_{targ}}{b_k}$$

We found that setting the target brightness $b_{targ} = 70$ worked best for our system. In order to measure $b_k$, instead of averaging over the whole image, we found it sufficient to sample only one pixel in a $32 \times 32$ block. We also set an upper limit on the exposure time since we do not want it to grow too large and cause motion blur. Hence, if the maximum allowed exposure time is reached due to low lighting conditions, a controller equivalent to the one used for the exposure time is used to control the gain on the camera.

### 4.3.2 Sensor Fusion

We have multiple sensors on the platform, each providing partial information about the state of the robot. Moreover, we need high rate state estimates for the controller. The sensors on the robot provide output at different rates, for example, the IMU is running at 200 Hz,
Figure 4.4: Sequence of images showing how our auto-exposure algorithm synchronously changes the exposure of both the left and right cameras as the robot goes from a dimly lit indoor to a bright outdoor environment. The actual exposure times change from 10 ms in the first pair of images to 0.01 ms in the last.

the output of the VIO is at the camera frame rate (40 Hz) while the downward pointing distance sensor runs at 20 Hz. We need to merge these pieces of partial information into a single consistent estimate of the full state of the robot.

The typical method used for such sensor fusion tasks is some variant of the Kalman filter. The quadrotor is a nonlinear system due to its rotational degrees of freedom. This requires the use of either an Extended Kalman filter (EKF) or an Unscented Kalman filter (UKF). The UKF has the advantage of better handling the system nonlinearities with only a small increase in computation, so we chose the UKF for our system. Figure 4.5 shows the inputs and outputs of the UKF module running on the robot. The state vector used in the UKF is,

\[
\mathbf{x} = \begin{bmatrix} \mathbf{p}^T & \dot{\mathbf{p}}^T & \phi & \theta & \psi & b_\alpha^T & b_\omega^T \end{bmatrix}^T
\]

where \( \mathbf{p} \) is the world-frame position of the robot, \( \dot{\mathbf{p}} \) is the world-frame velocity, \( \phi \), \( \theta \) and \( \psi \)
are the roll, pitch and yaw respectively, \( b_a \) is the accelerometer bias while \( b_\omega \) is the gyroscope bias. We use the ZYX convention for representing the rotations in terms of the Euler angles \( \phi \), \( \theta \) and \( \psi \). The Euler angle representation was chosen for representing the orientation primarily because of its simplicity. The well-known problem of gimbal lock when using Euler angles is not an issue in this case since the desired and expected roll and pitch of the robot is always less than 90°.

The UKF consists of a prediction step which uses the IMU data as the input and multiple update steps, one for each of the other sensors. The update step is performed whenever the corresponding sensor measurement arrives. The prediction step is nonlinear since the accelerometer and gyroscope measurements are in the body frame while the position and velocity in the state are in the world frame, which requires the transformation of the measured quantities from body to world frame using the estimated orientation.

Given that the state at iteration \( k \), \( \mathbf{x}_k \) (dimension \( n \)), has mean \( \bar{\mathbf{x}}_k \) and covariance \( \mathbf{P}_k \), we augment it with the process noise (dimension \( p \)) having mean \( \bar{\mathbf{v}}_k \) and covariance \( \mathbf{Q}_k \), creating the augmented state \( \mathbf{x}^a_k \) and covariance matrix \( \mathbf{P}^a_k \):

\[
\bar{\mathbf{x}}^a_k = \begin{bmatrix} \bar{\mathbf{x}}_k \\ \bar{\mathbf{v}}_k \end{bmatrix}, \quad \mathbf{P}^a_k = \begin{bmatrix} \mathbf{P}_k & 0 \\ 0 & \mathbf{Q}_k \end{bmatrix}
\]

Then, we generate a set of sigma points by applying the Unscented transform [47] to the
augmented state,

\[ X_0^a(k) = \bar{x}_k^0 \]

\[ X_i^a(k) = \bar{x}_k^0 + \sqrt{(L + \lambda) P_k^a} \quad i = 1, \ldots, L \]

\[ X_i^a(k) = \bar{x}_k^0 - \sqrt{(L + \lambda) P_k^a} \quad i = L + 1, \ldots, 2L \]

where \( L = n + p \) is the dimension of the augmented state and \( \lambda \) is a scaling parameter [114].

These sigma points are then propagated through the process model with the accelerometer and gyroscope measurements as input.

\[ X_i^x(k + 1 \mid k) = f \left( X_i^x(k), u(k), X_i^v(k) \right) \]

where \( X_i^x \) is the state part of the augmented state while \( X_i^v \) is the process noise part. The process model, \( f (x_k, u_k, v_k) \), for our system is given by

\[ u_k = \begin{bmatrix} a_{meas}^T \omega_{meas}^T \end{bmatrix} v_k = \begin{bmatrix} v_a^T \ v_\omega^T \ v_{ba}^T \ v_{b\omega}^T \end{bmatrix} \]

\[ a = a_{meas} - b_a + v_a \]

\[ \omega = \omega_{meas} - b_\omega + v_\omega \]

\[ p_{k+1} = p_k + \dot{p}_k \ dt \]

\[ \dot{p}_{k+1} = \dot{p}_k + (R_k a - g) \ dt \]

\[ R_{k+1} = R_k \left( I_3 + [\omega]_x \ dt \right) \]

\[ b_{ak+1} = b_{ak} + v_{ba} \ dt \]

\[ b_{\omega k+1} = b_{\omega k} + v_{b\omega} \ dt \]

where \( R_k = R (\phi_k, \theta_k, \psi_k) \) is the rotation matrix formed by using the ZYX convention for the Euler angles while \( v_a, v_\omega, v_{ba} \) and \( v_{b\omega} \) are the individual process noise terms.

From the transformed set of sigma points, \( X_i^x(k + 1 \mid k) \), we can calculate the predicted mean
and covariance,

\[
\bar{x}_{k+1|k} = \sum_{i=0}^{2L} w^m_i \mathbf{X}^m_i (k + 1 | k)
\]

\[
P_{k+1|k} = \sum_{i=0}^{2L} w^c_i \left[ \mathbf{X}^c_i (k + 1 | k) - \bar{x}_{k+1|k} \right] \left[ \mathbf{X}^c_i (k + 1 | k) - \bar{x}_{k+1|k} \right]^T
\]

where \(w^m_i\) and \(w^c_i\) are scalar weights [114].

Whenever a new sensor measurement, \(y_{k+1}\), arrives, we run the update step of the filter. First we generate a new set of sigma points in the same way as done during the prediction step, Equation 4.1, with the augmented state and covariance given by,

\[
\bar{x}^a_{k+1|k} = \begin{bmatrix} \bar{x}_{k+1|k} \\ \bar{n}_k \end{bmatrix}, \quad P^a_{k+1|k} = \begin{bmatrix} P_{k+1|k} & 0 \\ 0 & R_k \end{bmatrix}
\]

where \(\bar{n}_k\) is the mean of the measurement noise and \(R_k\) is the covariance. The generated sigma points are then used to generate the predicted measurement using the measurement function \(h (x, n)\),

\[
\mathbf{Y}_i (k + 1 | k) = h \left( \mathbf{X}^m_i (k + 1 | k), \mathbf{X}^n_i (k + 1 | k) \right)
\]

\[
\bar{y}_{k+1|k} = \sum_{i=0}^{2L} w^m_i \mathbf{Y}_i (k + 1 | k)
\]

\[
P_{yy} = \sum_{i=0}^{2L} w^c_i \left[ \mathbf{Y}_i (k + 1 | k) - \bar{y}_{k+1|k} \right] \left[ \mathbf{Y}_i (k + 1 | k) - \bar{y}_{k+1|k} \right]^T
\]
And finally the state is updated as follows,

\[
P_{xy} = \sum_{i=0}^{2L} w_i \left[ \mathbf{X}_i^x(k+1|k) - \bar{x}_{k+1|k} \right] \left[ \mathbf{Y}_i(k+1|k) - \bar{y}_{k+1|k} \right]^T \tag{4.4}
\]

\[
K = P_{xy} P_{yy}^{-1}
\]

\[
\bar{x}_{k+1} = \bar{x}_k + K \left( y_{k+1} - \bar{y}_{k+1|k} \right)
\]

\[
P_{k+1} = P_{k+1|k} - KP_{yy} K^T
\]

Note that for each sensor input to the UKF except the IMU, which is used for the prediction step, there is a separate measurement function, \( h(x, n) \), and the full update step is performed, with the corresponding measurement function, when an input is received from any of those sensors.

Handling jumps in downward lidar

In order to take care of jumps in the height sensor when going over obstacles such as boxes, the UKF maintains an internal floor height parameter. If there is a jump in the height sensor output compared to the expected value, the UKF assumes that the floor level has changed and uses the new floor level as the reference level for the height sensor. In this way, jumps in the height sensor output are properly taken care of. The limitation of this approach is that when the floor level changes slowly, the robot only maintains the desired height relative to the floor level and moves up and down as the floor level rises and falls.

Improving the Pixhawk orientation estimate

The attitude filter running on the Pixhawk is a simple complementary filter which can take an external reference orientation as an input. This allows us to provide the estimate from the UKF to the Pixhawk in order to improve the orientation estimate on the Pixhawk. This is important for good control performance since the orientation controller running on the Pixhawk uses the Pixhawk’s estimate of the orientation while our control commands are calculated using the UKF estimates. Without an external reference being sent to the
Pixhawk, the orientation estimates on the Pixhawk can be different from the UKF which would lead to an incorrect interpretation of the control commands.

4.4 Control

The controller used for the robot has the cascade structure, as shown in Figure 4.2, which is has become standard for MAVs. In this structure, we have an inner loop controlling the orientation and angular velocities of the robot while an outer loop controls the position and linear velocities. In our case, the inner loop runs at a high rate (400 Hz) on the Pixhawk autopilot while the outer loop runs at a slightly slower rate (200 Hz) on the Intel NUC computer.

At every time instance, the outer loop position controller receives a desired state, which consists of a desired position, velocity, acceleration and jerk, from the planner and using the estimated state from the UKF, computes a desired force, orientation and angular velocities which are sent to the orientation controller. The inner loop orientation controller receives these and computes the thrust and moments required to achieve the desired force and orientation. These are then converted into individual motor speeds that are sent to the respective motor controllers.

The controller formulation we use is based on the controller developed in [58] with some simplifications. The thrust command of the position controller is calculated as,

\[
e_{pos} = \hat{p} - p_{des}, \quad e_{vel} = \hat{\dot{p}} - \dot{p}_{des}
\]

\[
f = m \left( -k_{pos}e_{pos} - k_{vel}e_{vel} + ge_3 + \ddot{p}_{des} \right)
\]

\[
\text{Thrust} = f \cdot \hat{R}e_3
\]  

(4.5)

where \(\hat{p}\) is the estimated position of the robot in the world frame, \(\hat{\dot{p}}\) is the estimated velocity of the robot in the world frame, terms with the \(\text{des}\) subscript are the desired quantities, \(m\) is the mass of the robot, \(k_{pos}\) and \(k_{vel}\) are controller gains, \(e_3 = [0 \ 0 \ 1]^T\), and \(\hat{R}\) is the rotation matrix which converts vectors from body frame to world frame calculated using the
estimated roll, pitch and yaw.

The desired attitude is calculated as,

\[
b_{2,\text{des}} = \begin{bmatrix} -\sin \psi_{\text{des}}, & \cos \psi_{\text{des}}, & 0 \end{bmatrix}^T
\]

\[
b_3 = \frac{f}{\|f\|}, \quad b_1 = \frac{b_{2,\text{des}} \times b_3}{\|b_{2,\text{des}} \times b_3\|}, \quad b_2 = b_3 \times b_1
\]

\[
R_{\text{des}} = \begin{bmatrix} b_1, & b_2, & b_3 \end{bmatrix}
\]

(4.6)

\[
b_{2,\text{des}} = \begin{bmatrix} -\cos \psi_{\text{des}} \dot{\psi}_{\text{des}}, & -\sin \psi_{\text{des}} \dot{\psi}_{\text{des}}, & 0 \end{bmatrix}^T
\]

\[
b_3 = b_3 \times \frac{\dot{f}}{\|f\|} \times b_3, \quad \dot{b}_1 = b_1 \times \frac{b_{2,\text{des}} \times b_3 + b_{2,\text{des}} \times \dot{b}_3 \times b_1}{\|b_{2,\text{des}} \times b_3\|}, \quad \dot{b}_2 = \dot{b}_3 \times b_1 + b_3 \times b_1
\]

\[
[\Omega_{\text{des}}]_\times = R_{\text{des}}^T \dot{R}_{\text{des}}
\]

(4.7)

where \(\psi_{\text{des}}\) and \(\dot{\psi}_{\text{des}}\) are the desired yaw angle and yaw rate respectively.

Note that here we have to define \(b_{2,\text{des}}\) based on the yaw instead of defining \(b_{1,\text{des}}\) as done in [69] due to the different Euler angle convention, we use the ZYX convention while they used ZXY.

The thrust and attitude commands, from Equation 4.5, Equation 4.6 and Equation 4.7, are then sent to the Pixhawk autopilot through mavros\(^6\). The attitude controller running on the Pixhawk takes these commands and converts them to commanded motor speeds. First, using the \(R_{\text{des}}\) and the estimate of the current orientation, \(\hat{R}\), we calculate the desired moments as follows,

\[
[e_R]_\times = \frac{1}{2} \left( R_{\text{des}}^T \hat{R} - \hat{R}^T R_{\text{des}} \right), \quad e_\Omega = \Omega - \hat{R}^T R_{\text{des}} \Omega_{\text{des}}
\]

\[
M = -k_R e_R - k_\Omega e_\Omega
\]

where \(\Omega\) is the current angular velocity of the robot in the body frame, and \(k_R\) and \(k_\Omega\) are

\(^6\)https://github.com/mavlink/mavros
controller gains.

Then, from the desired thrust and moments, we can calculate the thrust required from each propeller which allows us to compute the desired motor speed as shown in [70].

4.4.1 Drag Compensation

The controller described above has good tracking performance when following aggressive trajectories but during fast flight, the aerodynamic effects become significant and cause deviations from the desired trajectory. In [100], we demonstrated the use of a simple lumped parameter model for drag and proposed a controller that compensates for its effect. We briefly describe the drag compensation controller here.

For the range of speeds that we are interested in, most sources of drag and drag-like effects such as blade flapping may be approximated as a linear function of the component of the velocity in the plane of the propellers [2]. Using this model, the drag force on the robot expressed in the world frame is given by

\[
\mathbf{f}_{\text{drag}} = -k_d \mathbf{RPR}^T \mathbf{v}
\]

where \( \mathbf{v} \) is the velocity of the quadrotor expressed in the world frame, \( k_d \) is the drag constant, \( \mathbf{R} \) is the orientation of the quadrotor expressed as a rotation matrix which takes points from the body frame to the world frame, and \( \mathbf{P} \) is the projection matrix:

\[
\mathbf{P} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}.
\]

Note that in contrast to [100], we do not include the motor speeds in the drag model since the motor controllers used on the current platform do not provide feedback about the motor speeds. Instead, we use an approximation that the motor speeds during most flights are close to their nominal (hover) values and lump them inside the drag constant.
Figure 4.6: The forces on the robot when it is moving towards the right with a velocity \( v \). Note that the modelled drag is in the x-y plane in the robot body frame, which may not be exactly opposite to the direction of velocity.

The drag force cannot be directly compensated by changing the magnitude of the thrust, since the drag is in the x-y plane in the body frame while thrust is along the body z axis and hence orthogonal to the drag force (see Figure 4.6). The compensation is achieved by changing the commanded orientation. Instead of the conventional approach of setting the desired body z-axis direction to be

\[
b_{3, \text{des}} = \frac{-k_x e_x - k_v e_v + a_{\text{des}} + g}{\| -k_x e_x - k_v e_v + a_{\text{des}} + g \|},
\]

(no drag)

we change it to

\[
b_{3, \text{des}} = \frac{-k_x e_x - k_v e_v + a_{\text{des}} + g + \frac{k_d}{m} v}{\| -k_x e_x - k_v e_v + a_{\text{des}} + g + \frac{k_d}{m} v \|},
\]

(with drag)

where \( k_x \) and \( k_v \) are positive gains, \( e_x \) and \( e_v \) are errors in position and velocity tracking, \( a_{\text{des}} \) is the feed-forward desired acceleration, \( g \) is the acceleration due to gravity, and \( m \) is the mass of the robot. Once \( b_{3, \text{des}} \) is computed, the desired orientation and angular velocity can be calculated using the desired yaw and yaw rate, similar to Equation 4.6 and Equation 4.7.

4.5 Mapping and Planning

As mentioned earlier, we use a 2D lidar mounted on a servo such that we can generate a 3D pointcloud by pitching the lidar up and down. One of the main concerns for the mapping solution for high speed flight is that it should be quick to update so that we can incorporate new obstacles into it as soon as possible. Updating the map and planning using a large 3D...
global map are both computationally expensive and in addition, with noise and estimation drift, the global map can be erroneous. Hence we utilize a local mapping technique where we generate a 3D uniform voxel grid map centered at current robot location. Since the local map only records the recent sensor measurements with respect to current robot location, the accumulated error in mapping due to estimation drift is small. The problem with only using a local map is that the robot can get stuck in places such as dead-ends due to the map not containing enough information for the planner to choose an alternative path. To avoid this problem, we also generate a coarser resolution 2D map in global frame by taking a slice of the local 3D map while taking care of removing obstacles that the robot can go over or under and performing a ray-trace of all the occupied voxels in this slice. We call this map the “global information map” since it contains two pieces of information: one is the known and unknown spaces so that we know which part has been explored, the other is the location of walls detected from local point cloud that the robot cannot fly over. There is no loop-closure or global scan-matching involved in constructing this map since the map is solely used for navigation where only nearby obstacles are relevant.

We use the search-based motion planning framework described in [62] for generating dynamically feasible, safe and optimal trajectories for the robot. The planner needs to periodically replan so as to avoid collisions with new obstacles which come into view as the robot moves and during high-speed flights, having a fast replan rate is important to quickly react to new obstacles. In our implementation of the search-based planner, we use acceleration as the input in the motion primitive based planner to keep our planner execution time low, even though the quadrotor is a fourth order system and has snap as the input [69]. An example of the constant acceleration motion primitives and the discretization of the positions reached as the primitives are expanded is shown in Figure 4.7. As the graph search progresses, the motion primitive based planner incrementally extends the trajectory by adding these fixed-duration segments till it reaches the goal. In this process, the planner also finds the optimal total time for the trajectory. This means that we do not require a prior time allocation for the trajectory as compared to the optimization based methods which require a given time allocation for
Figure 4.7: The set of positions reached from the origin after two steps of expansion using nine constant acceleration primitives at each step. Yellow lines represent the first expansion while the green curves represent the second. We can see the discretization in the set of positions reached by this set of primitives as it expands. Note that the primitives do not necessarily end at zero velocity, so the state of the robot at most of these positions has non-zero velocity.

Each segment. This prevents the generation of bad trajectories due to a bad time allocation which is a common problem with the optimization based approaches.

One of the main advantages of this planning method is that it can take into account not just the position but also the higher derivatives in the starting state when planning (see Figure 4.8). This leads to much better trajectories specially in cases when the robot is traveling at a high speed and needs to suddenly change the direction of motion, for example, due to new obstacles coming into the sensor field of view or a change in the position of the goal. In such cases, optimization based planners can sometimes fail to generate a feasible trajectory and the robot needs to execute a blind emergency stop procedure which can led to the robot crashing into an obstacle near the robot. The search-based planner avoids such failure cases.

Due to our use of constant acceleration motion primitives, the trajectory from the planner
Figure 4.8: Effect of initial state on the planned trajectory. The green dot is the start position and the red dot is the goal. (a) shows the output of a path based planner which does not take into account the dynamics of the robot, (b) shows the output of the motion primitives based planner where the initial state is at rest, and (c) shows the trajectory from the motion primitives based planner when the robot has an initial velocity in the upward direction.

Figure 4.9: The effect of the refinement on the trajectory generated from the graph search. The original trajectory from the search based planner with constant acceleration motion primitives is shown in orange while the refined trajectory is shown in blue. It can be seen that refined trajectory is smoother but the difference in positions between the original and the refined trajectories is very small.
has jumps in the desired acceleration as we go from one motion primitive segment to the next along the trajectory and leads to jerky motion if directly used for control. In order to provide smooth inputs to the controller, we run a trajectory refinement step where we use the intermediate waypoint states and the corresponding time allocation from the generated acceleration input trajectory and fit a higher order piece-wise polynomial trajectory to it while neglecting the dynamics and obstacle constraints. This step weakens the safety and feasibility guarantees from the planner, since the refined trajectory can deviate from the originally planned trajectory, but empirically we found that the actual deviation is very small and does not lead to collisions with the obstacles. Figure 4.9 shows an example of this refinement step. Using this separate refinement step allows us to run the planner at the rate of 3 Hz while still providing a smooth reference to the controller.

4.6 Experimental Results

The quadrotor navigation system described in this chapter has been tested extensively in the lab environment as well as in multiple real-world environments. The system has been used on our entry for the DARPA Fast Lightweight Autonomy (FLA) program and was able to successfully navigate multiple obstacle courses that were set up. The rules of the FLA program do not allow any human interaction after the robot is airborne, so all the runs described in this section were fully autonomous.

And finally to evaluate the entire system, we performed experiments in a forest like environment as well as a mixed outdoor + indoor setting where the robot was given an approximate position of a red barrel relative to the start position and the goal was to fly to the barrel and return to the start position fully autonomously.

4.6.1 High Speed Flights

While the planner can be tested in simulation, to evaluate the estimation and control modules we need to test with the real robot. We used publicly available datasets to check the performance of our estimation algorithm but we wanted to test the robustness of our system during fast flights for which there are no available datasets. We also wanted to compare the
improvement in trajectory tracking that the drag compensation in the controller provides during these high speed flights.

For testing the robustness and performance of our estimation and control modules, we ran experiments in a large open field where the robot was commanded to autonomously follow a 300 m straight line trajectory at increasing speeds. During these trajectories, the speed ramped up to the maximum value, stayed constant for most of the flight and ramped down to zero at the end. The highest speed that we could reliably achieve with our platform was around 18 m/s. Above that speed the robot started losing height while following the trajectory due to the physical limit of maximum thrust. For these experiments, we mounted a GPS unit on the robot which was only used for data logging in order to have a ground truth estimate.

**Figure 4.10** shows the commanded, and estimated velocity along with the velocity and position tracking errors for the straight line flight where the maximum commanded velocity was 17.5 m/s. During the high speed flight, the robot is able to achieve the desired velocity but the force due to drag causes the robot position to lag behind the commanded position. This motivates the use of drag compensation in the controller during such high speed flights.

To test the effect of drag compensation, we commanded the robot to fly along straight line trajectories at a maximum speed of 15 m/s, once with the drag compensation turned
Figure 4.10: Plots of the velocity and tracking errors during a high speed flight without drag compensation where the robot was commanded to follow a 300 m straight line trajectory with a maximum speed of 17.5 m/s.

off and then again with the drag compensation turned on. With the drag compensation turned off, the tracking error in the position is around 3 m which is quite large given that the robot diameter is 0.76 m. The tracking error drops down to less than 1 m when we enable the drag compensation in the controller showing a significant improvement in the high speed performance of the system. Note that here we only show the results from the drag compensation at one speed but in our tests we have seen consistent improvement in the tracking performance at speeds ranging from 5–15 m/s.

4.6.2 Outdoor + Indoor

In order to test the full system including the mapping and planning, we ran an experiment where the robot started outdoors while the goal location was specified to be inside a warehouse. Figure 4.12 and Figure 4.13 show an overview of the experiment. The robot starts at the position marked by the green circle on the left in the figure while the goal location is marked by the red circle inside the building on the right. In order to reach the goal, the robot had
to first navigate through a sparse, forest like environment. After traversing the forest, the robot moves towards the building and needs to find the open door on the opposite (right) side of the building. Only one of the doors of the warehouse was kept open, so the robot had to explore around the building to find the open door. Eventually the robot makes its way to the right side of the building, enters through the open door and gets to the goal location. The maximum speed of the robot was set to 5 m/s for safety reasons and the robot was able to achieve that speed during the segments with no obstacles nearby.
Figure 4.12: A series of images from the experiment showing the progress of the robot starting from the forest environment and ending inside the building.

Figure 4.13: An overhead map showing the full run for the indoor + outdoor test. The starting position of the robot is marked with a green circle while the goal position is marked in red. The path followed by the robot is shown in blue.
Chapter 5

Multi-Robot Autonomy – A Centralized Paradigm with External Infrastructure

5.1 System Architecture

The goal of QUADCloud is to monitor an area that is the size of a few city blocks, approximately a total area of $400\,\text{m} \times 400\,\text{m}$, while being responsive to the operator’s commands and dynamic information collected by one or more robots. We want the response times to these requests of surveillance to be less than 30s. This requires the robots to have maximum speeds of around $10\,\text{m/s}$, and maximum acceleration of $3\,\text{m/s}^2$. We want the size of the robots to be as small as possible for ease of handling and deployment (imagine deploying a team from the back of a pickup truck), and have enough payload capacity to carry all the components needed for autonomy and surveillance. Since the system runs outdoors, the robots also need to have enough thrust to be able to cope with moderate winds.

A key requirement for QUADCloud is that a single operator must be able to deploy, control
and monitor the team of robots easily. This requires sufficient autonomy on each robot to able to handle the navigation task from trajectory generation to position stabilization on-board. Further, on each robot, we need downward-facing cameras to provide imagery, and the computational resources to allow processing of images at around 5 Hz from the on-board camera. Each robot must have the requisite on-board intelligence to look for salient information. In this paper, we do assume that the targets of interest are relatively simple and can be easily identified using regular cameras at heights of around 10–20 m meters. However, the algorithms must allow for false positives and a probabilistic representation of the belief state of the environment. Finally, in order to send data back from all parts of the monitored area and to respond to commands issued by the operator, each robot must be able to communicate with the base station within a 400 m distance.

Thus the architecture must incorporate some elements of per-robot control, estimation and target detection and localization while allowing for a centralized, cloud computing model for command and control by the user and task planning. Further, we want a framework in which the user is agnostic to the number of robots, their identities and what exactly their individual states are. This attribute of anonymity increases the robustness of the system to failures and decreases the overhead on the human user.

The simplest architecture with these attributes is shown in Figure 5.1. Robots are able to
localize and control their motions to destinations, freeing the operator to work with high level task specifications such as target destinations or areas for surveillance. A centralized goal assignment and planning module decides which robot responds to what request and when. Individual robots independently decide how to follow these requests. Each robot periodically sends back its position estimates and images from the on-board camera to the base station. This information is presented to the operator on a simple user interface and through the user interface, the operator is also able to command goal positions which are sent to the planner.

5.2 Robot Hardware

5.2.1 Prototype 1

The initial robots used for this project, shown in Figure 5.2, are quadrotors designed and developed by KMel Robotics. These quadrotors have a tip-to-tip diameter of about 0.54 m and weigh around 0.95 kg in the configuration used for this project. They are equipped with an ARM Cortex-M3 processor, 3-axis accelerometer, gyroscope, magnetometer and a pressure sensor while a u-blox NEO-6T GPS module was added in order to get position estimates. A control loop running at 500 Hz on the ARM processor stabilizes the attitude of the quadrotor. The communication with the on-board processor in order to receive sensor outputs and send thrust and attitude commands is done via a UART port.

All the high-level computations on the robot are performed on an Odroid-U2 quad-core ARM single board computer. This compact board has four Cortex-A9 cores each running at up to 1.7 GHz allowing us to have a powerful processor in a small form-factor. The Odroid-U2 comes with a big passive heat-sink which we replaced with a small active heat-sink cutting its weight from 130 g to around 50 g. Each quadrotor is also equipped with a Matrix Vision mvBlueFOX camera in order to capture images for target detection and surveillance. We wanted each robot to send images to the base station for surveillance purposes, thus we had a requirement of long-range communications with sufficient bandwidth. Since the Odroid-U2 does not have any built-in wifi, but has an ethernet port, we decided to use a Ubiquity Networks Bullet M2 which provides the required bandwidth with a range of more than
Figure 5.2: The KMel kQuad500 quadrotor equipped with a u-blox NEO-6T GPS module, a Matrix Vision mvBlueFOX camera, a Ubiquity Networks Bullet M2 and a Odroid-U2 quad-core ARM single board computer.

350 m. This allows us to stream compressed images at more than 30 fps from a single robot or around 5 fps from each robot when we have a team of 8 quadrotors sending images back. The Bullet M2 comes with a large and heavy antenna connector which we replaced with a smaller one, and we also removed the plastic shell for weight saving, reducing its weight from 180 g to 50 g. The robots use a 3-cell 2.2 A h LiPo battery which gives a flight time of around 8–10 min with the current configuration of the robot.

5.2.2 Prototype 2

After testing and validating the system with the first set of robots, we made some changes in order to make them better suited to the task. The first thing we wanted to do was to reduce the size of the robots in order to make them safer and easier to use. In order to have sufficient thrust and still make the size smaller, the hex configuration was chosen. This configuration also provides potential robustness to single rotor failures which makes the system safer. These new robots, shown in Figure 5.3, have a tip-to-tip diameter of 0.34 m and weight 0.4 kg (less than half of the first prototypes). The other components are kept similar to the previous robots with minor changes such as removal of the Ethernet and USB sockets on the Odroid in order to save weight and change from the u-blox 6 to a u-blox M8.
GPS module in order to get faster GPS update rates to improve localization accuracy.

5.3 Software

The software system on the robot consists of three main components: estimation, control and image processing. The image processing is simple and is composed of detecting specified features (that characterize the targets) in the image and image compression for transmitting the images from the camera to the base station. The estimation and control subsystems are described in more detail in the next chapter.

5.4 Estimation and Control

In order to fly, the robot needs to know its 6-DoF pose and velocity, and control the motor speeds in order to keep itself stable and follow the given commands. An overview of the estimation and control systems running on each robot is shown in Figure 5.4.

5.4.1 State estimation

The state estimation on the robots uses the sensor readings from the accelerometer and gyroscope in the IMU, estimated yaw from the magnetometer, and estimated height from
the pressure sensor. These estimates along with the GPS module output are fused on the Odroid-U2. First, the GPS latitude, longitude and height are transformed to a local cartesian frame using GeographicLib [51] with a pre-determined location as the origin. We ignore the height from the GPS measurement since it has a large drift and instead rely on the barometer for getting the height. It should be noted that in addition to the position output, the GPS module also outputs velocity estimates which allow us to get smooth position estimates. The output of this node along with the other sensor outputs are then fed to a Unscented Kalman Filter (UKF) in order to generate full pose estimates. The UKF state is,

\[
\mathbf{x} = \begin{bmatrix}
\mathbf{p} & \dot{\mathbf{p}} & \psi & \theta & \phi & \mathbf{a}_b
\end{bmatrix}^T
\]

where \(\mathbf{p}\) is the world-frame position of the robot, \(\dot{\mathbf{p}}\) is the world-frame velocity, \(\psi\), \(\theta\) and \(\phi\) are the yaw, pitch and roll respectively and \(\mathbf{a}_b\) is the accelerometer bias along the three axes.

For the UKF, the accelerometer and gyroscope readings are the process inputs while the GPS position and velocity, height from the barometer and magnetometer heading are the measurements. The process model is non-linear since the accelerometer and gyroscope readings are in the body-frame while the state has world-frame position and velocity which requires a transformation of the sensor data from body to world frame. On the other hand, GPS, barometer and magnetometer measurements are in the world-frame, hence the measurement model is linear allowing us to use the measurement update equations of a
standard Kalman filter.

The equations for the UKF and the process model step are similar to the one described in subsection 4.3.2. The only difference is that we do not have the gyroscope bias in the state here, since we assume that it been calibrated to remove the bias.

As mentioned earlier, the update step is performed using the measurements from the GPS, barometer and magnetometer. We convert the GPS latitude, longitude and height outputs into local cartesian coordinates with the origin set at a fixed GPS location close to the robots.

We noticed that the altitude from the GPS measurements drifts by more than 10 m even when the robot is hovering at a single place, thus we do not use the altitude part of the local cartesian estimate. Instead, the altitude relative to the starting location is computed from the barometer output. The magnetometer output is transformed into a yaw estimate relative to north by using the known magnetic declination at the robot’s GPS location. We combine these individual measurements into a single measurement to get

\[
y = \begin{bmatrix} p^T & \dot{p}^T & \psi \end{bmatrix}.
\]

This measurement model is linear in the state, hence we can use the update step of a regular Kalman Filter to get the new state estimate, \( \hat{x}_{k+1} \), and covariance, \( P_{k+1} \),

\[
\begin{align*}
\dot{\hat{y}}_{k+1} &= H \hat{x}_{k+1|k} \\
\hat{y}_{k+1} &= y_{k+1} - \hat{y}_{k+1} \\
S_{k+1} &= HP_{k+1|k}H^T + R_{k+1} \\
K_{k+1} &= P_{k+1|k}H^T S_{k+1}^{-1} \\
\hat{x}_{k+1} &= \hat{x}_{k+1|k} + K_{k+1} \hat{y}_{k+1} \\
P_{k+1} &= (I - K_{k+1}H) P_{k+1|k}
\end{align*}
\]

where \( H \) is the observation matrix and \( R_{k+1} \) is the covariance of the measurement noise at
iteration $k + 1$.

Thus, by using the Unscented Transform only for the non-linear part (prediction step) and using the linear update equations for the measurement step, we are able to reduce the computations required for pose estimation.

5.4.2 Control

Our control architecture has the common cascade structure of backstepping controllers, like in [15, 70], with the attitude controller as the inner loop and the position controller as the outer loop around it. As mentioned earlier, the attitude controller runs at a very high rate on the on-board processor stabilizing the orientation of the robot, allowing us to run the position controller at a much slower rate on the Odroid-U2. A non-linear controller running at 50Hz takes position commands sent by a trajectory generator and, using the position estimates, converts them into thrust and attitude commands which are sent to the attitude controller running on the on-board processor.

The thrust command of the position controller is calculated as,

$$e_{pos} = \hat{p} - p_{des}, \quad e_{vel} = \hat{p} - \dot{p}_{des}$$

$$f = m \left(-k_{pos}e_{pos} - k_{vel}e_{vel} + g e_3 + \ddot{p}_{des}\right)$$

$$\text{Thrust} = f \cdot \hat{R}e_3$$

and the desired attitude is calculated as,

$$b_3 = \frac{f}{\|f\|}, \quad b_2 = b_3 \times \begin{bmatrix} \cos \psi_{des}, & \sin \psi_{des}, & 0 \end{bmatrix}^T, \quad b_1 = b_2 \times b_3$$

$$R_{des} = \begin{bmatrix} b_1, & b_2, & b_3 \end{bmatrix}$$

where $e_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ and $\hat{R}$ is the rotation matrix which converts vectors from body frame to world frame calculated using the estimated roll, pitch and yaw.
The attitude controller running on the robot takes the thrust and attitude commands and converts them to commanded motor speeds. It is similar to the controller developed in [58] with some simplifications. First, using the $R_{des}$ and the on-board estimate of the current orientation, $R$, it calculates the desired moments as follows,

$$[e_R]_x = \frac{1}{2} \left( R_{des}^T R - R_{des}^T R T_{des} \right), \quad e_\Omega = \Omega$$

$$M = -k_R e_R - k_\Omega e_\Omega$$

Here, we have set the desired angular velocity to be zero since the actual desired velocities are small, hence the angular velocity error is just the current angular velocity. From the desired thrust and moments, we can calculate the thrust required from each propeller which allows us to compute the desired motor speed.

### 5.5 Planning

To safely navigate the team of robots to goal locations, a motion planning algorithm is required that computes collision-free trajectories and respects the dynamics of the robots. It is well known that extending single robot motion planners to plan trajectories for a team of robots implies exponential computational complexity [24]. One attempt to solve this problem of computational intractability is to use a two step algorithm that decouples the path from the time parameterization of the trajectories as described in [57]. These decoupled approaches first plan motions for each individual robot while disregarding collisions with other robots. The second step is to specify the time parameterization the robot follows its path. Unfortunately, these approaches are not complete and cannot guarantee they will find a solution if one exists.

Fortunately, our team of robots are identical and we can exploit this interchangeability to generate collision-free time parameterized trajectories in a computationally tractable manner using the Goal Assignment and Planning (GAP) algorithm described in [104].

The GAP algorithm is a decoupled motion planner that maintains completeness by leveraging
the interchangeability of the robots. This algorithm begins by finding the cost associated with planning trajectories from the initial state of each robot to every goal location. We use Dijkstra’s algorithm to quickly find these \( N^2 \) motion plans where robot \( i \) has cost \( C_{ij} \) to travel to goal \( j \). The next step is the assignment of goals to robots where each robot is assigned to one goal. This assignment can be represented by a permutation matrix \( \phi \), where \( \phi_{ij} = 1 \) if and only if robot \( i \) is assigned to goal \( j \). The Hungarian Algorithm is used to find the assignment which minimizes the maximum cost:

\[
\minimize_{\phi} \sum_{i=1,2,...,N} \sum_{j=1,2,...,N} (\phi_{ij}C_{ij})^p
\]

where \( p \) is a large constant. In practice, \( p = 50 \) is used. Then, robots are prioritized using simple geometric considerations that are fully detailed in [104]. Finally, robots are assigned their full time parameterization to construct trajectories that guarantee collision avoidance.

For a team of 6 robots, these plans are generated in under 0.1 seconds. Additional details of this algorithm including boundary condition requirements for completeness are presented in [104].

5.6 Communication and Supervision

Communication

The base station communicates with each of the robots via wifi through the Bullet M2 high-power long-range wifi modules. We want each robot to send back position estimates at 50 Hz and image data at 5 Hz. The bandwidth requirement is dominated by the transmission of images from the multiple robots to the base station. The camera on each robot has a resolution of 1280 \( \times \) 960 which leads to a raw gray-scale image size of approximately 1.2 MB, so for raw image transmission at 5 Hz we require a bandwidth of about 48 Mbps. The Bullet M2 claims a maximum bandwidth of around 65 Mbps, but in real-world testing, we got a data rate of about 50 Mbps. Thus it is not possible to send raw images back from each of the robots at the desired rate. To reduce the bandwidth requirement, we decided
to jpeg compress the images before sending. This brings down the size of the images from 1.2 MB to about 130 kB allowing us to stream images at 5 Hz from up to 10 robots. If we want to add more robots, we would need to decrease the frame rate of the image data being sent back from the robots. A frame rate of 2 Hz is sufficient for surveillance purposes and would allow us to scale to around 20 robots.

User Interface

Since all the computations for autonomy are being done on the robots themselves, the operator does not need a very powerful base station to control the team; the base station can just be a small laptop. As mentioned earlier, the robots send their position estimates to the base station. This information is presented to the operator in the form of markers on an overhead schematic map of the area. In addition to monitoring the system, the user is able to send goal positions to the system without needing to specify which robot is assigned which goal. Using the algorithm described in section 5.5, the system assigns the goals to the robots in order to minimize the maximum travel time and plans trajectories for each of them. This reduces the cognitive burden on the operator by allowing the operator to focus on the high-level tasks.

5.7 Aggregation of visual imagery

The base station receives the images from the robots as well as their pose estimates. Using these poses, we can correct the perspective distortion of the image and project them onto the ground plane. This allows us to create an overhead map of the environment using the team of these robots. An example of this is shown in Figure 5.5 where images from three robots are being used to construct a mosaic of the area.

5.8 3D Reconstruction

In addition to the 2D mosaic, we can use the stream of images being sent back from the robots to create a 3D map of the area. This has been made possible through new developments in visual odometry and multi-view stereo techniques. In recent years there have been significant
advances in monocular visual odometry algorithms with feature based methods such as [53] and direct methods like [22]. Recently, [27] introduced a semi-direct visual odometry algorithm that combines the advantages of feature-based methods (parallel tracking and mapping, keyframe selection) and direct methods (speed and accuracy) in order to have a fast and robust visual odometry algorithm. Once we have the relative transforms between the images from the visual odometry algorithm, for the 3D reconstruction, we need a fast, robust and efficient method to construct and update the map. We use the technique described in [113] which is a per-pixel, probabilistic depth estimation scheme that updates the depth posterior of each point with every new frame. The pipeline for the 3D reconstruction is shown in Figure 5.6.

A sample 3D reconstruction from one of the robots is shown in Figure 5.7. Currently we are able to construct individual 3D maps for each robot, future work would involve merging these maps from each of the robots into one to have a combined 3D reconstruction of the
Figure 5.6: The 3D reconstruction pipeline running on the base station for each robot.

Figure 5.7: The 3D reconstruction of a 20 m × 20 m outdoor scene using the images from one robot. The colors represent the height, with increasing height going from red to green.

5.9 Experimental Results

5.9.1 Estimation and control benchmarking

The estimation and control system of the robot need to be able to estimate the true state of the robot and control to the desired state provided by the planner. Failure in any one of these tasks would compromise the safety of the system due to potential collisions among the robot and may lead to failure to perform the desired task. In order to test the performance of our estimation and control algorithms, we set up an OptiTrack motion capture system outdoors (during a cloudy day) to provide ground truth position and orientation estimates and commanded the robot to take-off and hover above the origin at a height of 8 m. The estimates from the UKF and the motion capture system during this hover test are shown.
Figure 5.8: UKF estimates during a representative hovering experiment in an open area. Ground truth from an OptiTrack motion capture system, which was set up specifically for this experiment, are shown for reference. The position tracking errors had a standard deviation of 0.158 m in the horizontal direction and 0.386 m in the vertical direction.

The deviation of the UKF estimate from the desired represents error in control whereas the deviation of the OptiTrack estimate from the desired represents the error due to state estimation. From the plots we can see that the control errors are small, meaning that the robot is able to hover precisely at a position if given a perfect state estimate. In comparison, the estimation error is larger, having a standard deviation of around 16 cm in the horizontal plane and 39 cm in the vertical direction. This difference in the horizontal and vertical errors in estimation is mainly due to using the GPS position measurements only for the horizontal plane and using the pressure sensor for the height. The GPS measurements are fairly reliable with a standard deviation of around 1 m along the X and Y axes whereas the pressure sensor is affected by gusts of wind leading to the jumps in height that are seen in the plot.
5.9.2 Multiple robots in an open field

In order to test the complete system, we designed an experiment to simulate the deployment of the team of robots in an industrial complex. The experiment involved six robots flying above a 200 m × 100 m field (Figure 5.9) where, instead of flying around real-world obstacles, we provided virtual obstacles to the planner so that we can perform the experiment in a much safer manner. Figure 5.10 shows the various steps involved in the experiment. We start the robots from the ground with a separation of about 4 m between each other so that we can take-off without worrying about collisions between the robots. Once they take-off and reach a specified height we switch to the trajectory tracker which takes in inputs from the central planner and sends position commands to the position controller. Once this stage is reached (5.10a), the operator can command the robots from the user interface and send the robots to the desired goal positions (5.10b). Upon receiving the goal positions, the planner assigns the goals to the robots and plans trajectories for each of them (5.10c) which are then followed by the robots (5.10d).

As mentioned in section 5.5, the planner models the robots as circles with a radius of 2 m, even though the actual robot radius is around 0.3 m, in order to allow some localization and control errors. Table 5.1 provides estimates of the controller errors during the six-robot experiment which shows that the robots have control errors in the range 0.2–0.7 m. Adding
Figure 5.10: A series of snapshots of the user interface while running an experiment with six robots.
Table 5.1: Mean error between the desired position and estimated position in the horizontal plane for each robot during the six robot experiment

<table>
<thead>
<tr>
<th>Robot</th>
<th>XY Error [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.194</td>
</tr>
<tr>
<td>2</td>
<td>0.195</td>
</tr>
<tr>
<td>3</td>
<td>0.709</td>
</tr>
<tr>
<td>4</td>
<td>0.382</td>
</tr>
<tr>
<td>5</td>
<td>0.179</td>
</tr>
<tr>
<td>6</td>
<td>0.190</td>
</tr>
</tbody>
</table>

the localization error of approximately 0.2–0.5 m gives us a total error of up to 1.2 m thus justifying the choice in the planner.
Chapter 6

Multi-Robot Autonomy – Towards a Decentralized Paradigm without External Infrastructure

6.1 Need for a common reference frame

As demonstrated by the QuadCloud system, one of the main advantages of a multi-robot team is that we can perform a co-ordinated task using multiple robots which may not be feasible to do with a single robot. A simple example of this is dividing an area for surveillance into parts, each of which is assigned to one robot. One important thing that is required for such co-ordination is that the robots need to be able to localize themselves in a common reference frame. If the robots did not have a common reference frame, it would prevent any multi-robot planning since, with different reference frames, the interpretation of the commands from the planner would not be consistent across robots. For example, different commanded poses from the planner could lead to the robots being at the same location and collide with each other. Thus the system needs an explicit process in order to estimate the poses of the robots in a common reference frame. This requires the robots to share some
sensor measurements with other robots in order to estimate their relative poses with respect to each other. Depending on the type of measurements shared by the robots, there can be different ways to achieve this.

### 6.2 Using absolute/indirect measurements

The problem of a common reference frame is trivially solved when global pose estimates are directly available, such as from GPS or motion capture since the measurements are already in a common frame. Thus there is no separate estimation step required when using these sensor modalities.

In comparison, when the robot's pose is estimated using local sensing, such as visual or lidar odometry, each robot has a separate reference frame each of which typically has the origin at the starting pose of the robot and we need a method to establish a common reference frame.

One simple way to have the common reference frame is by starting the robots in predetermined poses or conversely measuring the relative poses between the robots at the start and using them to compute the common frame for the robots. This of course requires a lot of human effort since we either need to start the robots at fixed poses or measure the relative poses between the robots at the start. This is only feasible when the number of robots is small (less than 10) and becomes harder as the number of robots grows.

Another way is by adding fiducials, such as AprilTags, to the environment such that the robots can localize themselves with respect to these fiducials. Thus each robot can determine its pose with respect to these fiducials which acts like a common frame for the system. The robots can share this pose with other robots and hence infer their relative poses. The main disadvantage of this method is that it requires changes to the environment which might not be desirable in certain applications.

Yet another method is to share a map of features extracted from the measurements of the sensors of one robot with other robots [28, 64]. The other robots can match the features
extracted from their own sensors to this map of shared features and compute the relative pose between the robots. One common example of this technique is when a robot creates a map of feature descriptors, such as SIFT, extracted from camera images and other robots use this map to localize themselves with respect to the first robot. The main problem with this approach is that it relies on robots being able to create a map of the features present in the environment which may be difficult to achieve with some sensor modalities.

6.3 Using relative measurements

Instead of localizing the robots with respect to features in the environment, we can try to directly measure the relative poses between the robots by direct observation of the other robots. For example, the cameras on the robots can detect the other robots and measure the approximate range and bearing to each of the robots in their field of view. These measurements can be used to find the relative poses between the robots and hence establish a common reference frame. Typically these methods use unique markers, either using different colors or different patterns, on the robots to make the detection task simple as well as to obtain the identity of the observed robot. Some recent examples using this method for relative localization of a team of aerial robots include [102] where the authors used colored circular markers on the robots to obtain relative bearings between the robots and [89] which used a concentric circle pattern on the robots to get the relative pose between them.

The main disadvantage of this method is that the robots needs to be modified to carry some type of marker and care has to be taken to ensure that the markers on each of the robots are distinct from each other so that they can be used for identity purposes.

6.4 Using unlabeled relative measurements

In our system, the robots are homogeneous and interchangeable. This means that all the robots look the same. This creates a challenge when detecting the robots since the measurements provide no information about the identity of the robots. This can be avoided by putting some distinct marker on each of the robots, but this requires modification of the
robots which can become cumbersome as the team size grows. Due to this missing identity in the measurements, we call them unlabeled measurements. For estimation, these unlabeled measurements need to be associated with the robot that is observed in the measurements, leading to the classical data association problem.

The most common way of obtaining the relative measurements between robots is by using a camera on the robot to look at other robots and using object detectors to obtain the regions of the image containing the other robots. These object detectors provide either a bounding box around the region in the image containing the object or mark the pixels of the image belonging to the object. As mentioned above, all the robots look the same, hence the object detectors provides no information about the identities of the robots and we need a way to handle these unlabeled measurements in order to estimate the relative poses of the robots. In addition, as with any image based method, we can have some false detections due to clutter in the environment (false positives) as well as some missed detections (false negatives). So we need a technique to estimate the relative poses of the robots in the team using unlabeled and noisy relative measurements from each of the robots.

### 6.5 Problem formulation

Consider a homogeneous team of robots, each able to estimate its own pose in some robot-specific reference frame. An example is a team of robots with each robot running a visual inertial odometry (VIO) algorithm for state estimation with the reference frame for each robot being their starting position. We assume that each robot is able to detect other robots nearby and compute an approximate estimate of the relative pose to them. Later, we extend the method to be able to use bearing-only measurements as well. Note that the robot team is homogeneous which means that these observations of the other robots do not provide any identity information about the robot. The observations of the other robots are also corrupted by noise and we may have missed or false detections. Using these measurements, we want to estimate the position of all the robots in the team in a common reference frame. In this case, since there is no global reference frame defined, we use the reference frame of one of
the robots as the common frame.

Suppose we have a team of $N$ robots each estimating its position and orientation in its own reference frame. We denote the position and orientation of robot $i$ in robot $j$’s reference frame by $f_j p_i$ and $f_j R_i$ respectively where $i,j \in \{1,\ldots,N\}$. Each of the robots is only able to estimate their position and orientation in their own reference frame, so they only have a direct estimate of $f_i p_i$ and $f_i R_i$. We want to estimate the positions and orientations of all the robots in common reference frame which we choose to be the reference frame of the first robot, i.e. we want to estimate $(f_1 p_i, f_1 R_i) \forall i \in \{1,\ldots,N\}$. Now,

$$
\begin{align*}
  f_1 p_i &= f_i R_{fi} f_i p_i + f_i t_{fi}, \\
  f_1 R_i &= f_i R_{fi} f_i R_i,
\end{align*}
$$

(6.1)

where $f_i R_{fi}$ is the rotation matrix which converts vectors in robot $i$’s reference frame to robot 1’s reference frame and $f_i t_{fi}$ is the translation of robot $i$’s reference frame in robot 1’s reference frame. Thus, to estimate the position and orientation of robot $i$ in robot 1’s frame, we need to estimate the rotation, $f_1 R_{fi}$, and translation, $f_1 t_{fi}$, between the two frames.

In a VIO system, the direction of gravity in the robot’s frame is observable and only the rotation perpendicular to gravity is unobservable [36]. Thus we can assume that the direction of the gravity vector is known for each robot which allows us to choose the reference frames such that all of them have their XY plane perpendicular to the gravity direction and hence there is only one degree of freedom left in the rotation between the frames.$^1$ Using this, we can simplify Equation 6.1 to

$$
\begin{align*}
  f_1 p_i &= R_z(f_1 \psi_{fi}) f_i p_i + f_i t_{fi}, \\
  f_1 R_i &= R_z(f_1 \psi_{fi}) f_i R_i,
\end{align*}
$$

(6.2)

where $f_1 \psi_{fi}$ is the rotation perpendicular to the gravity direction, i.e. the yaw, between the

$^1$We assume that the frames are close enough so that the curvature of the earth can be neglected.
frames $f_i$ and $f_1$.

There are two types of measurements for this system:

1. The position and orientation of a robot in its own reference frame from the estimator running on the robot.

2. The relative positions of other robots which are within the field of view of a robot.

The measurement model for the first is

$$z_i = \begin{bmatrix} f_i p_i \\ f_i R_i \end{bmatrix} = \begin{bmatrix} f_1 R_{f_i}^T (f_i p_i - f_i t_{f_i}) \\ f_1 R_{f_i}^T f_1 R_i \end{bmatrix}.$$  \hspace{1cm} (6.3)

where $f_1 R_{f_i} = R_z(f_1 \psi_i)$ for compactness.

The measurement model for the relative measurement when robot $i$ observes robot $j$ is given by,

$$z_{ij} = i p_j = f_1 R_i^T (f_i p_j - f_i p_i).$$  \hspace{1cm} (6.4)

However, the relative measurements do not provide any identity information about the other robot while the measurement model requires the knowledge of which robot is being observed, hence we cannot use this measurement model directly for updating the state.

As mentioned earlier, we can assume that the direction of the gravity vector is known for each robot and only the rotation around the gravity vector is unobservable. Thus we only need to estimate the one degree of freedom rotation around the gravity vector in the robot’s orientation. We can decompose the orientation $f_1 R_i$ into two parts, one corresponding to the rotation around the gravity vector and the other being the rotation in plane perpendicular to gravity:

$$f_1 R_i = R_z(f_1 \psi_i) f_1 R_i,_{xy},$$  \hspace{1cm} (6.5)

where $f_1 \psi_i$ represents the rotation angle about the gravity vector, i.e. the yaw angle, between
the reference frame of the first robot and robot i’s current pose. We assume that the $f_iR_{i,xy}$ is known.

From Equation 6.3, Equation 6.4, and Equation 6.5, we can see that we need to estimate the 3D position ($f_ip_i$) and yaw angle ($f_i\psi_i$) of robot i with respect to the reference frame of the first robot as well as the 3D translation, $f_it_{fi}$ and the yaw, $f_i\psi_{fi}$, between the robot frames. Thus our state vector is,

$$
\mathbf{x} = \begin{bmatrix} f_1p_1 \ T & f_1\psi_1 & f_1t_{f1}^T & f_1\psi_{f1} & \cdots & f_Np_N^T & f_N\psi_N & f_Nt_{fN}^T & f_N\psi_{fN} \end{bmatrix}^T.
$$

(6.6)

Note that we include $f_1t_{f1}$ and $f_1\psi_{f1}$ in the state vector to simplify the formulation, though both of them are identically zero.

6.6 Approach

We develop a centralized algorithm which fuses the information from each robot to estimate the state of the full system. At each time step, each robot communicates its estimated position and velocity in its own reference frame as well as the measurements of relative positions of the other robots detected in its camera image.

We use a system model where the velocity is used as the input to the system. We assume that the velocity estimate is corrupted with zero-mean Gaussian noise. We also assume a random walk behavior for the robot frame positions and yaw to handle the drift in the VIO estimates. Thus, the state equation is,
\[ \dot{x} = \begin{bmatrix} f_1 \dot{p}_1 \\ f_1 \dot{\psi}_1 \\ f_1 t_{f_1} \\ f_1 \dot{\psi}_{f_1} \\ f_2 \dot{p}_2 \\ f_1 \dot{\psi}_2 \\ \vdots \\ f_N \dot{p}_N \\ f_1 \dot{\psi}_N \\ f_1 t_{f_N} \\ f_1 \dot{\psi}_{f_N} \end{bmatrix} = \begin{bmatrix} f_1 v_1 \\ f_1 \omega_1 \\ 0 \\ 0 \\ f_2 v_2 \\ f_1 \omega_1 \\ \vdots \\ f_N v_N \\ f_1 \omega_1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \eta_{v_1} \\ \eta_{\omega_1} \\ 0 \\ 0 \\ \eta_{v_2} \\ \eta_{\omega_2} \\ \vdots \\ \eta_{v_N} \\ \eta_{\omega_N} \\ 0 \\ 0 \end{bmatrix}, \]

where \( f_i v_i, f_i \omega_i \forall i \in \{1, \ldots, N\} \) represents the translational and angular (about the Z-axis) velocity estimates received from each robot in their own reference frame and the \( \eta \) terms represent the noise which we assume to be zero-mean Gaussian. We can write this in the standard form,

\[ \dot{x} = Ax + Bu + w, \]

where

\[
A = 0_{7N}, \quad B = \begin{bmatrix}
I_4 & 0_4 & 0_4 & \cdots & 0_4 \\
0_4 & I_4 & 0_4 & \cdots & 0_4 \\
0_4 & 0_4 & I_4 & \cdots & 0_4 \\
0_4 & 0_4 & 0_4 & \cdots & I_4 \\
0_4 & 0_4 & 0_4 & \cdots & 0_4
\end{bmatrix}, \quad u = \begin{bmatrix} f_1 v_1 \\ f_1 \omega_1 \\ \vdots \\ f_N v_N \\ f_1 \omega_N \end{bmatrix}, \quad w \sim \mathcal{N}(0, Q),
\]

with \( Q \) being the covariance of the noise. Note that \( Q \) is positive semi-definite due to zero noise for the \( f_i t_{f_i} \) and \( f_i \dot{\psi}_{f_i} \) terms.

We can convert this continuous time equation into the discrete form using zero-order hold for the input,

\[ x_k = F x_{k-1} + G u_{k-1} + w_{k-1}, \quad (6.7) \]
where \( k \) is the current time step, \( F = I_{7N}, G = B\Delta t, w_{k-1} \sim \mathcal{N}(0, Q_d) \) is the discrete-time noise with \( Q_d = Q\Delta t \), and \( \Delta t \) is the sampling time for the discrete system.

At any time, there can be multiple robots in the field of view of the camera which would lead to multiple relative measurements, one for each detected robot. We denote this set of relative measurements by robot \( i \) at time step \( k \) by \( Z^i_k = \{z^i_{i_1}, z^i_{i_2}, \ldots, z^i_{M_i}\} \), where \( i_1, i_2, \ldots, i_{M_i} \) are the unknown indices of the observed robots and \( M_i \) is the number of relative measurements for robot \( i \).

In order to handle the unknown association between the measurements and robots we use methods from the field of multi-target tracking that were originally developed for radar tracking applications. Appendix A provides a brief overview of some of these methods.

In our system, the number of robots is fixed and known a priori, so we do not need to worry about new targets appearing or existing ones disappearing. This allows us to use a method similar to the JPDA filter for the unlabeled measurements.

Assume that at time step \( k - 1 \), the current estimate for the robot state is \( x_{k-1|k-1} \) and the covariance associated with the state is \( P_{k-1|k-1} \). We propagate the state and covariance through the system model (Equation 6.7) to obtain the a priori estimates for the state and covariance at time step \( k \),

\[
x_{k|k-1} = Fx_{k-1|k-1} + Gu_{k-1}, \\
P_{k|k-1} = FP_{k-1|k-1}F^T + Q. 
\] (6.8)

If we assume Gaussian noise for the measurements, the true PDF for the state is a Gaussian mixture. The JPDAF approximates this by a single Gaussian. It also assumes that conditioned on the past, the target states, and in turn the generated measurements, are independent. As can be seen from the model for the relative measurements (Equation 6.4), the measurements depend on multiple states. This leads to coupling of the states and we cannot use the
standard JPDA filter for state estimation.

6.6.1 CPDAF

The Coupled Probabilistic Data Association (CPDA) filter [12] extends the JPDAF to handle this coupling between the states. Compared to the JPDAF where we compute the association probabilities of single targets with single measurements, in the CPDAF we compute the association probabilities of subsets of the targets with subsets of the measurements.

We briefly describe the CPDAF with an example having \( N \) targets with 1-dimensional states and \( M \) 1-dimensional measurements at the current time step. Note that in our actual problem, for each robot \( i \) with relative measurements, we have \( N - 1 \) targets with 4-dimensional states and \( M_i \) 3-dimensional measurements.

First we define some helper variables, \( \phi \) is a \( N \)-dimensional binary vector representing which targets have been detected at the current time step, \( D_T = \sum_{i=1}^{N} \phi_i \) is the number of detected targets, \( \tilde{\chi} \) is a \( D_T \times M \) matrix which selects and generates a permutation of \( D_T \) true measurements among all the measurements at the current time step.

As shown in [12], the association probabilities in the CPDAF are given by,

\[
\beta(\phi, \tilde{\chi}) = \frac{1}{c} F(\phi, \tilde{\chi}) \cdot \lambda^{(N-D_T)} \cdot \prod_{i=1}^{N} \left( (1 - P_d^i (1 - \phi_i))(P_d^i \phi_i) \right), \tag{6.9}
\]

where \( c \) is a normalization constant, \( \lambda \) is the false observation spatial density, \( P_d^i \) is the detection probability of target \( i \), and

\[
F(\phi, \tilde{\chi}) = \frac{\exp \left( \mu(\phi, \tilde{\chi})^T S(\phi)^{-1} \mu(\phi, \tilde{\chi}) \right)}{\sqrt{(2\pi)^{D_T} \det(S(\phi))}}, \tag{6.10}
\]

where

\[
\mu(\phi, \tilde{\chi}) = \tilde{\chi} z - \Phi(\phi) H x, \quad S(\phi) = \Phi(\phi) (H P H^T + R) \Phi(\phi)^T, \tag{6.11}
\]

\( x \) is the current estimate of the state, \( H \) is the \( N \times N \) dimensional observation matrix, \( z \) is
the $M$-dimensional vector of stacked observations, $\Phi(\phi)$ is a $D_T \times N$ binary matrix with the $i^{th}$ row equal to the $i^{th}$ non-zero row of $\text{diag}(\phi)$, and $R$ is the expected measurement covariance.

We use validation gates for the measurements in order to ignore associations with low probability. The validation gate is defined as

$$G^i := \left\{ z' \mid \left( z' - H_i x \right)^T \left( H_i P H_i^T + R_{ii} \right)^{-1} \left( z' - H_i x \right) \leq \gamma \right\}, \quad (6.12)$$

where $H_i$ is the block of the observation matrix corresponding to the measurement for the $i^{th}$ target, $R_{ii}$ is the block of the measurement covariance corresponding to the $i^{th}$ target’s measurement, and $\gamma$ is a threshold selected from a chi-squared distribution with the appropriate degrees of freedom.

If a measurement falls outside the gate $G^i$, for example if $z_j \notin G^i$, we remove all the association hypotheses which have the $j^{th}$ measurement associated with target $i$, i.e. we set the corresponding $\beta(\phi, \tilde{\chi}) = 0$. Note that with gating, the normalization constant $c$ in Equation 6.9 needs to be computed taking the gating into account.

The CPDAF measurement update step is similar to the standard Kalman Filter but since we have multiple potential measurements each with a corresponding association probability, the state and covariance are updated as,

$$x_{k|k} = x_{k|k-1} + \sum_{\phi} K(\phi) \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}), \quad (6.13)$$
\[ P_{k|k} = P_{k|k-1} - \sum_{\phi} K(\phi)\Phi(\phi)P_{xx} \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \]
\[ + \sum_{\phi} K(\phi) \left( \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi})^T \right) K(\phi)^T \]
\[ - \left( \sum_{\phi} K(\phi) \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}) \right) \cdot \left( \sum_{\phi'} K(\phi') \sum_{\tilde{\chi}'} \beta(\phi', \tilde{\chi}') \mu(\phi', \tilde{\chi}') \right)^T, \]
\[ \text{(6.14)} \]
where \( K(\phi) = PH^T\Phi(\phi)^T S(\phi)^{-1} \) is the Kalman gain.

6.6.2 CPDAF with nonlinear measurements

In our system, the measurement models for the self and relative measurements (Equation 6.3 and Equation 6.4) are non-linear in the state. The CPDAF developed in [12] assumes a linear measurement model and hence cannot be used directly for our system. In order to handle the non-linear measurement models in our system, we modify the CPDAF update step to compute the predicted measurement and the related covariances similar to the Unscented Kalman Filter (UKF).

The first change is in Equation 6.11, where we have a non-linear observation function \( h(x) \) instead of an observation matrix. The updated equations are,
\[ \mu(\phi, \tilde{\chi}) = \tilde{\chi}z - \Phi(\phi) h(x), \quad R(\phi) = \Phi(\phi) P_{zz} \Phi(\phi)^T, \]
\[ \text{(6.15)} \]
where \( z \) is the \( M \)-dimensional vector of stacked observations, \( h(x) \) is the \( N \)-dimensional vector of predicted measurements, \( \Phi(\phi) \) is a \( D_T \times N \) binary matrix with the \( i^{th} \) row equal to the \( i^{th} \) non-zero row of \( \text{diag}(\phi) \), and \( P_{zz} \) is the innovation covariance. The computation of \( P_{zz} \) is same as that used for the UKF, see Equation 4.3.

The validation gate computation also changes slightly for the nonlinear model, where, instead of Equation 6.12, we have,
\[ G^i := \left\{ z' \mid (z' - h_i(x))^T P^{-1}_{zz} (z' - h_i(x)) \leq \gamma \right\}, \]
\[ \text{(6.16)} \]
where $h_i(x)$ is the measurement corresponding to the $i^{th}$ target, and $P_{zi,z_i}$ is the block of the $P_{zz}$ matrix corresponding to that measurement.

And finally, the CPDAF state and covariance update with the nonlinear measurements is,

$$x_{k|k} = x_{k|k-1} + \sum_{\phi} K(\phi) \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}) ,$$  

(6.17)

$$P_{k|k} = P_{k|k-1} - \sum_{\phi} K(\phi) \Phi(\phi) P_{xx} \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi})$$

$$+ \sum_{\phi} K(\phi) \left( \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi})^T \right) K(\phi)^T$$

$$- \left( \sum_{\phi} K(\phi) \sum_{\tilde{\chi}} \beta(\phi, \tilde{\chi}) \mu(\phi, \tilde{\chi}) \right) \cdot \left( \sum_{\phi'} K(\phi') \sum_{\tilde{\chi}'} \beta(\phi', \tilde{\chi}') \mu(\phi', \tilde{\chi}') \right)^T ,$$

(6.18)

where

$$K(\phi) = (\Phi(\phi) P_{xx})^T P_{zz}^{-1}$$

is the Kalman gain, and $P_{xx}$ is the measurement-state cross-covariance matrix, computed as in Equation 4.4.

### 6.6.3 Efficient Measurement Gating

The total number of joint association events given $N$ targets and $M$ measurements can be computed as [17],

$$\text{Number of joint association events} = \sum_{l=0}^{\min(M,N)} \binom{N}{l} \binom{M}{l} \cdot l! ,$$

(6.19)

where $l$ represents the number of observed targets in each iteration of the summation, $\binom{N}{l}$ is the number of ways to select $l$ observed targets out of $N$ targets, $\binom{M}{l}$ is the number of ways of selecting $l$ true measurements from $M$ measurements, and $l!$ is the number of possible assignments of $l$ measurements to $l$ targets.

The number of these joint association events grows very quickly with the increase in the
number of targets and measurements as can be seen in Figure 6.1. Thus the worst-case complexity of the JPDAF/CPDAF algorithms is very high and it is intractable to directly use these algorithms for large number of targets.

In order to reduce this computational complexity, we use measurement validation gates to ignore low probability association events. The validated measurements for the \( i \)th target are computed as shown in Equation 6.12 and Equation 6.16.

By ignoring these low probability events, the measurement gating removes a lot of terms in the computations for the mean and covariance in Equation 6.17 and Equation 6.18 and allows us to use the filter with larger number of targets and measurements.

Many approaches have been proposed for computing the association probabilities in an efficient manner by assuming certain properties about the PDF of the measurements or the states. For example, [18] showed that given a particular target-to-measurement assignment, if we assume that the posterior PDF of the state of the targets are independent, then the association probabilities can be computed efficiently using the Matrix Permanent. [67] and [40] proposed a novel formulation which allows the computation of the association

Figure 6.1: Number of joint association events as a function of the number of targets and number of measurements. Note that the Z-axis has log scale.
probabilities in sub-exponential time by representing the target hypotheses using a ‘net’ structure but they made the assumption that the posterior individual target states are independent which allows certain simplifications of the probability computations. These assumptions do not hold in our case since the unlabeled relative measurements depend on the state of two robots leading to coupling of their PDF, so we are unable to use these efficient methods for computation of the association probabilities.

We use an algorithm inspired by the one presented in [117] to enumerate only the valid joint association events after gating. The algorithm, shown in Algorithm 6.1, works by creating a tree of association events, where each level of the tree represents a target while the branches represent the measurement associations, and traverses this tree in a depth-first fashion. Figure 6.4 shows an example of such a tree with the valid branches that are selected by the algorithm. Using this algorithm, we can reduce the number of joint association events that need to enumerated when measurements are received allowing us to use our method for a larger number of robots. The reduction in the number of evaluated hypotheses due to gating is shown in Figure 6.2. Comparing Figure 6.1 and Figure 6.2, it can be seen that we can use
Figure 6.3: Comparison of the computation time required for the CPDAF update step with and without gating. Note that the Y-axis has log scale.

A larger number of robots in the team for the same number of enumerated joint association events when running the CPDAF with gating. Conversely, this leads to a significantly faster runtime for the CPDAF update step for the same number of robots compared to when no gating is used (see Figure 6.3).

6.7 Experiments

The robots used in the experiments are the same quadrotors as those described in ???. The robots are approximately 0.4 m in diameter and have a mass of 1.3 kg. They carry a custom designed board consisting of a Nvidia TX2 and a stereo camera synchronized with an IMU. We used the stereo VIO described in subsection 4.3.1 to estimate the pose of each robot. The VIO assumes that the starting position of the robot is the origin of its reference frame, leading to each robot having a different reference frame.

6.7.1 Detection of other robots

Instead of modifying the robots by adding markers to them for detection, we use a convolutional neural network architecture based on ERFNet for pixel-wise semantic segmentation [86]. We selected this architecture as it has been specifically optimized for efficient perfor-
Algorithm 6.1 Enumerating target to measurement associations using gated measurements

Require: $V$ is a list of length equal to the number of detected targets with each element being a list of validated measurements for the corresponding target

1: procedure EnumerateAssociations($V$)
2:  for $i \leftarrow 0$ to len($V$) - 1 do
3:    $V_{idx}[i] \leftarrow 0$
4:  done $\leftarrow$ false
5:  for all $V^i \in V$ do
6:    if $V^i = \emptyset$ then
7:      done $\leftarrow$ true
8:      break
9:  $X \leftarrow \emptyset$
10:  $j \leftarrow 0$
11:  while $\neg$done do
12:    while $j < \text{len}(V)$ do
13:      valid, $X_j, V_{idx,new} \leftarrow \text{GetNext}(V[j], V_{idx}[j], X)$
14:      if valid then
15:        Append $X_j$ to $X$
16:        $V_{idx}[j] \leftarrow V_{idx,new}$
17:        $j \leftarrow j + 1$
18:      else
19:        if $j = 0$ then
20:          done $\leftarrow$ true
21:          break
22:        $V_{idx}[j] \leftarrow 0$
23:        Remove last element of $X$
24:        $j \leftarrow j - 1$
25:      end if
26:    end while
27:    end while
28:  end while
29:  Use $X$ \quad \triangleright X[i] \text{ is the measurement associated with target } i$

function GetNext($A, idx, X$)
30:  $X_j \leftarrow 0$
31:  valid $\leftarrow$ false
32:  while $\neg$valid and $idx < \text{len}(A)$ do
33:    $X_j \leftarrow A[idx]$
34:    $idx \leftarrow idx + 1$
35:    valid $\leftarrow$ true
36:  for all $x \in X$ do
37:    if $x = X_j$ then
38:      valid $\leftarrow$ false
39:      break
40:  end for
41:  return valid, $X_j, idx$
Figure 6.4: An example of the tree of association events for three targets and three measurements after gating. In this example, the gated measurements for the three targets are \{1, 2\}, \{1, 2, 3\}, and \{3\} respectively. Note that one measurement can only be associated with a single target for a valid joint association, hence some of the branches in the tree are invalid. The set of valid joint associations in this case is \{(1,2,3), (2,1,3)\}.

Performance even on embedded GPUs such as the Nvidia TX2. The original architecture has an encoder-decoder structure, with 16 encoders and 7 decoder layers. Each layer contains an efficient combination of factorized convolutional layers and residual connections, also choosing wider over deeper in overall network structure, resulting in efficient use of parameters.

We collected a dataset of 225 images over the course of multiple flight tests. These images were then labelled with semi-coarse accuracy, with each pixel in the image corresponding to one of two classes, either quadrotor or background. We then randomly sampled a validation set of 55 images and trained different variants of the original ERFNet architecture. The training dataset was augmented to add random rotations, translations, motion blur, and mirroring across the vertical axis since our quadrotor structure is close to symmetric. This augmentation results in a training sample size of 1200 images.

The original ERFNet architecture has roughly 2.06 million trainable parameters, and achieves a test Intersection-over-Union (IoU) accuracy of 77.01% on the foreground quadrotor class. The typical runtime of this model on the Nvidia TX2 is 120 ms for a 640×512 grayscale image. We used the original learning rate schedule and trained the network for 1000 epochs, taking approximately 7 hours on a single Nvidia Titan Xp. We wanted to increase the
efficiency of the network to achieve execution times closer to 10 frames/second, with the belief that a two-class segmentation network should not require as many parameters as one that successfully learns a model for 19 classes which the original ERFNet was developed for. Given the already minimal nature of the decoder structure of this network, our attempts at reducing parameter size here resulted in significantly poorer performance.

However, since the bulk of the parameters are concentrated in the encoder layers, we shrank the overall encoder by 6 layers, maintaining the original encoder output resolution, totalling around 1.1 million parameters and achieved a steady 10 frames/sec. Interestingly, this network architecture achieves a better foreground quadrotor IoU accuracy of 79.24%, indicating that this smaller network is capable of performing well on a simpler task than the original, and can be trained to a higher accuracy in a shorter period of time. We further tried to shrink the parameters of the model by reducing the encoder layers from 10 to 7, but the resulting model, with roughly 0.73 million parameters, performed significantly worse than the previous with a foreground quadrotor IoU accuracy of only 74.81%.

The output of the detector is a mask with the quadrotor pixels highlighted as shown in Figure 6.5. As can be seen from the figure, the mask does not provide any identity information about the robots present in the image. Using the mask and the known size of the robot, we
\[
\tan \alpha = \frac{x_r}{Z} = \frac{x_i}{f}, \quad \tan \beta = \frac{x_r + w_r}{Z} = \frac{x_i + w_i}{f}
\]

\[
\Rightarrow \quad \frac{w_r}{Z} = \frac{w_i}{f} \quad \Rightarrow \quad Z = \frac{w_r f}{w_i}
\]

Figure 6.6: Computing the distance of the robot from the image size. \(w_r\) is the real robot width, \(w_i\) is the robot width in the image, \(f\) is the camera focal length, and \(Z\) is the distance along the optical axis of the robot from the camera.

can estimate the relative position of the detected robots. First we extract the boundary of each connected component in the image and then fit an ellipse to each contour. The center of the ellipse in the image provides a bearing estimate for the relative position of the robot while the length of its major or minor axes can be used to compute the distance to the robot (as shown in Figure 6.6), thus allowing us to get the full 3D relative position of the robot in the camera frame.

### 6.7.2 Results

We show results from an experiment consisting of three robots in a 5 m × 5 m area. The CPDAF is run on the ground station and each robot sends its pose and velocity estimates from the VIO as well as the relative position measurements of the other robots detected in the camera image to the ground station. The filter has no knowledge about the starting positions of the robots and only knows whether the initial yaw is closer to 0° or 180°, so we initialize it with a large covariance. In this experiment, two of the robots (Robot 1 and 3) were flown manually by two pilots while Robot 2 was kept stationary during this test. The values of the parameters used for this experiment were: detection probability of 0.9, gating probability of 0.99, and false observation density of 0.001 obs/m³.

Figure 6.7 shows the estimates of the position and yaw of the robots in the common reference frame which is chosen to be Robot 1’s starting frame. Since we use Robot 1’s frame as the
Figure 6.7: Comparison of the position and yaw of the three robots in the common reference frame estimated using our approach during a flight test with ground truth from a motion capture system.
common reference frame, its local estimate is already the ground truth. Due to the way we started the processes on the robots, the measurements sent from Robot 2 and 3 start at approximately 3s and 6s respectively. As mentioned before, Robot 2 is stationary while Robot 3 is moving, and we can see that their estimates converges to the true state within 15s after their measurements start arriving at the ground station.

6.8 Extension: Bearing only relative measurements

In the previous section, the relative measurements of the other robots detected in the camera image consist of their full 3D relative positions. The size of the detected robot is assumed to be known in order to compute the distance to the robot and hence the full 3D position. In cases when the detection is not precise or if the size of the robot is unknown, we can only use the bearing information from the detection. This bearing information is clearly not as rich as the full 3D relative position but we show that it can still be used with the CPDAF framework to estimate the position and yaw of the robots in a common reference frame.

6.8.1 Measurement Model

We need to update the measurement model from Equation 6.4 to account for the relative bearing measurements. The relative bearing measurement of robot $j$ by robot $i$, represented by $z_{ij}^i$, is computed as,

$$z_{ij}^i = \frac{i^j p_j}{\|i^j p_j\|}, \quad i^j p_j = f^1 R_i^T \left( f^1 p_j - f^1 p_i \right). \quad (6.20)$$

The one complication with using the unit vector for the relative bearing measurement is that this representation has three degrees of freedom while the actual measurement only has two degrees of freedom. This needs to be taken into account during the measurement update of the CPDAF when computing the innovation and covariances. To handle this, we use the method described in [35] where the difference between two 3D unit vectors is defined to be in the two dimensional tangent space of $S^2$. Let $x = [x_1 \ x_2 \ x_3]^T$ and $y = [y_1 \ y_2 \ y_3]^T$ be two 3D unit vectors. We define this difference in the tangent space (represented by $\mathbb{D}$) between
these two 3D unit vectors as,

\[ y \boxtimes x := \log(R_x^T y) \]

where

\[
\log(x) := \begin{cases} 
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} & \text{if } \sqrt{x_2^2 + x_3^2} = 0 \\
\begin{bmatrix} \tan^{-1}(r, x_1) \\ r \\ x_3 \end{bmatrix} & \text{otherwise, where } r = \sqrt{x_2^2 + x_3^2}, \end{cases}
\]

\[
R_x := \begin{bmatrix} x_1 & -r & 0 \\ x_2 & x_1 \cos(\alpha) & -\sin(\alpha) \\ x_3 & x_1 \sin(\alpha) & \cos(\alpha) \end{bmatrix}
\]

where \( r = \sqrt{x_2^2 + x_3^2}, \) \( \alpha = \tan^{-1}(x_3, x_2) \)

6.8.2 Results

We show results from the same experiment as subsection 6.7.2, but now only using relative bearing measurements in Figure 6.8. Note that we use the first robots starting frame as the common reference frame, so the local estimate of the first robot is the same as its ground truth. Since the relative bearing measurements do not have as much information as the relative position measurements, the filter with relative bearing measurements takes a longer time to converge compared to the earlier result. We can see from the figure that the estimates with the bearing only measurements take approximately 30 s to converge compared to 15 s with the relative position measurements. Thus, even though it is feasible to estimate the position and yaw of the robots in a common reference frame with unlabeled relative bearing measurements, it is significantly better to use the relative position estimates computed using the size of the robot in the image.
Figure 6.8: Estimates of the position and yaw of the three robots in the common reference frame using our approach with unlabeled relative bearing measurements during a flight test.
Chapter 7

Conclusion

In this work we have described the development of a team of quadrotor robots which can perform cooperative tasks without requiring any external infrastructure. We developed the system in three steps.

First, we described the system design and algorithms for robust single robot autonomy and demonstrated it with experiments in a variety of environments. We developed new algorithms for state estimation, control, and planning which allow the robot to autonomously navigate through natural and man-made environments at high speeds.

Next, we developed a centralized system architecture to allow a single operator to deploy, control, and monitor a team of robots. In order to focus on the multi-robot coordination problem, the robots used for this part utilized GPS for state estimation and required known maps of the environment for obstacle avoidance, which limited the usage of the system to known outdoor environments.

Finally, we combined the robots from the first part with this multi-robot framework. The main challenge here is that due to using onboard sensing for state estimation, each robot has a different reference frame for its state estimates, typically with the origin at the starting pose.
of the robot while the high-level planner for the team requires all the robot state estimates to be in a common reference frame. We developed a method which does not require any external infrastructure or creation of maps that need to be shared across the robots and only uses the relative position/bearing measurements of nearby robots detected using the onboard camera to solve this problem.

The complication in this method is that these relative measurements do not contain any identity information since all the robots in the team are equivalent and look the same. In order to fuse these unlabeled measurements and estimate the states of the robots in a common reference frame, we used the Coupled Probabilistic Data Association Filter (CPDAF) and developed an approximation to reduce its computational complexity enabling online estimation with a team of up to 15 robots.
Appendix A

Multi-target tracking

Techniques for estimation with unlabeled measurements have been studied extensively in the past for use in radars. The measurements from radars typically consist of the bearing and range to the targets but contain no information about the target identity. There measurements are corrupted by measurement noise as well as false positives due to spurious objects being detected. The main challenge with radars is to estimate the location of one or multiple targets using these noisy measurements with no identity information. If the measurements are directly used in standard recursive filtering or optimization based methods, it can lead to divergence due to the wrong data association and false positives. This requires methods which handle this unknown data association as well as potential false positives when incorporating these sensor measurements into the target’s estimate. We describe a few methods which have been successfully used to solve this problem.

A.1 Joint Probabilistic Data Association

The earliest works in target tracking in clutter with unknown data association used a nearest neighbor type association for matching the measurements to the targets [44, 92–94]. These methods chose the measurement which is nearest to the predicted measurement, based on the current estimate for each target, and used that single measurement to update the
target’s estimate. This led to wrong estimates and lost tracks as the number of false positive measurements due to clutter increased.

A.1.1 Probabilistic Data Association Filter

Instead of using the hard association of the nearest-neighbor solutions, Bar-Shalom and Tse [5] proposed the Probabilistic Data Association Filter (PDAF) for single target tracking in clutter where they assign an association probability to each of the measurements leading to soft associations. These probabilities are computed based on the distance between the predicted measurement given the estimate of the state and the actual measurement received from the sensor. The main assumptions made by the PDAF are [4]:

- One target of interest is present, and the target’s state transition and measurement generation functions are known.
- The past information can be summarized approximately by a sufficient statistic in the form of a Gaussian posterior.
- A measurement validation region can be constructed around the predicted measurement in order to select candidate measurements for the target. Also, at most one measurement in the validation region can be the true measurement from the target.
- The probability density function of a measurement conditioned upon all the past measurements, given that it is a true measurement, is known.
- The probability density function of a measurement given that it is not associated with the target is uniform.
- No inference can be made about the number of measurements at each time step based on the past measurements.
- The prior probability of each of the measurements being the true measurement is the same and does not depend on past data.
Consider a target with the state transition equation given by

\[ x_k = f(x_{k-1}, u_k, v_k), \]

where \( x \) is the state of the target, \( u \) is the input to the system and \( v \) is the process noise with a known PDF.

The true measurement generated by the target is given by

\[ z_k = h(x_k, w_k), \]

where \( z \) is the measurement and \( w \) is measurement noise with a known PDF.

In each sensor reading, the true measurement may or may not be present and there may be false measurements created by the clutter. Thus we represent the sensor reading at each time step as a set \( Z_k \).

The key assumption made by the PDAF is that all the past information can be summarized by a Gaussian posterior,

\[ P(x_k | Z_k, Z_{k-1}, \ldots) = \mathcal{N}(x_k | \hat{x}_k, \Sigma_k), \]

where \( \hat{x} \) is the estimate of the state and \( \Sigma \) is the covariance associated with the estimate.

The PDAF follows the Kalman filter framework with slight modifications in the update step to handle the data association uncertainty with multiple measurements. Depending on linearity of system dynamics and measurement function, the PDAF can either be based on the KF or the EKF.

Assume that the \textit{a posteriori} estimate of the state at time step \( k - 1 \) is \( \hat{x}_{k-1|k-1} \) with covariance \( \Sigma_{k-1|k-1} \). Using the prediction step of the KF/EKF, we obtain the \textit{a priori} estimate of the state and covariance at time step \( k \), \( \hat{x}_{k|k-1} \) and \( \Sigma_{k|k-1} \) respectively.
Using the \textit{a priori} state estimate, the validation region for the measurements can be computed as an ellipsoid around the predicted measurement,

\[
\mathcal{V}(k, \gamma) := \left\{ z : \left( z - \hat{z}_{k|k-1} \right)^T S_k^{-1} \left( z - \hat{z}_{k|k-1} \right) \leq \gamma \right\},
\]

where \( \hat{z}_{k|k-1} \) is the predicted measurement from the \textit{a priori} state estimate, \( S_k \) is the innovation covariance at time step \( k \) and \( \gamma \) is the gate threshold which corresponds to a gate probability \( P_G \).

Let the set of measurements after validation at time step \( k \) be \( \{ z^i_k \}_{i=1}^{m_k} \) where \( m_k \) is the number of validated measurements at time step \( k \). Assuming a Poisson density for the number of false measurements due to clutter with spatial density \( \lambda \), we compute the association probabilities for \( z^i_k \) being the correct measurement as

\[
\beta^i_k = \frac{\mathcal{L}^i_k}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}^j_k}, \quad i = 1, \ldots, m_k,
\]

where \( P_D \) is the target detection probability and \( \mathcal{L}^i_k \) is the likelihood ratio of the measurement \( z^i_k \) originating from the target rather than from the clutter and is given by

\[
\mathcal{L}^i_k = \frac{N(z^i_k | \hat{z}_{k|k-1}, S_k) P_D}{\lambda}.
\]

The probability that none of the measurements is the true measurement is

\[
\beta^0_k = \frac{1 - P_D P_G}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}^j_k}.
\]

Once we have the association probabilities, the state update equation is the same as the standard KF/EKF,

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \nu_k
\]
where $K_k$ is the standard Kalman gain and $\nu_k$ is the combined innovation given by,

$$\nu_k = \sum_{i=1}^{m_k} \beta_k^i \nu_k^i, \quad \nu_k^i = z_k^i - \hat{z}_{k|k-1}.$$ 

The covariance associated with the updated state is given by,

$$\Sigma_k|k = \beta_k^0 \Sigma_k|k-1 + (1 - \beta_k^0) \Sigma_c^k + \tilde{\Sigma}_k,$$

where $\Sigma_c^k$ represents the covariance of the state updated with the correct measurement computed as

$$\Sigma_c^k = \Sigma_k|k-1 + K_k S_k K_k^T,$$

while $\tilde{\Sigma}_k|k$ represents the increase in the covariance due to uncertainty in the data association and is given by

$$\tilde{\Sigma}_k|k = K_k \left( \sum_{i=1}^{m_k} \beta_k^i \nu_k^i \nu_k^i^T - \nu_k \nu_k^T \right) K_k^T.$$

The limitation of the PDAF is that it assumes that only a single target is present and models all the false measurements as random interference coming from clutter with a uniform density. In comparison, with multiple targets, true measurements from a nearby target can fall into the validation region of a target for multiple time steps and act as persistent interference. This persistent interference can lead to significant errors in the estimates from the PDAF since the nearby target’s measurements will be erroneously assigned high association probabilities and start affecting the state estimate of the first target.

### A.1.2 Joint Probabilistic Data Association Filter

The Joint Probabilistic Data Association Filter (JPDAF) [29] is an extension of the PDAF which handles the multi-target case by jointly computing the measurement-to-target association probabilities across the targets and clutter.

Most of the assumptions of the JPDAF are similar to the PDAF, except that instead of a
single target there are a known number of targets that are being tracked. The key idea of
the JPDA algorithm is evaluation of the conditional probabilities of the joint events,

\[ A = \bigcap_{j=1}^{m} A_{jt}, \]

where \( A_{jt} \) is the event that measurement \( j \) originated from target \( t_j \) with \( j = 1, \ldots, m \) and \( t_j = 0, 1, \ldots, N_T \). \( t_j \) is the index of the target to which the measurement \( j \) is associated with in the particular joint event. \( N_T \) is the known number of targets. Note that \( t = 0 \) corresponds to the “clutter target”. Among all the possible joint events, feasible joint association events are ones where each measurement only has a single source and each target, except clutter, has at most one measurement.

We now introduce two binary variables for convenience, \( \tau_j(A) \) called the measurement association indicator which indicates whether the measurement \( j \) is associated with an actual target in the event \( A \) and \( \delta_t(A) \) called the target detection indicator which denotes whether any of the measurements are associated with target \( t \) in the event \( A \). With these defined, the number of false (clutter) measurements in the joint event \( A \) is given by

\[ \phi(A) = \sum_{j=1}^{m} (1 - \tau_j(A)). \]

Assuming a Poisson density for the number of false measurements due to clutter with spatial density \( \lambda \), the joint association probabilities at time step \( k \) are computed as,

\[ P(A | Z_k, Z_{k-1}, \ldots) = \frac{1}{c} \prod_j \left( \lambda^{-1} f_{t_j}(z^j_k) \right)^{\tau_j} \prod_t \left( P_D^t \right)^{\delta_t} \left( 1 - P_D^t \right)^{1-\delta_t}, \]

where \( c \) is a normalization constant and,

\[ f_{t_j}(z^j_k) = \mathcal{N}(z^j_k | \hat{z}^j_{k|k-1}, S^j_{k}), \]
where $\hat{z}_{k|k-1}^{t_j}$ is the predicted measurement for target $t_j$ with the associated innovation covariance $S_k^{t_j}$.

If we assume that the states of the targets conditioned on the past observations are mutually independent, we can decouple the update step of the filtering algorithm for each target. In this case, the marginal association probabilities are required which can be obtained from the joint probabilities by summing over all the joint events in which the marginal event of interest occurs,

$$\beta_{jt} := P(A_{jt} | Z_k, Z_{k-1}, \ldots) = \sum_{A: A_{jt} \in A} P(A | Z_k, Z_{k-1}, \ldots)$$

The update step for each of the targets now proceeds exactly as in the PDAF.

### A.2 Multi Hypothesis Tracking

Multi hypothesis tracking (MHT) uses a deferred decision logic\[10\] in which alternative data association hypotheses are formed whenever there is an uncertainty in the observation-to-target association. Rather than choosing the best association as in the nearest-neighbor based algorithms or combining all the possible associations as in the JPDA method, MHT keeps track of the alternative hypothesis and propagates them in the future with the anticipation that further observations would help in resolving the ambiguity. For each hypothesis, a filter such as the KF, EKF, or UKF is used to update the target state using the associated measurement. Thus, compared to the JPDAF approach, MHT is able to use information from measurements at multiple time steps in order to find the best data association leading to better performance albeit at the cost of higher computation and memory since it needs to maintain and propagate the multiple hypothesis. Also, unlike JPDAF where the number of targets was assumed to be known and fixed, MHT methods can handle addition and removal of targets at run time.

There are two types of MHT, the hypothesis-oriented MHT (HOMHT) \[83\] and track-oriented MHT (TOMHT) \[55\]. Within TOMHT, there are tree based \[55\] and non-tree based methods.
Figure A.1: An example showing the predicted measurements, $\hat{z}_1$ and $\hat{z}_2$, from two targets and three observations, $z_1$, $z_2$, and $z_3$, with ellipsoids representing the measurement validation gates.

We compare the HOMHT with the tree based TOMHT but the arguments hold for the non-tree based TOMHT as well. The HOMHT keeps track of global hypotheses between time steps whereas the tree based TOMHT only maintains a tree for each target, each containing tracks which are incompatible with each other.

Consider the situation shown in Figure A.1 with two targets and three measurements. $\hat{z}_1$ and $\hat{z}_2$ are the predicted measurements for the two targets while $z_1$, $z_2$, and $z_3$ are the actual measurements. The ellipsoids show the measurement validation gates for the respective targets. Thus, all three measurements can be associated with target 1 while only $z_2$ and $z_3$ can be associated with target 2. Figure A.2 shows the TOMHT target trees generated by these validated measurements. Note that in addition to the validated measurements, we have two missed detection hypotheses for the known targets and three new targets hypotheses in the set of target trees. Note that here we assumed that the unassigned measurements are due to new targets instead of clutter. The global hypotheses generated by the HOMHT for the same example is shown in Table A.1. It is just a coincidence that the number of track hypotheses from TOMHT and global hypotheses from HOMHT is the same for this scenario.

In any MHT algorithm, the number of association hypotheses can grow exponentially as the measurements at each time step are processed sequentially. Thus MHT algorithms use a number of techniques such as clustering, gating, N-scan pruning, and track-score based pruning to limit this exponential growth [110]. A strong argument can be made
Figure A.2: TOMHT target trees generated by the example shown in Figure A.1. $\emptyset$ represents that none of the measurements correspond to that track. $T_3$, $T_4$ and $T_5$ are potential new targets.

Table A.1: The possible HOMHT global hypotheses generated from the example shown in Figure A.1. $\emptyset$ represents that the target is not associated with any measurement. $T_3$, $T_4$ and $T_5$ are potential new targets (see Figure A.2).

<table>
<thead>
<tr>
<th>Global hypothesis</th>
<th>Hypothesis structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$T_1 - z_1, T_2 - z_2, T_5 - z_3$</td>
</tr>
<tr>
<td>$G_2$</td>
<td>$T_1 - z_1, T_2 - z_3, T_4 - z_2$</td>
</tr>
<tr>
<td>$G_3$</td>
<td>$T_1 - z_1, T_2 - \emptyset, T_4 - z_2, T_5 - z_3$</td>
</tr>
<tr>
<td>$G_4$</td>
<td>$T_1 - z_2, T_2 - z_3, T_3 - z_1$</td>
</tr>
<tr>
<td>$G_5$</td>
<td>$T_1 - z_2, T_2 - \emptyset, T_3 - z_1, T_5 - z_3$</td>
</tr>
<tr>
<td>$G_6$</td>
<td>$T_1 - z_3, T_2 - z_3, T_3 - z_1$</td>
</tr>
<tr>
<td>$G_7$</td>
<td>$T_1 - z_3, T_2 - \emptyset, T_3 - z_1, T_4 - z_2$</td>
</tr>
<tr>
<td>$G_8$</td>
<td>$T_1 - \emptyset, T_2 - z_2, T_3 - z_1, T_5 - z_3$</td>
</tr>
<tr>
<td>$G_9$</td>
<td>$T_1 - \emptyset, T_2 - z_3, T_3 - z_1, T_4 - z_2$</td>
</tr>
<tr>
<td>$G_{10}$</td>
<td>$T_1 - \emptyset, T_2 - \emptyset, T_3 - z_1, T_4 - z_2, T_5 - z_3$</td>
</tr>
</tbody>
</table>
in favor of the TOMHT compared to HOMHT since the combinatorics of the hypotheses formation are such that there are typically many more hypotheses formed than tracks. As the number of targets grows, especially for difficult scenarios, there may be several thousand comparable global hypotheses from several hundred tracks in the target trees [10]. Maintaining these thousands of hypotheses and expanding them upon a new measurement can be computationally prohibitive while maintaining and expanding the several hundred tracks into new hypotheses has been demonstrated to be quite fast [88].

A.2.1 Track Scoring

The evaluations of the track hypotheses requires a probabilistic expression that incorporates all the information that is available, such as the false measurement density, the prior probability of target presence and the detection sequences. A likelihood ratio (LR) is the preferred method for scoring the tracks [3]. The likelihood ratio is defined as the ratio of the probabilities of the true target hypothesis and false measurement hypothesis and is computed as [10],

\[ LR := \frac{P_T}{P_F} = \frac{P(z | H_1) P_0(H_1)}{P(z | H_0) P_0(H_0)}, \]

where \( H_1 \) and \( H_0 \) are the true measurement and false measurement hypotheses respectively, \( z \) is the measurement, \( P(z | H_i) \) is the PDF evaluated at the current measurement with the assumption that \( H_i \) is correct and \( P_0(H_i) \) is the prior probability of \( H_i \).

For computational reasons, the logarithm of the likelihood ratio (LLR) is commonly used. Assuming that the measurements at each different time step are independent, this allows the LLR at time step \( k \) to be easily computed from the LLR at time step \( k - 1 \),

\[ \text{LLR}_k = \text{LLR}_{k-1} + \text{LΔLR}_k \]

Using this definition, the LΔLR for a true measurement event at time step \( k \) is given by,

\[ \text{LΔLR}_{\text{true}} = \log \left( \frac{P_D P(z_k | Z_k, Z_{k-1}, \ldots)}{\lambda_F} \right), \]
where $P_D$ is the target detection probability, $z_k$ is the measurement, $Z_i$ is the set of measurements received at time step $i$ and $\lambda_F$ denotes the spatial density of false measurements. We assume that the number of false measurements in the measurement is space is Poisson distributed. The $L\Delta LR$ for a missed detection is given by,

$$L\Delta LR_{\text{miss}} = \log(1 - P_D P_G),$$

where $P_G$ is the gate probability.

For a new track, we initialize the LLR as,

$$LLR_{\text{new}} = \log \left( \frac{\lambda_N}{\lambda_F} \right),$$

where $\lambda_N$ is the spatial density of new targets. Note that we assume that the number of new targets appearing in the measurement space is also Poisson distributed.

### A.2.2 Best Global Hypothesis and Pruning

As mentioned before, the number of track hypotheses in TOMHT can grow exponentially as new measurements arrive. This can lead to computational problems specially when a large number of targets is present. In order to keep the computation complexity within reasonable bounds, the tree based TOMHT requires pruning to only keep the tracks with high scores. The most common pruning methods are track score based pruning and N-scan pruning. Track score based pruning simply removes the tracks which have a score below a certain threshold. N-scan pruning limits the maximum depth of the trees by removing branches from the tree which are not part of the best global hypothesis (BGH).

Consider the example shown in Figure A.3. At time step $k - 1$, we have two resolved tracks $T_1$ and $T_2$, and at time steps $k$ and $k + 1$ we receive measurements sets $\{z_1, z_2\}$ and $\{z_3\}$ respectively. Following the process of expanding the trees in TOMHT, ten track hypotheses $\{h_1, \ldots, h_{10}\}$ are generated at time step $k + 1$. We want to select the set of track hypotheses from these which maximizes the sum of their LLR but we also have to satisfy some constraints.
in this process. For a feasible solution, only one of the track hypotheses for each target tree can be selected. In addition, at each time step, each measurement can only be assigned to one of the tracks. In order to compute the BGH, we construct a binary programming problem as follows. Let $u$ and $x$ be 10-dimensional vectors, where $u$ contains the LLR for each of the tracks and $x$ is a vector of binary values, i.e. $x_i \in \{0, 1\}$. The elements of $x$ represent whether the corresponding track hypothesis is part of the BGH. We construct the matrix

$$A = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 & h_{10} \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

in order to encode the constraints mentioned above. Let $b$ be a vector consisting of all ones with length equal to the number of rows of $A$. Using these, the binary programming problem
is,

\[
\arg \max_a \ u \cdot a \\
\text{Subject to } Aa = b
\]

The Lagrangian relaxation algorithm [79] and approximate linear programming [98], can be used to solve this problem.

Now, suppose that tracks \(h_2\) and \(h_5\) are included in the BGH and we choose the \(N\) in N-scan as 1. Thus we move one step back from \(k + 1\) to \(k\) and remove all the branches in trees for \(T_1\) and \(T_2\) that are not part of the BGH. Thus, tracks \(h_1, h_4, h_6, h_7, h_8\) and \(h_9\) are deleted by N-scan pruning.

### A.3 Random Finite Sets

Random finite sets (RFS) are a new and emerging approach for multi-target tracking. The distinguishing feature of RFS is that instead of first solving the data association problem and then calculating the target state estimates, the RFS formulation directly seeks both optimal and suboptimal estimates of the multi-target state [110]. In the RFS formulation, the collection of individual targets is treated as a *set-valued state*, and the collection of individual observations is treated as a *set-valued observation*. Using Finite Set Statistics, a multi-target Bayes filter can be derived that computes the multi-target posterior density by propagating and updating the current multi-target state[33]. The main issue with this multi-target Bayes filter is that it is computationally intractable due to the combinatorial nature of the multi-target densities and multiple integrals on the infinite dimensional multi-target state space, hence approximations are required to apply it to practical problems.

An approximation for the Bayes multi-target filter was proposed in [66] called the Probability Hypothesis Density (PHD) filter which propagates the first order statistical moment of the RFS instead of the complete multi-target posterior density. The PHD, also known as the intensity function, is a function whose integral over any region of state space is equal to
the expected number of targets in that region. The peaks of the PHD function occur at points with a high expectation of the target being present and can be used to generate the multi-target state estimate. The PHD filter in general requires evaluation of integrals with no closed form solutions, hence Monte Carlo techniques have been proposed \cite{111, 116} to propagate the PHD in time and clustering techniques are used to extract the state estimates. The main drawbacks with these approaches are that a large number of samples (or particles) are required to represent the intensity function and the clustering techniques are unreliable for accurately extracting the state estimates \cite{109}.

In order to solve these issues for the case of linear Gaussian target dynamics and Gaussian density for new targets appearing, an analytic solution was proposed in \cite{109} called the Gaussian Mixture PHD (GM-PHD) filter. It was shown that when the initial prior intensity is a Gaussian mixture, the posterior intensity at any subsequent time step is also a Gaussian mixture and closed-form expressions for the weights, means, and covariances of the mixture components were derived. Another advantage of the GM-PHD filter is that it allows state estimates to be extracted from the posterior intensity in a much more efficient and reliable manner than clustering in the particle-based approaches. The number of components in the GM-PHD filter keeps on growing at every step, hence mixture reduction techniques such as pruning negligible components and merging similar components are used to manage this growth \cite{109}.

One of the problems with the PHD filter is that it only estimates the intensity function which provides no information about the identity of the targets. This is limiting in cases when we want to keep track of the individual targets. This was solved by the Generalized Labeled Multi-Bernoulli (GLMB) Filter \cite{107, 108} which can track individual targets by appending a label to each target’s state. An approximation to the GLMB filter, known as the LMB filter \cite{84}, has been shown to be able to track thousands of targets simultaneously in clutter while running on a laptop computer \cite{112}.
Bibliography


