2016-12-01

Autonomous Goal-Based Mapping and Navigation Using a Ground Robot

Jeffrey L. Ferrin
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/6190

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Autonomous Goal-Based Mapping and Navigation Using a Ground Robot

Jeffrey L. Ferrin

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Timothy W. McLain, Chair
Neal K. Bangerter
Randal W. Beard
Steven K. Charles
Mark B. Colton

Department of Mechanical Engineering
Brigham Young University

Copyright © 2016 Jeffrey L. Ferrin
All Rights Reserved
ABSTRACT

Autonomous Goal-Based Mapping and Navigation Using a Ground Robot

Jeffrey L. Ferrin
Department of Mechanical Engineering, BYU
Doctor of Philosophy

Ground robotic vehicles are used in many different applications. Many of these uses include tele-operation of the robot. This allows the robot to be deployed in locations that are too difficult or are unsafe for human access. The ability of a ground robot to autonomously navigate to a desired location without a-priori map information and without using GPS would allow robotic vehicles to be used in many of these situations and would free the operator to focus on other more important tasks. The purpose of this research is to develop algorithms that enable a ground robot to autonomously navigate to a user-selected location. The goal is selected from a video feed from the robot and the robot drives to the goal location while avoiding obstacles. The method uses a monocular camera for measuring the locations of the goal and landmarks. The method is validated in simulation and through experiments on an iRobot Packbot platform.

A novel goal-based robocentric mapping algorithm is derived in Chapter 3. This map is created using an extended Kalman filter (EKF) by tracking the position of the goal along with other available landmarks surrounding the robot as it drives towards the goal. The mapping is robocentric, meaning that the map is a local map created in the robot-body frame. A unique state definition of the goal states and additional landmarks is presented that improves the estimate of the goal location. An improved 3D model is derived and used to allow the robot to drive on non-flat terrain while calculating the position of the goal and other landmarks. The observability and consistency of the proposed method are shown in Chapter 4.

The visual tracking algorithm is explained in Chapter 5. This tracker is used with the EKF to improve tracking performance and to allow the objects to be tracked even after leaving the camera field of view for significant periods of time. This problem presents a difficult challenge for visual tracking because of the drastic change in size of the goal object as the robot approaches the goal. The tracking method is validated through experiments in real-world scenarios.

The method of planning and control is derived in Chapter 6. A Model Predictive Control (MPC) formulation is designed that explicitly handles the sensor constraints of a monocular camera that is rigidly mounted to the vehicle. The MPC uses an observability-based cost function to drive the robot along a path that minimizes the position error of the goal in the robot-body frame. The MPC algorithm also avoids obstacles while driving to the goal. The conditions are explained that guarantee the robot will arrive within some specified distance of the goal.

The entire system is implemented on an iRobot Packbot and experiments are conducted and presented in Chapter 7. The methods described in this work are shown to work on actual hardware allowing the robot to arrive at a user-selected goal in real-world scenarios.

Keywords: GPS-denied localization, vision-based navigation, MPC, robocentric mapping
ACKNOWLEDGMENTS

I would first like to thank my wife, Camille for her never-ending love, support, and optimism during this time. I could not have done this without her. I am grateful to her for listening to me and encouraging me along this entire path. I also want to thank my children, Tanner, Lydia, Tessa, and Ava for their love and patience with me. Being with them brings me much joy, strength, and hope. I also thank my extended family members for their constant encouragement and for their prayers.

I am very grateful to have worked with Dr. McLain. I thank him for his technical guidance and his always steady leadership. He has helped in many ways and I could feel his confidence in me as I met with him. I always left his office feeling more confident than when I entered. I also thank Dr. Beard for his help. It was fun to discuss technical ideas and see his excitement for different ways to solve difficult problems.

I am very grateful to Mel Torrie at Autonomous Solutions, Inc. (ASI). Mel is a great friend and I am grateful for his patience with me and providing me the opportunity to work on this project. There are many people at ASI who have helped me and I need to mention some in particular. I thank Joel Alberts and Matt Berkemeier for the many discussions and sharing their thoughts and ideas. Jeremy Adams was extremely helpful in overcoming some software/hardware road blocks that allowed me to get the final implementation working on the robot. I also thank John Petersen for his friendship and brainstorming ideas with me.

Finally, I thank the great members of the MAGICC lab at BYU. I was only in the lab for a short time but it was a great experience. I especially thank Brandon Cannon, Robert Leishman, Joe Nichols and David Wheeler for their help and for making it fun. I also thank Raj Sharma for his help while a member of the MAGICC lab and also for taking time to meet and discuss certain parts of this work while a professor at Utah State University.
# TABLE OF CONTENTS

LIST OF TABLES ......................................................... vii

LIST OF FIGURES ....................................................... viii

Chapter 1  Introduction .................................................. 1
  1.1 Motivation .......................................................... 1
  1.2 Problem Description ............................................. 2
    1.2.1 Goal Localization ............................................. 3
    1.2.2 Path Planning ............................................... 3
    1.2.3 Vehicle and Sensor Constraints ............................. 4
    1.2.4 Visual Tracking .............................................. 4
    1.2.5 Reacquiring Landmarks ..................................... 4
  1.3 Literature Review ................................................ 4
    1.3.1 Visual Servoing .............................................. 5
    1.3.2 SLAM .......................................................... 5
    1.3.3 SLAM Environments ......................................... 7
    1.3.4 Path Planning ............................................... 9
  1.4 Summary of Contributions ....................................... 12
    1.4.1 EKF Mapping ................................................ 12
    1.4.2 Visual Tracking .............................................. 12
    1.4.3 Model Predictive Control ................................ 13
    1.4.4 Experimental Results ...................................... 13
  1.5 Document Layout ................................................. 13

Chapter 2  Experimental Setup ........................................ 15
  2.1 Robot Platform ................................................... 15
  2.2 Sensors ............................................................ 16
    2.2.1 Camera ....................................................... 16
    2.2.2 Track Encoders ............................................. 17
    2.2.3 LiDAR ......................................................... 17
    2.2.4 Inertial Measurement Unit ................................ 17
    2.2.5 GPS .......................................................... 18
  2.3 Computer .......................................................... 19
  2.4 Test Facility ..................................................... 19

Chapter 3  Mapping ........................................................ 20
  3.1 Problem Description ............................................. 20
  3.2 Literature Review ................................................. 22
  3.3 Extended Kalman Filter for Robocentric Mapping .............. 23
    3.3.1 Coordinate Frame Definition ............................... 24
    3.3.2 Discrete-Time Extended Kalman Filter Formulation ..... 24
    3.3.3 State Definitions .......................................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>State Motion Model</td>
<td>30</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Vehicle Model</td>
<td>31</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Goal Model</td>
<td>32</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Landmark Model</td>
<td>35</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Conversion to $\rho$</td>
<td>37</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Process Noise</td>
<td>39</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Motion Model Jacobian</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>Measurement Equations</td>
<td>46</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Camera Coordinate Frames</td>
<td>46</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Camera Model</td>
<td>48</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Measurement Jacobian</td>
<td>51</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Chapter 4 Mapping Analysis</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Observability</td>
<td>55</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Lie Derivatives</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Consistency of EKF</td>
<td>63</td>
</tr>
<tr>
<td>4.3.1</td>
<td>2D Simulation Results</td>
<td>65</td>
</tr>
<tr>
<td>4.3.2</td>
<td>3D Simulation Results</td>
<td>71</td>
</tr>
<tr>
<td>4.4</td>
<td>Algorithm Performance</td>
<td>77</td>
</tr>
<tr>
<td>4.4.1</td>
<td>2D Algorithm Comparison</td>
<td>77</td>
</tr>
<tr>
<td>4.4.2</td>
<td>3D Algorithm Performance</td>
<td>80</td>
</tr>
<tr>
<td>4.5</td>
<td>Summary</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Chapter 5 Visual Tracking</td>
<td>85</td>
</tr>
<tr>
<td>5.1</td>
<td>Problem Description</td>
<td>85</td>
</tr>
<tr>
<td>5.2</td>
<td>Goal Tracking</td>
<td>85</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Goal Selection</td>
<td>86</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Goal Tracking</td>
<td>87</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Goal Reacquisition</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>Landmark Tracking</td>
<td>99</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Landmark Selection</td>
<td>100</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Landmark Tracking</td>
<td>101</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Landmark Reacquisition</td>
<td>102</td>
</tr>
<tr>
<td>5.4</td>
<td>Summary</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Chapter 6 Nonlinear Model Predictive Control</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>105</td>
</tr>
<tr>
<td>6.2</td>
<td>Literature Review</td>
<td>105</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Robot Control</td>
<td>106</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Obstacle Avoidance</td>
<td>106</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Sensor Constraints</td>
<td>107</td>
</tr>
<tr>
<td>6.3</td>
<td>General Problem Formulation</td>
<td>107</td>
</tr>
<tr>
<td>6.4</td>
<td>Cost Function</td>
<td>110</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

4.1 Parameters used in the 2D Monte Carlo simulations. . . . . . . . . . . . . . . . . . 65  
4.2 Parameters used in the 3D Monte Carlo simulations. . . . . . . . . . . . . . . . . . 71  
6.1 Parameters used in the Matlab MPC simulation. . . . . . . . . . . . . . . . . . . . 138  
7.1 Parameters used for the flat ground experiment . . . . . . . . . . . . . . . . . . . 159  
7.2 Parameters used for the rough terrain experiment . . . . . . . . . . . . . . . . . . 169
# LIST OF FIGURES

2.1 iRobot Packbot 400. ................................................................. 15
2.2 Xtion PRO LIVE sensor used for the monocular camera image. ........ 16
2.3 Incremental encoder and Hokuyo LiDAR. .................................... 17
2.4 KVH 1725 IMU mounted on the Packbot. ................................... 18

3.1 Example of triangulation using a monocular camera. .................... 21
3.2 $\mathcal{F}^b$ definition shown on robot. ..................................... 25
3.3 State definitions for the goal point in $\mathcal{F}^b$. ............................ 28
3.4 Single landmark state definitions. ........................................... 29
3.5 Intermediate landmark angle definitions, shown for a single landmark. 36

4.1 Path overview of 2D path with no goal loss. ............................... 67
4.2 Path of goal point in the image. ............................................. 68
4.3 Average NEES value for 2D path with no goal loss in image. ........ 68
4.4 Path overview of the 2D path that causes the goal to leave the camera field of view. 69
4.5 Path of goal point in the image. ............................................. 70
4.6 Average NEES value for 2D path with goal loss in the image. ......... 70
4.7 Path overview of the 3D path with no goal loss in the image. .......... 72
4.8 Angle states error for the 3D path with no goal loss in the image. .... 73
4.9 Path of the goal point in the image for 3D path. .......................... 73
4.10 Average NEES value for 3D path with no goal loss in image. ........ 74
4.11 Path overview of the 3D path with the goal leaving the image. ....... 75
4.12 Angle states error for 3D path with the goal leaving the image. ....... 76
4.13 Path of the goal point in the image for the 3D path. ..................... 76
4.14 Average NEES value for 3D path with the goal leaving the image. .... 77
4.15 Goal state error between two methods for 2D path without loss of the goal. 79
4.16 Position error between two methods for 2D path without loss of the goal. 80
4.17 Goal state error between two methods for 2D path with goal loss. ... 81
4.18 Position error between two methods for 2D path with goal loss. ...... 82
4.19 Goal states for 3D path with no goal loss. ............................... 83
4.20 Goal states for 3D path with goal loss. .................................... 84

5.1 Template definition in image. ............................................... 87
5.2 Search region definition. ...................................................... 89
5.3 Template matching. ............................................................ 90
5.4 Image sequence for template update. ..................................... 92
5.5 HSV template offset. .......................................................... 93
5.6 Image template sequence. .................................................... 94
5.7 The HSV images corresponding to the template images shown in Figure 5.6. 94
5.8 First experimental image sequence overview with templates and HSV images. 95
5.9 Second experimental image sequence overview with templates and HSV images. 96
5.10 Image template offset. ....................................................... 98
5.11 Images showing goal object leaving image. ............................. 99
7.15  Path of the robot during the hill experiment. ................................. 170
7.16  Zoomed-in view of the robot path around the first obstacle on the hill test. . 171
7.17  Zoomed-in view of path of robot around second obstacles. ....................... 172
7.18  Elevation profile and roll and pitch of the vehicle. ............................... 172
7.19  Plot of the state $\rho$ for the goal for the experiment. ............................. 173
7.20  Plot of the estimated range and the actual range. ................................. 174
7.21  Plot of the range error vs. time for the entire run. ............................... 175
7.22  Plot of the angles $\alpha$ and $\beta$ for the goal. ................................. 175
7.23  Plot of the desired and actual angular rates. ................................. 176
7.24  Plots of the angular rates $p$ and $q$. ................................................. 176
7.25  Linear velocity of the robot. ................................................................. 177
7.26  Camera measurement locations of the goal in the image frame for the entire run. . . 177
7.27  First image sequence from rough-terrain experiment. ......................... 178
7.28  Second image sequence from rough-terrain experiment. ......................... 179
CHAPTER 1. INTRODUCTION

1.1 Motivation

The advance of technology in the field of robotics has allowed robots to be used in many different applications. These robots include vehicles that operate on land, in water, in air, and in space. Land robots can range in size from small robots that fit in the palm of the hand up to large mining haul trucks that can carry over 170,000 lb. There are both civil and military applications for these robots. Some examples of robotic applications include military robots for explosive ordnance disposal, unmanned aerial vehicles (UAV) used for surveillance missions, autonomous tractors for agricultural use, and unmanned mining vehicles used for higher efficiency and mine safety.

Many of the robots in use today are dependent on the Global Positioning System (GPS) for position, heading and velocity information. This information is crucial to the robot being able to complete the desired task. Without this information these robots would not be able to stay on a desired path or arrive at the desired location. This dependency on GPS is a problem when the vehicle is in an area where the GPS update is not available, when the positional accuracy of the GPS is too large for the application, or in GPS denied environments such as indoors, underground, canyons (urban or natural) and locations where the GPS signal is jammed.

Of course, there are many robotic applications that need to continue to function even when the GPS is not sending valid position information or when the accuracy of the GPS is degraded. For example, a robotic car that is driving down a road using only GPS for position feedback is completely reliant on the accuracy of the GPS. If there is drift in the GPS measurement, the vehicle will start to diverge from the desired location of the road. If the drift in the GPS becomes greater than half the width of the road then the vehicle could drive off the road. It is, therefore, desirable that the vehicle be able to navigate even when the GPS is not giving valid position data.

One way that robots are currently used without the dependence on GPS is under remote control. The operator controls the robot via a joystick or other operator control unit (OCU). The
operator can control the robot within a line-of-sight view or through a video feed from the robot. This allows the robot to be driven to a desired destination or perform a desired task without dependency on GPS.

In many robotic applications, the robot must be driven to a certain location to perform a given task, such as manipulating an object or surveillance from a selected point. A ground robot under remote control from a human operator can easily be driven to a desired destination and even when the operator is controlling the vehicle using a video feed with a limited field of view, it remains a trivial task. The operator can drive to a desired goal by monitoring the location of the goal in the video and steer the vehicle to avoid collisions with other objects and properly arrive at the desired goal location. It is possible for the operator to do this, but it comes at the cost of requiring the complete attention of the operator while the vehicle is moving toward the goal. As the environment becomes more difficult (e.g., more obstacles, obstruction of view to the goal, etc.), the operator must be more aware of the surroundings, therefore requiring more focus.

The ability for a ground robot to autonomously navigate to a selected goal is a behavior that would decrease the work-load of the human operator, allowing the operator to focus on other, more important tasks (e.g. monitoring a current dangerous situation, simultaneously controlling multiple robots). For this to be possible, the robot must be able to calculate in real-time, the position of the goal while driving towards the goal. To reach the goal safely, the robot must be able to track landmarks and features that could be several hundred meters away from the current robot location.

1.2 Problem Description

The problem addressed by this work is the task of autonomously navigating a ground robot to a selected goal destination in an unknown environment, without the use of GPS, and in the presence of obstacles. This work will focus on a ground robot using a monocular camera for landmark measurements. A LiDAR is used for near-vehicle (five meters) obstacle detection.

There are many contributing factors that make this problem difficult to achieve in practice on an actual robot platform. Some of the main factors addressed in this work include:

- Goal localization in the robot-body frame
- Calculating a path to the goal in the unknown environment and in the presence of obstacles

2
• Vehicle and sensor constraints

• Visual tracking of landmarks in camera images

• Reacquiring landmarks in the image

These items and how they contribute to the difficulty of the problem are discussed in the following sections.

1.2.1 Goal Localization

When the goal is first selected in the monocular camera image, there is no depth information. This means that the goal could be anywhere on an infinite ray extending from the center of the camera image through the goal to infinity, however, it is not assumed in this work that the goal will be visible over the entire path from the starting point of the robot to the goal. For this reason the position of the goal must be estimated so that when the goal does leave the image, it will be possible to find the goal at a later time when the robot can turn back towards it.

1.2.2 Path Planning

Once the goal is selected, a path must be generated that will drive the robot to the goal location. The difficulty of the path planning for this work is increased due to the fact that the environment is unknown. The path planning must be dynamic for two main reasons. First, there are obstacles in the environment but the locations of the obstacles are unknown and they are only known when they are close enough to the vehicle for the 3D sensor to see them. Second, the position of the goal with respect to the robot is being estimated and therefore is changing as more measurements are taken of the goal.

The goal of this work in the area of path planning is to plan and drive an optimal path to the goal location. The path will be optimized to minimize the uncertainty in the position of the goal with respect to the robot. As the robot travels to the goal, the location estimate of the goal will improve.
1.2.3 Vehicle and Sensor Constraints

As the robot drives toward the goal, the vehicle constraints and sensor constraints will limit what types of paths can be taken. The vehicle constraints that are considered include the maximum vehicle velocity, maximum turn rate, and the platform nonholonomic constraint. The sensor constraints that are addressed include the sensor field of view and the fact that the sensor is on a fixed mount to the robot.

1.2.4 Visual Tracking

The camera is rigidly mounted on a small ground robot. As the vehicle drives on uneven terrain the camera will be moving rapidly. This motion will make the visual tracking of landmarks difficult. Another difficult problem for the visual tracking is the significant change in size of the goal object. When first selected the goal could be very small in the image. This causes difficulty in the tracking because the size of the goal object changes significantly while driving towards it and the visual features also change.

1.2.5 Reacquiring Landmarks

This work also addresses the problem of reacquiring landmarks after they have left the field of view of the camera. The nonholonomic vehicle constraint coupled with the fixed sensor mount on the vehicle means that the sensor cannot always be pointing in the direction of the goal. For example, under certain situations the vehicle might have to drive around an obstacle and while doing so it might have to turn away from the goal such that the goal is no longer visible. This is undesirable because if the goal is not visible then no information is gained about the position of the goal. When this happens it is important that the visual tracker be able to reacquire the goal so the position estimate can improve.

1.3 Literature Review

This section will discuss the literature relevant to this problem. This will help show where the gaps are in the current literature for solving this problem.
1.3.1 Visual Servoing

This work assumes that the vehicle is not equipped with a GPS and that the only initial information about the goal is the location in the image where the goal was selected. There has been much work in the area of visual servoing [1–4]. Visual servoing allows the robot or the manipulator to arrive at an object by continually calculating a command that will keep the object in the video frame. The key difference between the work presented here with visual servoing is that in this work the robot is calculating the position goal with respect to the robot to allow for times when the goal will be unavailable in the video image. This will allow the robot to still drive to the goal while it is not seen in the image.

1.3.2 SLAM

SLAM is a widely researched topic in the field of robotics and extends across land [5, 6], air [7, 8] and water vehicles [9, 10]. Within each of these areas there are different types of SLAM based on different sensors. Currently, there exist SLAM algorithms using stereo vision [11, 12], laser [13], monocular camera [14, 15], structured light sensors [16], and sonar [17]. The different sensors are used to gain information about the surrounding environment. The information from the sensor is passed into the SLAM algorithm as a measurement. These measurements are then used to localize the robot and map the environment. It is the method in which this localization and mapping are performed which distinguishes the SLAM algorithm. Three main types of SLAM algorithms exist, these are extended Kalman Filter (EKF) SLAM, Particle Filter SLAM, and GraphSLAM.

EKF SLAM

The EKF slam algorithm is the most widely used slam algorithm [18–20]. This algorithm is an extension of a normal EKF used for vehicle states only. The EKF SLAM algorithm takes the vehicle state vector used for normal localization and appends landmark states as new landmarks are observed. This new state vector continues to grow as each new landmark is added into the state. As one can imagine, this state vector can grow very rapidly. There are some drawbacks of EKF SLAM as described in [19]; the computational complexity scales quadratically with the number of landmarks. It therefore becomes a problem if the robot is to traverse a large area that
will have a large amount of landmarks. The EKF formulation of SLAM is also fragile with respect to the data association of landmarks. This can lead to inconsistency and is especially difficult for loop closures. The EKF also linearizes the models used and this linearization can also cause inconsistencies.

**Particle Filter SLAM**

The most widely used particle-filter-based SLAM algorithm is the FastSLAM algorithm. This was developed by Montemerlo [21,22] to decrease the computational complexity of the problem when large amounts of landmarks are used. The FastSLAM algorithm scales logarithmically with the number of landmarks used. This increase in speed is possible by representing the map landmarks as independent variables [19]. This allows the trajectory of the robot alone to be represented by the weighted particles and the map landmarks are updated through a simple EKF (simple because they are independent) when they are observed, however, the FastSLAM algorithm is limited in producing a consistent map as described in [23]. Here the author notes that through re-sampling, particles are lost and with the loss of a particle, important map information is lost that is needed for map consistency.

**GraphSLAM**

The method of GraphSLAM uses a graph comprised of nodes and edges to store the information. The nodes of the graph are the robot poses at different locations while the edges in the graph represent nonlinear constraints that include measurements that link two consecutive poses (nodes) and also measurements to landmarks that are seen at certain poses [24]. The constraints are then used to solve for the map. The nonlinear constraints can be linearized and used in an information matrix in a way to reduce the complexity of the problem.

Some of the more recent work in the area of GraphSLAM has been by Konolige [25–28]. This work has focused on using a GraphSLAM framework to marginalize out (remove) information from the graph to make it possible for the robot to map large runs (up to ten km) [25]. The idea is that the system is reduced by using stored frames from the stereo camera instead of storing features. These frames then store the information needed to solve the full SLAM problem of computing the
trajectory and the map of the environment through which the robot traveled. The graph is even optimized to keep only the needed frames to reduce the system even more. The used frames are called skeleton frames. The skeleton frames are then used by a place recognition algorithm for loop closure.

1.3.3 SLAM Environments

The SLAM research has focused on different types of environments which can be characterized into four main groups. The applicable research in these different environments will be presented. These environments are:

- Indoor
- Structured outdoor urban
- Unstructured outdoor
- Unstructured outdoor with rough terrain

Indoor Environment

Much of the SLAM work has focused on indoor environments. For ground vehicles these indoor environments simplify the problem because it can be a 2D problem where the robot only has to calculate the positions in \((x, y)\) and heading. Impressive results have been shown in indoor environments. Takezawa, Herath, and Dissanayake [29] used stereo vision to navigate indoors on a planar surface using SLAM. Artificial landmarks were placed in the area of the desired mapping location for aiding in localization. Nguyen, You, and Oh [5] demonstrate impressive mapping of a building with a monocular SLAM algorithm. Sola [30] has shown indoor mapping using features from both a stereo sensor and monocular sensor together. This enables the use of points that are far away and also closer points that give 3D information. Monocular SLAM has also been researched for indoor environments [14]. Zhang et al. [31] showed indoor mapping with limitations on sensing range. They use short-range wall sensors to both follow a wall and map the building. There has been much success in indoor SLAM, however, SLAM in outdoor environments is more difficult and remains an open area of research [6].
Structured Outdoor Urban Environment

A structured urban environment [32, 33] typically has flat surfaces where the robot travels and is rich in visual features. In vision-based SLAM algorithms (stereo or monocular), this feature richness can decrease the complexity of the outdoor problem. The flat surfaces where the robot travels also creates a simpler problem because the robot only estimates a 2D position and heading.

Monocular SLAM has been demonstrated in an outdoor structured urban environment in [34]. The camera is carried along a line between rows of buildings that give many visual features used for the SLAM algorithm. The main contribution from this work is combining visual odometry with monocular SLAM to create a consistent map with less map landmarks while using many features in the visual odometry.

A full six DOF SLAM algorithm using only a stereo camera carried by hand is presented in [35]. The results show impressive mapping of a public square where no IMU or other information was used. The novel part of this work is that points that are too far from the stereo sensor are used for bearing only information until those points are close enough to the sensor to obtain 3D information. This work does not solve the problem of navigating in an unstructured environment where there are not as many features. The work states that it assumes smooth motion which is why the stereo camera is carried in hand. This is not the case of a sensor that is mounted on a robot. A more complex motion model of the camera must be used to properly use SLAM on a ground robot.

Unstructured Outdoor Environment

An unstructured environment does not have the buildings and roads for a nice structured scene. In this environment it is more difficult for the robot to find features in the scene that can be used to track for landmarks in the map. There has been some success in the area of unstructured outdoor environments recently. Agrawal et al. [6, 11, 36] have done nice work in this area with a small ground robot using stereo vision. This work deals with autonomous navigation of the robot with the goal of having the robot enter an unknown area, map it, and reach a goal point. The robot is also equipped with a GPS unit. The GPS unit is low quality but is used together with visual odometry to create a globally consistent map. The system also uses a global gradient planner to safely navigate through the outdoor environment. Some short comings of this work with regards
to the proposed research is the dependency on GPS to reach a globally referenced goal point. The GPS is aided with visual odometry but it is still necessary to reach the desired goal point. The work also neglects the interaction between the terrain and the vehicle. This assumption works well for the environments that were tested because of the relatively smooth terrain in comparison to the size of the robot. This kept the attitude of the robot level so that the vehicle roll and pitch were not issues.

**Unstructured Outdoor Environment With Rough Terrain**

There is less work dealing with SLAM in rough terrain. The definition of rough terrain is dependent on the platform that is traversing the terrain. Rough terrain in this dissertation is terrain that causes large enough changes in the vehicle attitude that these changes cannot be neglected for proper performance. This type of terrain will cause the movement of the vehicle to be abrupt which increases the complexity of the problem. There has been work in this environment dealing with visual odometry by Konolige [37]. This work describes how to accurately track selected points and compute the translation and rotation in rough terrain.

Work in [26] talks about performing GraphSLAM in rough terrain. Unfortunately, only a qualitative description of the rough terrain is given. The paper mentions abrupt changes in roll and pitch and abrupt vertical movement but the amount is not specified. The results of the work show that the vehicle was able to traverse a five km long path. With visual odometry alone, the result was close to 100 m of error and using the GraphSLAM approach the error was reduced to close to ten m.

**1.3.4 Path Planning**

Much of the literature in the SLAM research community does not deal with the actual navigation of the vehicle through the surrounding environment. It is generally assumed that the vehicle passes through the environment without having to decide where to go. This is done by carrying the sensor in hand, manually driving the vehicle on which the sensor is mounted or by tele-operating the vehicle through the environment. If the end goal of the SLAM process is a map then the method of navigation is not important and can be assumed to be trivial, however, if the
robot must arrive at a destination point without a pre-planned path, there must be a way to plan an achievable path through the environment. This path should allow the robot to arrive at the desired location while avoiding obstacles.

One method of path planning that is decoupled from the SLAM algorithm is proposed in [38]. The goal of this planner is to autonomously explore the surrounding environment to build a map. The path planner works by finding the obstacles near the vehicle and uses a vector force from each obstacle that will shape the path to keep the vehicle away from that obstacle. The path planner then sums all of the forces from the obstacles along with a goal destination to compute the desired path. This method has no guarantee that the robot will not get stuck or that it will cover the desired area. Another method that is decoupled from the SLAM algorithm is in [6]. A gradient based planner is used for the global planning. The planner requires a cost map that is used to calculate the optimal path through the cost map. The goal of this planner is to arrive at a desired location. Given that the desired location is most likely in an area of the environment that has not been mapped, the planner operates on sub maps of the environment that allow for incremental progress to the desired location. This method of planning is used in conjunction with a GPS unit allows the planner to know the location of the goal.

The problem with a decoupled path planner is that the planner is not taking into consideration the uncertainty in the localization and mapping of the robot. This could lead to paths that are optimal in time (getting the robot to the desired location fast) but the uncertainty of the robot position might grow beyond the feasible amount. Path planning that uses the localization and mapping information in the planning algorithm is termed Active SLAM in the research literature.

Methods of Active SLAM are explained in [39–41]. The goal of the path planning is to ensure complete area coverage while maintaining the highest accuracy possible in computing the map of the environment. In [39] model predictive control is used with an attractor. The model predictive control part of the planner is used for the small local motion around where the vehicle currently is. The attractor is then used for large vehicle motions. The placement of the attractor is based on whether the vehicle needs to localize more accurately, improve the map, or explore. The work in [40] is similar but is used for indoor environments to map buildings. This method extracts lines from the indoor environment to use for the mapping and planning process.
The work in [10,42] tries to reduce the information entropy (uncertainty) of the map by planning paths that increase the information gain over the traveled segment. The overarching goal is to explore the environment and to revisit past positions so that loop closures can be used to increase the accuracy of the map. Results are presented of a ground robot driving in an underground tunnel. Active SLAM is used to plan a path through the tunnel and to plan a loop closure path which it does successfully.

Calleja, Cetto, Davison, et al. [43] propose a manner to command the motion of a monocular camera performing SLAM. The output of the planner is a discrete set of outputs that tell the sensor whether to go up, go down, go left or go right. The output is calculated to maximize the information used which decreases the uncertainty of the localization and mapping. Of course, this method alone will not allow the robot to arrive at a goal point but it could be combined with another cost function that will drive the robot towards the goal point while maximizing the information gain.

The goal of Active SLAM is to plan a path in a way that minimizes the uncertainty in the localization and mapping estimates. The Active SLAM planning methods discussed above have achieved important results that allow the robot to drive through the environment safely and also send the robot to areas in the environment where the information gained will be maximized. These methods do not, however, address the issue of planning an optimal path to a desired location. The measure of optimality is interesting in the case of driving the robot to a goal location. Ideally, the robot would be able to arrive at the goal location with 100% accuracy and do so in the shortest amount of time. In this case there are two goals that at times could be conflicting. The first being the amount of error in the position of the robot with respect to the goal point and the second being the amount of time taken by the robot to arrive at the desired location. The challenge of the problem is to get the lowest amount of error in the least amount of time. This has not been addressed in the literature.

The ability to arrive at a goal point when this point is not given as a globally referenced goal point also remains a challenge. When a user selects an object in the current image from the robot, the point might be out of the 3D sensor range which means that a simple \((x,y,z)\) coordinate of the point is not known. The vehicle now has to plan a path to the goal location while estimating the actual location of the goal point.
1.4 Summary of Contributions

The key contributions of this work deal with the EKF mapping, visual tracking, and the model predictive control. These algorithms were designed to drive a ground robot to a selected goal in an unknown environment with rough terrain and obstacles.

1.4.1 EKF Mapping

In this work, an extended Kalman filter was designed and implemented to calculate the positions of the goal and additional landmarks in the robot-body frame. There are two contributions dealing with the design and implementation of the EKF mapping algorithm.

First, a novel algorithm was derived for goal-based robocentric mapping. In this formulation, the goal is an anchor state and all additional landmarks in the local map are referenced to the goal. This is shown to improve the position estimate of the goal in the robot-body frame. This improved position estimate is key to the success of the robot arriving at the goal.

Second, a 3D formulation was derived that estimates the positions of additional landmarks relative to the goal position. This enhances the ability of the robot to drive on rough terrain and maintain proper position estimates of the goal and the additional landmarks. This is shown to significantly improve convergence of the goal estimates and even allows the vehicle to accurately track the goal state when it is not in the field of view.

1.4.2 Visual Tracking

A visual tracker was developed that allows for the goal and additional landmarks to be tracked under 3D motion. This tracker utilizes the state estimates from the EKF to aid in reacquiring landmarks after they have been out of the camera field of view. The tracker was designed to handle large changes in size of the objects that it is tracking. This was a key development to allow the robot to track the goal and additional landmarks as the robot approaches the goal.
1.4.3 Model Predictive Control

A model predictive control method was designed to drive the robot in an unknown environment and avoid obstacles while driving towards the goal. There are two contributions dealing with the MPC algorithm.

First, the MPC algorithm was designed to drive the robot on a path to the goal that would minimize the uncertainty of the position estimate of the goal. The MPC was also designed to explicitly handle the field-of-view constraint of the camera. The field-of-view constraint was incorporated directly into the cost function as a soft constraint. This method of a soft constraint allows the robot to drive around obstacles when necessary.

Second, the MPC algorithm was analyzed to show proof of arrival of the robot at the goal. The proofs shows that the robot will keep the goal in the camera field of view when possible, avoid obstacles, and arrive within some small distance of the goal.

1.4.4 Experimental Results

The above algorithms were implemented and tested on an iRobot Packbot robot platform. In particular, two experiments are presented that show the robot working in real-world scenarios. These experiments highlight the contributions of this work by driving the robot to the goal in conditions where the goal is forced to leave the camera field of view due to obstacles and also the robot drives in rough terrain.

1.5 Document Layout

The organization of this dissertation from this point on will be as follows. Chapter 2 describes the experimental setup that is used for the testing of the developed algorithms. The purpose is to understand what types of sensors will be available for use. These sensors will drive much of the design of the algorithms that follow. The mapping method used to calculate the position of the goal and other landmarks is derived and explained in Chapter 3. Chapter 4 provides analysis of the proposed mapping method. The observability of the system is examined and a consistency analysis through simulations is performed. Once the mapping method is demonstrated then the visual tracking methods and algorithms are described in Chapter 5. The method of tracking and reacquiring is
explained using example images from the robot testing. The method of path planning and control are explained in Chapter 6. Here the method of model predictive control is described and the cost function used in this work is derived. This chapter also explains the method of obstacle detection using a 2D planar LiDAR and also shows a proof of arrival for the robot at the goal point. The chapter ends with many different simulation results that illustrate the behavior of the vehicle in different environments. Chapter 7 documents the results from two separate real-world experiments run on the hardware explained in Chapter 2. Chapter 8 concludes the work by summarizing the results and also exploring some ideas for future work.
CHAPTER 2. EXPERIMENTAL SETUP

2.1 Robot Platform

The robot that is used for this work is an iRobot Packbot 400. This robot is shown in Figure 2.1. The Packbot is a differential drive robot with two tracks that can be controlled independently for controlling the linear velocity and the angular velocity of the vehicle. The Packbot uses a proprietary control protocol via an Ethernet connection for vehicle control. A laptop computer running Windows is used to interface with the Packbot for control. This computer is seen sitting on the Packbot in Figure 2.1. This work relies on that interface to send velocity and angular rate commands to drive the robot to the desired location.

Figure 2.1: Packbot 400 made by iRobot used for experimental testing of this work. The robot is shown with the Windows laptop used for control and with the sensors used for testing.
2.2 Sensors

As can be seen in Figure 2.1, there are various sensors installed on the robot. Each of the sensors used will be discussed here.

2.2.1 Camera

The camera is the main sensor used in this work. The camera is used to track the position of the goal and other landmarks as the robot drives towards the goal. The Xtion PRO LIVE camera is used for this work and is shown in Figure 2.2. This camera is an RGB-D camera which means that the camera can return 3D information about the scene also. The depth is computed using structured-light technology which projects an infrared pattern onto the seen and the sensor can detect that and produce a depth map of objects relatively close to the camera (within 5 meters). This 3D data could be used for obstacle detection, however, this type of RGB-D camera does not work in direct sunlight which reduces the utility of the 3D sensor to indoor use. For this reason, only the monocular camera is used for this work. The monocular camera produces a color image that is 640x480 in size at a rate of 30 Hz. The camera uses a high-speed USB 3.0 interface. The camera is mounted high on the vehicle as shown in Figure 2.1.

![Figure 2.2: Xtion PRO LIVE sensor used for the monocular camera image.](https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/)
2.2.2 Track Encoders

Rotary incremental encoders are used on each track to calculate the track velocity. Figure 2.3(a) shows the encoder mounted on the track drive wheel. These encoders use quadrature output so the direction of the encoder can be calculated. The quadrature lines of the encoder are connected to a Phidget high speed encoder board that has a USB interface.

2.2.3 LiDAR

A 2D scanning LiDAR is used for obstacle detection. The Hokuyo UTM-30LX LiDAR is used on the Packbot. The LiDAR is shown in Figure 2.3(b). This LiDAR functions in direct sunlight and therefore can be used outdoors. The UTM-30LX has a range of 30 meters and has an angular range of 270°. It is mounted in the front of the vehicle as shown in Figure 2.1. This LiDAR uses a USB interface.

2.2.4 Inertial Measurement Unit

An inertial measurement unit (IMU) is used to measure the angular rate of the robot. The sensor is shown in Figure 2.4. This sensor has three accelerometers that measure linear acceleration
along the sensor $x, y,$ and $z$ axes. There are also three fiber optic gyros (FOG) used to measure the angular rates about the sensor $x, y,$ and $z$ axes. The sensor interface is RS-422. An RS-422 to USB adapter is used so that the USB can be plugged into a computer.

![Figure 2.4: KVH 1725 IMU mounted on the Packbot.](image)

2.2.5 GPS

The methods developed in this work do not rely on GPS for feedback. A GPS receiver is used only for ground-truth reference for the experimental results. The GPS unit is mounted high on the Packbot as shown in Figure 2.1. The GPS used is the Trimble Ag 162 unit. This unit is capable of DGPS for 0.2 to 0.3 m of pass-to-pass accuracy. The interface to the GPS receiver is serial RS-232. This interface is converted to USB using an RS-232 to USB converter.
2.3 Computer

A Lenovo ThinkPad with a Quad core 2.4 GHz Intel Core i7 processor is used for sensor interfacing and algorithm processing. This computer runs the Robot Operating System (ROS) [44] which is a framework used for writing different software applications that can use the built-in messaging for easy interfacing. The layout of individual software components will be described in more detail in Chapter 7. The Linux machine sends drive commands (linear and angular velocity) to the Windows machine via an Ethernet connection and the Windows machine simply forwards that on to the Packbot also over an Ethernet connection using the proprietary control interface.

2.4 Test Facility

The robot is tested in different scenarios at Autonomous Solutions, Inc. (ASI). ASI is a robotics company that has different test tracks and testing areas that can be used for on-road and off-road test scenarios. In particular, a pavement test track with barriers is used for on-road testing. This presents a good test area for a scenario where the robot is required to navigate obstacles and lose the goal for a significant amount of time. The second area used for testing will be an off-road scenario where the robot will drive in rough terrain. This is to test the robot in terrain that will cause the robot to pitch and roll while driving towards the goal.
CHAPTER 3. MAPPING

3.1 Problem Description

The goal location for the robot is chosen by a user manually selecting a point in the current image from the on-board camera. When the goal is selected in the image, the location of the goal is known in the image frame, however, because this is a monocular camera, the position of the goal is not known. The position of the goal is constrained to be on an infinite-length line that starts at the camera image center and extends through the actual goal position. Any point along the infinite line will have the same image frame coordinates.

The selected goal point is a position relative to the robot, therefore the position of the robot in the world is not required. The problem then becomes how to calculate the actual position of the goal with respect to the robot. The sensors that are available on the robot for localization are two track velocity sensors and a camera. The camera images are used to track the goal in the image as it moves due to vehicle motion. The track velocity measurements are used for calculating the position of the vehicle as it drives by dead-reckoning. These measurements, however, contain noise and the vehicle model also contains errors which means that the estimated position will drift from the true position.

Because the camera only sees the current scene projected onto an image plane, it is not possible to calculate the depth to an unknown object in an image from a single camera image. For this reason the selected object must be tracked in the image while the vehicle is moving. These different views of the object coupled with motion of the vehicle can be used to estimate the depth to the object. As more and more image measurements are taken while the vehicle moves, the depth estimate of the object can be improved.

As an illustration of this, consider a camera with perfectly known position and orientation as shown in Figure 3.1. The goal is shown in the figure as the yellow circle. If view 1 is only used then the goal could be at any location along the line-of-sight ray that passes through the
camera focal point and the goal (shown in the blue line). This line-of-sight ray extends to infinity. Adding a second view then constrains the position of the goal to be at the intersection of the two different line-of-sight rays. If the positions \((x, y)\) and orientation \(\psi\) are known perfectly for view 1 and view 2 then the position of the goal can be calculated using triangulation. In a similar way, the measurements from a camera with uncertain position and uncertain orientation can be used to estimate the position of a goal point with some uncertainty.

Figure 3.1: The position of a point can be calculated using a monocular camera. If the position and orientation from each view are known then the position of the point can be calculated.

The depth of an object can be calculated using a stereo camera. A stereo camera only needs to see the object from one position because of the fact that there are two cameras separated by a baseline distance which is similar to the situation described in Figure 3.1. Stereo cameras calculate the depth of an object in view of both cameras by finding the object in both images and calculating the disparity (difference of image location in the left image and the right image) and using the known camera configuration to convert the disparity into a depth. This has been found to work well for objects that are close to the camera. The usable distance does depend on the baseline between the two cameras. The wider the baseline then the farther an object can be from the camera and still measure the depth. The depth of the object is inversely proportional to the disparity. This creates a nonlinear relationship and causes the stereo camera to have high depth resolution for objects that are relatively near the camera [45]. This work focuses on selecting objects in the
image frame that are far away from the camera when selected. For this reason, the stereo camera is not used for depth estimation and the monocular camera image is used.

### 3.2 Literature Review

There has been a large amount of work done in the area of robot localization, mapping and SLAM. Much of the work was discussed in Chapter 1. The objective of this work is to drive the robot to the selected goal location. This presents a different problem than most SLAM problems where the desired output from SLAM is a map of the area covered by the robot. This work focuses on calculating the position of the goal with respect to the robot, which allows the robot to properly drive to the goal location.

There is a small body of research that has focused on this problem of calculating the positions of landmarks in the robot-body frame with the main purpose being to improve the consistency of the EKF-SLAM formulation. Some of the research and the applications to this work will be described here. In [46] the method of robocentric mapping is first introduced. The work cites a need for a robocentric formulation to reduce the linearization errors found in the traditional EKF-SLAM methods. The method uses a prediction and a measurement step similar to standard EKF-SLAM methods, but with the states expressed in the robot-body frame. After the prediction and measurement steps there is an additional step of map composition that is used to transform the robocentric map back into an inertial-fixed map frame. Because the linearization is performed in the robocentric frame the overall estimate of the map is improved and the map stays consistent. This research in the robocentric frame was then furthered, as described in [47] and [48] to create the robocentric map-joining algorithm. This method builds on the robocentric mapping approach. In this work the overall global map is created by joining smaller sub-maps. Each sub-map is created using the robocentric mapping method. This allows for large-scale maps to be created while still improving the consistency of the overall map.

The robocentric mapping method was explored further in [49] where the coordinate transform for each landmark position was derived in polar coordinates. This method showed improvements in linearization since the measured state variables were linear in the states defined in polar coordinates. Simulation results were presented showing the validity of the coordinate transform definitions. One limitation of this work is that the motion of the robot is limited to 2D motion along
a flat plane. The state definitions for the landmarks were also restricted to 2D locations along the same plane on which the robot traveled.

The robocentric mapping work in [49] was then extended in [50] to include 3D landmark definitions. The 3D landmarks use a modified spherical coordinate representation of the landmarks, however, the motion of the robot was still constrained to 2D planar motion. Simulation results were presented comparing the robocentric method against a standard world-centric SLAM method. The state definitions for the landmarks use the inverse-depth parametrization derived in [51]. This is an improvement over the use of the depth to the landmark due to the improved linearization properties of the inverse-depth state. The measurement device used was assumed to measure the angles to the landmarks directly and did not include measurement equations that are required for a monocular camera.

The robocentric mapping formulation is used as the mapping framework for the work described in this work. This formulation fits well into the objective, which is to arrive at the selected goal location. This requires knowing where the goal is with respect to the current robot location. The position of the goal in a global frame is not required. The term robocentric mapping as pertaining to this work is defined as the calculation of landmark positions with respect to the current robot location.

### 3.3 Extended Kalman Filter for Robocentric Mapping

As discussed previously, the objective of the robot is to arrive at the goal location which requires the position of the goal point in the robot frame to be known with as little uncertainty as possible. It is also important to note that the goal will not always be visible by the robot camera while driving towards the goal. For this purpose, other landmarks are tracked and estimated also. By adding other landmarks to the mapping filter the position uncertainty of the goal is improved when the goal is not visible.

An extended Kalman filter (EKF) is used as the framework for the monocular based mapping algorithm. A novel formulation is presented that builds upon the method presented in [50], in which a robot-centric frame is used in place of a world-centric frame. The method presented here takes a different approach with the formulation of the state vector of the landmarks. Two types of points are used in the state vector, the main goal point and additional landmarks. These additional
landmarks are used to reduce the uncertainty in the position estimate of the goal when it is not in view. The purpose of the EKF in this work is to track the range and angles from the robot to the goal point. For this reason, the goal point is most important in the EKF, therefore the goal point is treated as the anchor in the state vector. As the anchor, the goal is used to reference the position of each landmark. The state vector only includes the states for the goal point and additional landmarks. There are no vehicle states because the goal point and the landmarks are all mapped with respect to the current robot position.

3.3.1 Coordinate Frame Definition

There are two coordinate frames used in the derivation of the equations of motion for the states used in the EKF. These coordinate frames will be presented here for clarity so they can be used without confusion in the derivations that follow. The first frame is the inertial frame \( F_i \) and the second is the body frame \( F_b \).

**Inertial Frame**

The inertial frame, \( F_i \) is an earth-fixed coordinate system with the unit vector \( k_i \) pointed up. The origin of this frame can be placed anywhere and will not affect the outcome of the derivations.

**Body Frame**

The body frame \( F_b \) is fixed to the vehicle. The origin of this frame is at the center of the vehicle on the ground. This frame translates and rotates with the vehicle. The \( i_b \) axis points out the front of the vehicle with the \( j_b \) axis pointing out the left side of the vehicle and the \( k_b \) axis point up. This frame is shown in Figure 3.2.

3.3.2 Discrete-Time Extended Kalman Filter Formulation

The general equations for a discrete-time extended Kalman filter are given below to help in the derivation of the Kalman filter equations for this problem. The formulation of the EKF used
Figure 3.2: $\mathcal{F}^b$ shown in an isometric view (a) and shown in a right-side view (b). The origin of $\mathcal{F}^b$ is at the center of the vehicle on the ground.

here follows that of [24]. Let the state and measurement equations be governed by the following:

$$x_k = g(u_k, x_{k-1}) + \varepsilon_k$$

$$z_k = h(x_k) + \delta_k,$$

where $\varepsilon_k$ and $\delta_k$ are independent, zero-mean Gaussian random vectors with covariance $R_k$ and $Q_k$ respectively. The subscript $k$ is used to denote the time at $t = k$. These variables $\varepsilon_k$ and $\delta_k$ model the process and measurement noise respectively. The function $g$ is the nonlinear motion model of the state $x$. The function $h$ is the nonlinear measurement model for the sensors used. The EKF has two main steps, a prediction step and a measurement update step.

**Prediction**

The prediction step calculates the current states from the previous states by using the nonlinear motion model $g$. This step simply predicts the future state as

$$\bar{\mu}_k = g(u_k, \mu_{k-1})$$

(3.1)

where the variable $\mu$ is the estimated mean of the state vector $x$. The symbol $\bar{\cdot}$ means the predicted value before a measurement update. The prediction step also predicts the uncertainty of the state
by updating the covariance using the following equation

$$\Sigma_k = G_k \Sigma_{k-1} G_k^T + R_k$$  \hspace{1cm} (3.2)$$

where $\Sigma_k$ is the state covariance matrix at time $t = k$ and $G_k$ is the Jacobian of the nonlinear motion model $g$, and the matrix $R_k$ is the process noise of the state.

**Update**

The update step uses the measurements to update the states and also to update the covariance. This is done by first calculating the Kalman gain as

$$K_k = \Sigma_k H_k^T \left( H_k \Sigma_k H_k^T + Q_k \right)^{-1}$$  \hspace{1cm} (3.3)$$

and then updating the state with the following equation

$$\mu_k = \hat{\mu}_k + K_k \left( z_k - h(\hat{\mu}_k) \right).$$  \hspace{1cm} (3.4)$$

The covariance is then updated by

$$\Sigma_k = (I - K_k H_k) \tilde{\Sigma}_k.$$  \hspace{1cm} (3.5)$$

**Inputs**

The nonlinear prediction equation of Equation 3.1 shows the state propagated using the function $g(u_k, \mu_{k-1})$. The vector $u_k$ is the input vector used in the prediction equation. For this work the input vector is defined as

$$u_k = \begin{bmatrix} V_k \\ p_k \\ q_k \\ r_k \end{bmatrix},$$  \hspace{1cm} (3.6)$$
where \( V_k \) is the forward velocity along the \( i^b \) axis at time \( t = k \) and the variables \( p_k, q_k, \) and \( r_k \) are angular velocities at time \( t = k \) about the body \( i^b, j^b, \) and \( k^b \) axes respectively.

### 3.3.3 State Definitions

The state vector is the current map state. This includes the states for the goal and for all additional landmarks that are being tracked in the filter. The state vector is defined as

\[
x = \left[ \begin{array}{c} x^T_g, \ x^T_{l_1}, \ x^T_{l_2}, \ \ldots, \ x^T_{l_N} \end{array} \right]^T,
\]

where \( N \) is the number of landmarks in addition to the goal, \( x_g \) is the vector that contains the goal states and \( x_{l_i} \) is the vector containing the states for landmark \( i \).

The goal and landmarks are tracked in 3D so each point will consist of three states. The states for the goal are the same states as described in [50] which are modified spherical coordinates. The goal point states are defined as

\[
x_g = \begin{bmatrix} \alpha \\ \beta \\ \rho_g \end{bmatrix}.
\]  \tag{3.7}

The goal point states are shown in Figure 3.3. The angle \( \alpha \) is the azimuth angle which is the same angle used in typical spherical coordinates. The angle \( \alpha \) is measured about the \( k^b \)-axis and is the angle from the \( i^b \)-axis to the projection of the goal point vector onto the \( i^b-j^b \) plane. The angle \( \beta \) is different than the typical polar or zenith angle from spherical coordinates. The angle \( \beta \) is the complement of the polar angle and is measured from the projection of the goal point vector onto the \( i^b-j^b \) plane up to the line along the vector \( p^b_g \). The vector \( p^b_g \) is the vector from the origin of \( \mathcal{F}^b \) to the goal point. The angle \( \beta \) will be referred to as the elevation angle of the goal point. The inverse-depth state \( \rho_g \) is the inverse of the distance \( d \) to the goal point in frame \( \mathcal{F}^b \) as described in [51]. This distance is \( d = \| p^b_g \| \). Using these definitions the position of the goal in \( \mathcal{F}^b \) is

\[
p^b_g = \begin{bmatrix} \frac{1}{\rho_g} \cos \alpha \cos \beta \\ \frac{1}{\rho_g} \sin \alpha \cos \beta \\ \frac{1}{\rho_g} \sin \beta \end{bmatrix}.
\]  \tag{3.8}
Figure 3.3: State definitions for the goal point. The states are defined relative to $F^b$. The states $\alpha$ and $\beta$ are shown on the image. The inverse-depth state ($\rho_g$) for the goal is also shown.

The state vector used for the representation of landmark $i$ is defined as

$$x_i = \begin{bmatrix} \theta_i \\ \eta_i \\ \rho_i \end{bmatrix}.$$ (3.9)

The two angle states $\theta_i$ and $\eta_i$ are similar to the angles $\alpha$ and $\beta$ yet they are defined with respect to the goal point angles and not with respect to the axes of $F^b$. An example of a single additional landmark is shown in Figure 3.4. The angle $\theta_1$ is the angle from the projection of the goal point vector onto the $i^b$-$j^b$ plane (shown by the dashed blue line) to the projection of the landmark vector $p_{l_1}^b$ onto the $i^b$-$j^b$ plane. The vector $p_{l_1}^b$ is the vector from the origin of $F^b$ to the landmark point. The angle $\beta_1$ is the angle from the goal point vector $p_g^b$ rotated about the $k^b$ axis through the angle $\theta_1$ up to the landmark vector. The goal point vector $p_g^b$ rotated through the angle $\theta_1$ is shown in red in Figure 3.4. This definition of the landmark states is a key difference in this work compared to
the work explained in [50]. Using these definitions the position of landmark \(i\) in \(\mathcal{F}^b\) is

\[
\mathbf{p}^b_{li} = \begin{bmatrix}
\frac{1}{\rho_i} \cos (\alpha + \theta_i) \cos (\beta + \eta_i) \\
\frac{1}{\rho_i} \sin (\alpha + \theta_i) \cos (\beta + \eta_i) \\
\frac{1}{\rho_i} \sin (\beta + \eta_i)
\end{bmatrix}
\] (3.10)

Figure 3.4: State definitions for a single landmark. The states are defined relative to the goal point states from Figure 3.3. The angle \(\theta_1\) is the angle from the projection of the goal point vector onto the \(i^b\)-\(j^b\) plane (shown by the dashed blue line) to the projection of the landmark vector onto the \(i^b\)-\(j^b\) plane. The angle \(\beta_1\) is the angle from the goal point vector \(\mathbf{p}^g_b\) rotated about the \(k^b\) axis through the angle \(\theta_1\) (shown in dashed red) up to the landmark vector.
3.4 State Motion Model

The extended Kalman filter equations shown in Equation 3.1 to 3.5 are discrete-time functions. The state motion model is therefore desired in discrete-time also, however, it is beneficial to derive the motion model in continuous-time and then convert that to discrete-time for the EKF equations.

The state definitions for the goal point and landmark states were defined above in Equations 3.7 and 3.9, however, the derivation of the state motion equations can be simplified by using the actual depth to the goal \( d_g \) and the actual depth to landmark \( d_i \) in place of the inverse-depth states \( \rho \) and \( \rho_i \). Temporary state vectors for the goal point and landmarks using depth in place of inverse depth are given by

\[
x_{g,d} = \begin{bmatrix} \alpha \\ \beta \\ d_g \end{bmatrix},
\]

and

\[
x_{l,i,d} = \begin{bmatrix} \theta_i \\ \eta_i \\ d_i \end{bmatrix}.
\]

The subscript \( d \) is used to denote that this is the depth state definition. Using these state representations the position of the goal in \( F^b \) is

\[
p^b_g = \begin{bmatrix} d_g \cos \alpha \cos \beta \\ d_g \sin \alpha \cos \beta \\ d_g \sin \beta \end{bmatrix}
\] (3.11)

and the position of landmark \( i \) in \( F^b \) is

\[
p^b_{l,i} = \begin{bmatrix} d_i \cos (\alpha + \theta_i) \cos (\beta + \eta_i) \\ d_i \sin (\alpha + \theta_i) \cos (\beta + \eta_i) \\ d_i \sin (\beta + \eta_i) \end{bmatrix}.
\] (3.12)
With these definitions, the state motion model can be derived. First, the vehicle model is explained. Next, the motion model for the goal states are derived and then the motion model for landmark states are derived. The derivations assume that the goal and all the landmarks are fixed in frame $F_i$. The purpose of the motion model is to know how the states evolve under vehicle motion.

3.4.1 Vehicle Model

The Packbot robot is a tracked differential drive robot which has nonholonomic motion. Work has been done in [52] to account for the track slippage but this will not be used in the state derivations. The derivations of the states will not be limited to driving on a flat plane. The states used in this work model the position of the goal relative to the robot. Because of the relative nature of this problem, the normal Euler angle definitions are not required, even for 3D motion of the vehicle. The evolution of the states describing the locations of the goal and landmarks in time is independent of the vehicle roll, pitch, and yaw with respect to frame $F_i$. This will be shown in the derivations that follow. For the purpose of these derivations, the vehicle model is explained with respect to frame $F^b$.

There are two control inputs into the vehicle. The first is the velocity along the body $i^b$-axis. Because of the nonholonomic constraint of the vehicle, the vehicle can only move along the current $i^b$-axis, therefore, the vehicle velocity in the body frame is

$$V^b = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}.$$ 

The second control input is the angular rate about the $k^b$-axis and this is denoted as $r$. The two control inputs $(V, r)$ will be controlled by the vehicle tracks. The actual $V$ and $r$ values will be measured using track speed encoders and also an IMU. Two other inputs into the system are the angular rates about the $i^b$ and $j^b$ axes which are $p$ and $q$ respectively. These two inputs are not control inputs but depend on the terrain upon which the vehicle is driving. These rates will be measured by an IMU and fed into the vehicle model.
3.4.2 Goal Model

The motion model of the goal is derived using two frames, the body frame $\mathcal{F}^b$ and the inertial frame $\mathcal{F}^i$. For this derivation the goal is placed at the origin of frame $\mathcal{F}^i$. The vector $\mathbf{p}^b$ is the vector from the goal to the robot expressed in frame $\mathcal{F}^b$. The goal motion model is derived by taking the time derivative of $\mathbf{p}^b$ in frame $\mathcal{F}^i$.

Differentiation of a Vector

The time derivative of a vector $\mathbf{p}$ is derived in [53]. This derivation calculates the time derivative of $\mathbf{p}$ as seen in frame $\mathcal{F}^i$. Let $\mathbf{p}$ be moving in frame $\mathcal{F}^b$ and let $\mathcal{F}^b$ be rotating with respect to $\mathcal{F}^i$. The time derivative of $\mathbf{p}$ in frame $\mathcal{F}^i$ is

$$\frac{d}{dt_i} \mathbf{p} = \frac{d}{dt_b} \mathbf{p} + \omega_{b/i} \times \mathbf{p}. \tag{3.13}$$

The first term on the right-hand side of Equation 3.13 is the motion of the vector $\mathbf{p}$ as viewed from the rotating frame $\mathcal{F}^b$ and the second term represents the motion of $\mathbf{p}$ due to the rotation of $\mathcal{F}^b$ relative to $\mathcal{F}^i$ which is $\omega_{b/i}$.

Motion Model

The position of the goal in frame $\mathcal{F}^b$ is defined in Equation 3.11. This is then used to define the vector from the goal to the vector as

$$\mathbf{p}_r^b = -\mathbf{p}_g^b = \begin{bmatrix} -d_g \cos \alpha \cos \beta \\ -d_g \sin \alpha \cos \beta \\ -d_g \sin \beta \end{bmatrix}. \tag{3.14}$$

Now computing the time derivative of $\mathbf{p}_r^b$ following Equation 3.13 gives

$$\frac{d}{dt_i} \mathbf{p}_r = \frac{d}{dt_b} \mathbf{p}_r + \omega_{b/i} \times \mathbf{p}_r. \tag{3.15}$$
It is desired to calculate how the goal states evolve in time with respect to frame $F^b$. This is done by rearranging Equation 3.15 as

$$\frac{d}{dt_b} p_r = \frac{d}{dt_i} p_r - \omega_{b/i} \times p_r. \tag{3.16}$$

Now it is desired to evaluate Equation 3.16 in frame $F^b$. This is done by expressing each vector in frame $F^b$ as

$$\frac{d}{dt_b} p_r^b = \frac{d}{dt_i} p_r^b - \omega_{b/i}^b \times p_r^b, \tag{3.17}$$

where $p_r^b$ is defined in Equation 3.14 and the angular velocity vector $\omega_{b/i}^b$ is the angular velocity of the body with respect to frame $F^i$ expressed in the body frame and is

$$\omega_{b/i}^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{3.18}$$

The term on the left-hand side of Equation 3.17 is the time derivative of the goal with respect to frame $F^b$. This is the time derivative of Equation 3.14 and is

$$\frac{d}{dt_b} p_r^b = \begin{bmatrix} d_g \dot{\alpha} \sin \alpha \cos \beta + d_g \dot{\beta} \cos \alpha \sin \beta - d_g \cos \alpha \cos \beta \\ -d_g \dot{\alpha} \cos \alpha \cos \beta + d_g \dot{\beta} \sin \alpha \sin \beta - d \sin \alpha \cos \beta \\ -d_g \dot{\beta} \cos \beta - d_g \sin \beta \end{bmatrix}. \tag{3.19}$$

The first term on the right side of Equation 3.17 is the linear velocity of the robot with respect to frame $F^i$ expressed in frame $F^b$ and is

$$\frac{d}{dt_i} p_r^b = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}. \tag{3.20}$$
Equation 3.19 can be used to separate out the values of $\dot{\alpha}, \dot{\beta},$ and $\dot{d}_g$ as

$$\frac{d}{dt_b} \mathbf{p}^b = A \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{d}_g \end{bmatrix},$$  \hspace{1cm} (3.21)$$

where

$$A = \begin{bmatrix} d_g \sin \alpha \cos \beta & d_g \cos \alpha \sin \beta & -\cos \alpha \cos \beta \\ -d_g \cos \alpha \cos \beta & d_g \sin \alpha \sin \beta & -\sin \alpha \cos \beta \\ 0 & -d_g \cos \beta & -\sin \beta \end{bmatrix}.$$ 

Substituting Equations 3.21, 3.20, 3.18, and 3.14 into Equation 3.17 gives

$$A \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{d}_g \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} -d_g \cos \alpha \cos \beta \\ -d_g \sin \alpha \cos \beta \\ -d_g \sin \beta \end{bmatrix},$$  \hspace{1cm} (3.22)$$

which is then used to solve for the motion of the states in frame $\mathcal{F}^b$ as

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{d}_g \end{bmatrix} = A^{-1} \left( \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} -d_g \cos \alpha \cos \beta \\ -d_g \sin \alpha \cos \beta \\ -d_g \sin \beta \end{bmatrix} \right),$$  \hspace{1cm} (3.23)$$

where

$$A^{-1} = \begin{bmatrix} \frac{\sin \alpha \sec \beta}{d_g} & \frac{-\cos \alpha \sec \beta}{d_g} & 0 \\ \frac{\cos \alpha \sin \beta}{d_g} & \frac{\sin \alpha \sin \beta}{d_g} & \frac{-\cos \beta}{d_g} \\ -\cos \alpha \cos \beta & -\cos \beta \sin \alpha & -\sin \beta \end{bmatrix}.$$ 

The final motion model for the goal states is solved by performing the operations and simplifying Equation 3.23, which gives

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{d}_g \end{bmatrix} = \begin{bmatrix} \frac{V \sin \alpha}{d_g \cos \beta} + p \cos \alpha \tan \beta + q \sin \alpha \tan \beta - r \\ \frac{V \cos \alpha \sin \beta}{d_g} - p \sin \alpha + q \cos \beta \\ -V \cos \alpha \cos \beta \end{bmatrix}.$$  \hspace{1cm} (3.24)$$
This confirms that the evolution of $\alpha$, $\beta$, and $d_g$ in time are independent of the attitude of the robot, and interesting and simplifying result.

### 3.4.3 Landmark Model

In [50] the landmarks are defined independent of one another. Meaning the observation of one landmark does not introduce new information about the position of another landmark. In this work, the goal point is the most important landmark because that is where the vehicle needs to arrive. It is important for the vehicle to gain as much information about the goal as possible. For this purpose, a new state formulation is derived for landmarks that are not the goal.

To derive the motion of the landmark states $\begin{bmatrix} x_{l1}^T, x_{l2}^T, \ldots, x_{lN}^T \end{bmatrix}^T$ it is beneficial to define two new variables. Let the angle $\gamma_i$ be the azimuth angle to landmark $i$ similar to how the angle $\alpha$ is the azimuth angle to the goal point. Also, let the angle $\lambda_i$ be the elevation angle for landmark $i$. These angles are shown in Figure 3.5. These angles are defined as

$$\gamma_i = \alpha + \theta_i \quad \text{(3.25)}$$

and

$$\lambda_i = \beta + \eta_i. \quad \text{(3.26)}$$

Using these angles the position of landmark $i$ in frame $F^b$ is

$$p_{l,i}^b = \begin{bmatrix} d_i \cos \gamma_i \cos \lambda_i \\ d_i \sin \gamma_i \cos \lambda_i \\ d_i \sin \lambda_i \end{bmatrix}.$$

Differentiating Equations 3.25 and 3.26 with respect to time gives

$$\dot{\gamma}_i = \dot{\alpha} + \dot{\theta}_i$$

and

$$\dot{\lambda}_i = \dot{\beta} + \dot{\eta}_i.$$
Solving for the desired states for the landmarks gives

\[ \dot{\theta}_i = \dot{\gamma}_i - \dot{\alpha} \]  \hspace{1cm} (3.27)

and

\[ \dot{\eta}_i = \dot{\lambda}_i - \dot{\beta}. \]  \hspace{1cm} (3.28)

To solve for the motion of the landmark states \( \theta_i \) and \( \eta_i \), the motion of the goal states \( \alpha \) and \( \beta \) are needed. These have already been computed in Equation 3.24. The motion of \( \gamma_i \) and \( \lambda_i \) are also needed. These can be found by utilizing the relationship equivalence between \( \gamma_i \), \( \lambda_i \) and the landmark with \( \alpha \), \( \beta \) and the goal point.

Making the substitution of \( \gamma_i \) from Equation 3.25 and \( \lambda_i \) from Equation 3.26 for \( \alpha \) and \( \beta \) respectively into Equation 3.24 gives the following motion model for the angles \( \gamma \) and \( \lambda \)
\[
\begin{bmatrix}
\dot{\gamma}_i \\
\dot{\lambda}_i
\end{bmatrix} = \begin{bmatrix}
\frac{V \sin(\alpha + \theta_i)}{d_i \cos (\beta + \eta_i)} + p \cos (\alpha + \theta_i) \tan (\beta + \eta_i) + q \sin (\alpha + \theta_i) \tan (\beta + \eta_i) \\
\frac{V \cos(\alpha + \theta_i) \sin (\beta + \eta_i)}{d_i} - p \sin (\alpha + \theta_i) + q \cos (\beta + \eta_i)
\end{bmatrix}.
\] (3.29)

Now plugging in the components of Equation 3.29 with the components of Equation 3.24 into Equations 3.27 and 3.28 gives the motion model for the landmark states as

\[
\begin{bmatrix}
\dot{\theta}_i \\
\dot{\eta}_i \\
\dot{d}_i
\end{bmatrix} = \begin{bmatrix}
\frac{V \sin (\alpha + \theta_i)}{d_i \cos (\beta + \eta_i)} + p \cos (\alpha + \theta_i) \tan (\beta + \eta_i) + q \sin (\alpha + \theta_i) \tan (\beta + \eta_i) \\
- \left( \frac{V \sin \alpha}{d_g \cos \beta} + p \cos \alpha \tan \beta + q \sin \alpha \tan \beta \right) \\
\frac{V \cos (\alpha + \theta_i) \sin (\beta + \eta_i)}{d_i} - p \sin (\alpha + \theta_i) + q \cos (\beta + \eta_i) - \\
\frac{V \cos \alpha \sin \beta}{d_g} - p \sin \alpha + q \cos \beta \\
- V \cos (\alpha + \theta_i) \cos (\beta + \eta_i)
\end{bmatrix}.
\] (3.30)

The equation for \( \dot{d}_i \) was computed by substituting Equations 3.25 and 3.26 into Equation 3.24.

### 3.4.4 Conversion to \( \rho \)

The equations of motion for the states were derived using \( d_g \) and \( d_i \) in place of \( \rho_g \) and \( \rho_i \) respectively. Equations 3.24 and 3.30 are now converted to inverse-depth. This is done by using the defined relationship of

\[
d = \frac{1}{\rho}.
\] (3.31)

Taking the time derivative of Equation 3.31 gives the following relationship

\[
\dot{d} = - \frac{\rho}{\rho^2}.
\] (3.32)

The state equations of motion are converted by simply substituting Equations 3.31 and 3.32 into Equations 3.24 and 3.30. Performing these substitutions gives the following motion model for the
and the following motion model for the landmarks

\[
f_i = \begin{bmatrix}
\dot{\theta}_i \\
\dot{\eta}_i \\
\dot{\rho}_i
\end{bmatrix} = \begin{bmatrix}
\frac{V \rho_i \sin (\alpha + \theta_i)}{\cos (\beta + \eta_i)} + p \cos (\alpha + \theta_i) \tan (\beta + \eta_i) + q \sin (\alpha + \theta_i) \tan (\beta + \eta_i) - V \rho_i \cos (\alpha + \theta_i) \sin (\beta + \eta_i) - p \sin (\alpha + \theta_i) + q \cos (\beta + \eta_i) - V \rho_i^2 \cos (\alpha + \theta_i) \cos (\beta + \eta_i)
\end{bmatrix}.
\]  

(3.34)

**Discrete-Time Motion Models**

The discrete-time motion model is used in the EKF as shown in Equation 3.1. Let the motion model equation for the entire state vector be defined as

\[
g = \begin{bmatrix}
g_g \\
g_l_i
\end{bmatrix},
\]

where

\[
g_g = \begin{bmatrix}
g_\alpha \\
g_\beta \\
g_\rho_l
\end{bmatrix}.
\]
and

\[ g_{li} = \begin{bmatrix} g\theta_i \\ g\eta_i \\ g\rho_i \end{bmatrix}. \]

The continuous-time motion models for the states are given in Equation 3.33 and Equation 3.34. The discrete-time state motion model is derived by using the current state plus the continuous-time rate equations integrated by the current \( \Delta t \) of the system. For the goal states this becomes

\[
\begin{bmatrix}
\alpha + \left( \frac{V\rho_g \sin \alpha}{\cos \beta} + p \cos \alpha \tan \beta + q \sin \alpha \tan \beta - r \right) \Delta t \\
\beta + (V\rho_g \cos \alpha \sin \beta - p \sin \alpha + q \cos \beta) \Delta t \\
\rho_g - (V \cos \alpha \cos \beta) \Delta t 
\end{bmatrix}.
\]

The motion model equation for landmark \( i \) is

\[
\begin{bmatrix}
\theta_i + \left( V\rho_i \sin \left( \alpha + \theta_i \right) \cos \left( \beta + \eta_i \right) + p \cos \left( \alpha + \theta_i \right) \tan \left( \beta + \eta_i \right) + q \sin \left( \alpha + \theta_i \right) \tan \left( \beta + \eta_i \right) \\
\beta_i + \left( V\rho_i \cos \left( \alpha + \theta_i \right) \sin \left( \beta + \eta_i \right) - p \sin \left( \alpha + \theta_i \right) + q \cos \left( \beta + \eta_i \right) -
\right. \\
\left. - V\rho_g \cos \alpha \sin \beta + p \sin \alpha - q \cos \beta \right) \Delta t \\
\rho_i - (V \cos \left( \alpha + \theta_i \right) \cos \left( \beta + \eta_i \right)) \Delta t 
\end{bmatrix}.
\]

### 3.4.5 Process Noise

The process noise \( R_k \) in Equation 3.2 is used to model the uncertainty due to the motion of the states. This can simply be modeled as a zero-mean Gaussian distribution with covariance \( R_k \) where the diagonal elements of the covariance are used to add noise at each step onto the corresponding state. In this work, a method explained in [24] will be used. First the covariance of the noise in the control space is computed and then this is mapped into the state space to add uncertainty into the states. The control space for this work is the input vector \( u \) defined in Equation 3.6.
The covariance for the control space is

\[
M_k = \begin{bmatrix}
\delta_1 V_k^2 + \delta_2 r_k^2 & 0 & 0 & 0 \\
0 & \delta_3 p_k^2 & 0 & 0 \\
0 & 0 & \delta_4 q_k^2 & 0 \\
0 & 0 & 0 & \delta_5 V_k^2 + \delta_6 r_k^2
\end{bmatrix},
\]

where the constants \( \delta_i \) are standard deviations used to model the noise that is proportional to the input. The values of \( \delta_i \) are chosen based on the data of the inputs used. This will be set based on the vehicle and the measurement device used to measure the inputs. For this vehicle, the velocity \( V_k \) is measured from track encoders and the angular velocity is measured from an inertial measurement unit (IMU). The noise in control space is transformed to state space by using a linear approximation of the motion model \( g \). The Jacobian for this is the partial derivative of \( g \) with respect to the input vector \( u_k \). The Jacobian for the goal and for each additional landmark are computed independently. The Jacobian for the goal is defined as

\[
\gamma_{g,k} = \frac{\partial g(u_k, \mu_{k-1})}{\partial u_k} = \begin{bmatrix}
\frac{\partial g_\alpha}{\partial V_k} & \frac{\partial g_\alpha}{\partial p_k} & \frac{\partial g_\alpha}{\partial q_k} & \frac{\partial g_\alpha}{\partial r_k} \\
\frac{\partial g_\beta}{\partial V_k} & \frac{\partial g_\beta}{\partial p_k} & \frac{\partial g_\beta}{\partial q_k} & \frac{\partial g_\beta}{\partial r_k} \\
\frac{\partial g_\rho}{\partial V_k} & \frac{\partial g_\rho}{\partial p_k} & \frac{\partial g_\rho}{\partial q_k} & \frac{\partial g_\rho}{\partial r_k}
\end{bmatrix}.
\]

The partial derivative terms are listed below.

\[
\frac{\partial g_\alpha}{\partial V_k} = \rho g \sin(\alpha) \sec(\beta) \Delta t
\]

\[
\frac{\partial g_\alpha}{\partial p_k} = \cos(\alpha) \tan(\beta) \Delta t
\]

\[
\frac{\partial g_\alpha}{\partial q_k} = \sin(\alpha) \tan(\beta) \Delta t
\]

\[
\frac{\partial g_\alpha}{\partial r_k} = -\Delta t
\]

\[
\frac{\partial g_\beta}{\partial V_k} = \rho g \cos(\alpha) \sin(\beta) \Delta t
\]
\[
\frac{\partial g_\beta}{\partial p_k} = -\sin(\alpha)\Delta t
\]
\[
\frac{\partial g_\beta}{\partial q_k} = \cos(\alpha)\Delta t
\]
\[
\frac{\partial g_\beta}{\partial r_k} = 0
\]
\[
\frac{\partial g_\rho_k}{\partial V_k} = \rho_g^2 \cos(\alpha) \cos(\beta)\Delta t
\]
\[
\frac{\partial g_\rho_k}{\partial p_k} = 0
\]
\[
\frac{\partial g_\rho_k}{\partial q_k} = 0
\]
\[
\frac{\partial g_\rho_k}{\partial r_k} = 0
\]

The input Jacobian for landmark \(i\) is defined as
\[
\mathcal{J}_{i,k} = \frac{\partial g_i(u_k, \mu_{k-1})}{\partial u_k} = \begin{bmatrix}
\frac{\partial g_{\theta_i}}{\partial V_k} & \frac{\partial g_{\theta_i}}{\partial p_k} & \frac{\partial g_{\theta_i}}{\partial q_k} & \frac{\partial g_{\theta_i}}{\partial r_k} \\
\frac{\partial g_{\eta_i}}{\partial V_k} & \frac{\partial g_{\eta_i}}{\partial p_k} & \frac{\partial g_{\eta_i}}{\partial q_k} & \frac{\partial g_{\eta_i}}{\partial r_k} \\
\frac{\partial g_{\rho_i}}{\partial V_k} & \frac{\partial g_{\rho_i}}{\partial p_k} & \frac{\partial g_{\rho_i}}{\partial q_k} & \frac{\partial g_{\rho_i}}{\partial r_k}
\end{bmatrix}
\]

The partial derivative terms are listed below.

\[
\frac{\partial g_{\theta_i}}{\partial V_k} = (\rho_i \sin(\alpha + \theta_i) \sec(\beta + \eta_i) - \rho_g \sin(\alpha) \sec(\beta))\Delta t
\]
\[
\frac{\partial g_{\theta_i}}{\partial p_k} = (\cos(\alpha + \theta_i) \tan(\beta + \eta_i) - \cos(\alpha) \tan(\beta))\Delta t
\]
\[
\frac{\partial g_{\theta_i}}{\partial q_k} = (\sin(\alpha + \theta_i) \tan(\beta + \eta_i) - \sin(\alpha) \tan(\beta))\Delta t
\]
\[
\frac{\partial g_{\theta_i}}{\partial r_k} = 0
\]
\[
\frac{\partial g_{\eta_i}}{\partial V_k} = (\rho_i \cos(\alpha + \theta_i) \sin(\beta + \eta_i) - \rho_g \cos(\alpha) \sin(\beta))\Delta t
\]
\[
\frac{\partial g_{\eta_i}}{\partial p_k} = (\sin(\alpha) - \sin(\alpha + \theta_i))\Delta t
\]
\[
\frac{\partial g_{\eta_i}}{\partial q_k} = (\cos(\alpha + \theta_i) - \cos(\alpha)) \Delta t \\
\frac{\partial g_{\eta_i}}{\partial r_k} = 0 \\
\frac{\partial g_{\rho_i}}{\partial V_k} = \rho_i^2 \cos(\alpha + \theta_i) \cos(\beta + \eta_i) \Delta t \\
\frac{\partial g_{\rho_i}}{\partial p_k} = 0 \\
\frac{\partial g_{\rho_i}}{\partial q_k} = 0 \\
\frac{\partial g_{\rho_i}}{\partial r_k} = 0
\]

The process noise covariance \( R_k \) is then computed as

\[
R_k = \begin{bmatrix}
V_{g,k}M_kV^T_{g,k} & 0 & 0 & \ldots & 0 \\
0 & V_{l_1,k}M_kV^T_{l_1,k} & 0 & \ldots & 0 \\
0 & 0 & V_{l_2,k}M_kV_{l_2,k} & 0 & 0 \\
\vdots & \vdots & 0 & \ddots & 0 \\
0 & 0 & \ldots & 0 & V_{l_N,k}M_kV_{l_N,k}
\end{bmatrix}
\]

### 3.4.6 Motion Model Jacobian

As shown in Equation 3.2 the Jacobian \( G_k \) of the nonlinear function \( g \) is needed to predict the covariance matrix at each time step. The Jacobian is

\[
G_k = \frac{\partial g(u_k, \mu_{k-1})}{\partial x_{k-1}}.
\]
The overall state Jacobian $G$ is defined in the following block matrix form

$$G = \begin{bmatrix}
G_g & 0 & 0 & \ldots & 0 \\
G_{1,g} & G_{1,1} & 0 & \ldots & 0 \\
G_{2,g} & 0 & G_{2,2} & 0 & 0 \\
\vdots & \vdots & 0 & \ddots & 0 \\
G_{N,g} & 0 & \ldots & 0 & G_{N,N}
\end{bmatrix},$$

where

$$G_g = \begin{bmatrix}
\frac{\partial g_\alpha}{\partial \alpha} & \frac{\partial g_\alpha}{\partial \beta} & \frac{\partial g_\alpha}{\partial \rho_g} \\
\frac{\partial g_\theta}{\partial \alpha} & \frac{\partial g_\theta}{\partial \beta} & \frac{\partial g_\theta}{\partial \rho_g} \\
\frac{\partial g_\eta}{\partial \alpha} & \frac{\partial g_\eta}{\partial \beta} & \frac{\partial g_\eta}{\partial \rho_g} \\
\frac{\partial g_{g_0}}{\partial \alpha} & \frac{\partial g_{g_0}}{\partial \beta} & \frac{\partial g_{g_0}}{\partial \rho_g}
\end{bmatrix},$$

$$G_{i,g} = \begin{bmatrix}
\frac{\partial g_{\theta_i}}{\partial \theta_i} & \frac{\partial g_{\eta_i}}{\partial \eta_i} & \frac{\partial g_{\rho_i}}{\partial \rho_g} \\
\frac{\partial g_{\theta_i}}{\partial \theta_i} & \frac{\partial g_{\eta_i}}{\partial \eta_i} & \frac{\partial g_{\rho_i}}{\partial \rho_g} \\
\frac{\partial g_{g_{\theta_i}}}{\partial \theta_i} & \frac{\partial g_{g_{\eta_i}}}{\partial \eta_i} & \frac{\partial g_{g_{\rho_i}}}{\partial \rho_g}
\end{bmatrix},$$

and

$$G_{i,i} = \begin{bmatrix}
\frac{\partial g_{\theta_i}}{\partial \theta_i} & \frac{\partial g_{\eta_i}}{\partial \eta_i} & \frac{\partial g_{\rho_i}}{\partial \rho_i} \\
\frac{\partial g_{\theta_i}}{\partial \theta_i} & \frac{\partial g_{\eta_i}}{\partial \eta_i} & \frac{\partial g_{\rho_i}}{\partial \rho_i} \\
\frac{\partial g_{g_{\theta_i}}}{\partial \theta_i} & \frac{\partial g_{g_{\eta_i}}}{\partial \eta_i} & \frac{\partial g_{g_{\rho_i}}}{\partial \rho_i}
\end{bmatrix}.$$
\[ \frac{\partial g_\beta}{\partial \alpha} = -\Delta t (\sin(\alpha)(\rho_g V \sin(\beta) + q) + p \cos(\alpha)) \]

\[ \frac{\partial g_\beta}{\partial \beta} = 1 + \Delta t \rho_g V \cos(\alpha) \cos(\beta) \]

\[ \frac{\partial g_\beta}{\partial \rho_g} = \Delta t V \cos(\alpha) \sin(\beta) \]

\[ \frac{\partial g_\rho}{\partial \alpha} = -\Delta t \rho_g^2 V \sin(\alpha) \cos(\beta) \]

\[ \frac{\partial g_\rho}{\partial \beta} = -\Delta t \rho_g^2 V \cos(\alpha) \sin(\beta) \]

\[ \frac{\partial g_\rho}{\partial \rho_g} = 1 + 2 \Delta t \rho_g V \cos(\alpha) \cos(\beta) \]

The elements of the matrix \( G_{i,g} \) are now shown. These are the partial derivatives of the landmark \( i \) states with respect to the goal states \( x_g \).

\[ \frac{\partial g_{\theta_i}}{\partial \alpha} = \Delta t \left( p (\sin(\alpha) \tan(\beta) - \sin(\alpha + \theta_i) \tan(\beta + \eta_i)) \right) + \]
\[ \cos(\alpha + \theta_i) \sec(\beta + \eta_i) (q \sin(\beta + \eta_i) + \rho_i V) - \cos(\alpha) \sec(\beta) (q \sin(\beta) + \rho_g V) \right). \]

\[ \frac{\partial g_{\theta_i}}{\partial \beta} = \Delta t \left( p \cos(\alpha + \theta_i) \sec^2(\beta + \eta_i) + q \sin(\alpha + \theta_i) \sec^2(\beta + \eta_i) + \rho_i V \sin(\alpha + \theta_i) \tan(\beta + \eta_i) - p \cos(\alpha) \sec^2(\beta) - q \sin(\alpha) \sec^2(\beta) - \rho_g V \sin(\alpha) \tan(\beta) \sec(\beta) \right) \]

\[ \frac{\partial g_{\theta_i}}{\partial \rho_g} = -\Delta t V \sin(\alpha) \sec(\beta) \]
\[
\frac{\partial g_{\eta_i}}{\partial \alpha} = \Delta t \left( -\rho_i V \sin(\alpha + \theta_i) \sin(\beta + \eta_i) + \rho_s V \sin(\alpha) \sin(\beta) - p \cos(\alpha + \theta_i) + p \cos(\alpha) - q \sin(\alpha + \theta_i) + q \sin(\alpha) \right)
\]

\[
\frac{\partial g_{\eta_i}}{\partial \beta} = \Delta t \left( \rho_i V \cos(\alpha + \theta_i) \cos(\beta + \eta_i) - \rho_s V \cos(\alpha) \cos(\beta) \right)
\]

\[
\frac{\partial g_{\eta_i}}{\partial \rho_s} = -\Delta t V \cos(\alpha) \sin(\beta)
\]

\[
\frac{\partial g_{\rho_i}}{\partial \alpha} = -\Delta t \rho_i^2 V \sin(\alpha + \theta_i) \cos(\beta + \eta_i)
\]

\[
\frac{\partial g_{\rho_i}}{\partial \beta} = -\Delta t \rho_i^2 V \cos(\alpha + \theta_i) \sin(\beta + \eta_i)
\]

\[
\frac{\partial g_{\rho_i}}{\partial \rho_i} = 0
\]

The elements of the matrix \( G_{i,i} \) are now shown. These are the partial derivatives of the landmark \( i \) states with respect to the landmark \( i \) states.

\[
\frac{\partial g_{\theta_i}}{\partial \theta_i} = 1 - \Delta t p \sin(\alpha + \theta_i) \tan(\beta + \eta_i) + \Delta t \cos(\alpha + \theta_i) \sec(\beta + \eta_i) (q \sin(\beta + \eta_i) + \rho_i V)
\]

\[
\frac{\partial g_{\eta_i}}{\partial \eta_i} = \Delta t \sec^2(\beta + \eta_i) (\sin(\alpha + \theta_i) (\rho_i V \sin(\beta + \eta_i) + q) + p \cos(\alpha + \theta_i))
\]

\[
\frac{\partial g_{\eta_i}}{\partial \rho_i} = \Delta t V \sin(\alpha + \theta_i) \sec(\beta + \eta_i) \Delta t V \sin(\alpha + \theta_i) \sec(\beta + \eta_i)
\]

\[
\frac{\partial g_{\eta_i}}{\partial \theta_i} = -\Delta t (\sin(\alpha + \theta_i) (\rho_i V \sin(\beta + \eta_i) + q) + p \cos(\alpha + \theta_i))
\]
\[ \frac{\partial g_{\eta_i}}{\partial \eta_i} = 1 + \Delta t \rho_i V \cos(\alpha + \theta_i) \cos(\beta + \eta_i) \]

\[ \frac{\partial g_{\eta_i}}{\partial \rho_i} = \Delta t V \cos(\alpha + \theta_i) \sin(\beta + \eta_i) \]

\[ \frac{\partial g_{\rho_i}}{\partial \theta_i} = -\Delta t \rho_i^2 V \sin(\alpha + \theta_i) \cos(\beta + \eta_i) \]

\[ \frac{\partial g_{\rho_i}}{\partial \eta_i} = -\Delta t \rho_i^2 V \cos(\alpha + \theta_i) \sin(\beta + \eta_i) \]

\[ \frac{\partial g_{\rho_i}}{\partial \rho_i} = 1 + 2 \Delta t \rho_i V \cos(\alpha + \theta_i) \cos(\beta + \eta_i) \]

The above equations define the elements of the motion model Jacobian, \( G_k \) at each time step \( k \). \( G_k \) is computed with the current states each step. The Jacobian is then used in Equation 3.2 to propagate the covariance.

### 3.5 Measurement Equations

A camera is used to track the goal point and the landmarks in the image. The camera provides two measurements for the goal point and two for each landmark. The two measurements are an \( x \) and \( y \) pixel location within the image. This section will derive the measurement equations for the image pixel locations in terms of the goal states and the landmark states. The camera is mounted to the vehicle with a known position and orientation.

#### 3.5.1 Camera Coordinate Frames

There are four additional coordinate frames that are introduced to help in the derivation of the measurement equations for the camera. These coordinate frames will be described below.
Camera-1 Frame $\mathcal{F}^c_1$ 

The camera-1 frame, $\mathcal{F}^c_1$ is a coordinate frame with the origin at the center of the camera. This frame is aligned with $\mathcal{F}^b$. This frame is used for translation of the camera within $\mathcal{F}^b$.

Camera-2 Frame $\mathcal{F}^c_2$ 

The camera-2 frame, $\mathcal{F}^c_2$ has the same origin as $\mathcal{F}^c_1$ but this frame has been rotated about the $k^c_1$-axis through the angle $\psi_c$. The angle $\psi_c$ defines the yaw angle with respect to frame $\mathcal{F}^b$. This is a fixed value that is measured upon mounting the camera to the robot. The rotation matrix from $\mathcal{F}^c_1$ to $\mathcal{F}^c_2$ is given by

$$ R^c_2 = \begin{bmatrix} \cos \psi_c & \sin \psi_c & 0 \\ -\sin \psi_c & \cos \psi_c & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.36) $$

Camera Frame $\mathcal{F}^c$ 

The camera frame, $\mathcal{F}^c$ has the same origin as $\mathcal{F}^c_2$ but frame $\mathcal{F}^c$ is rotated to the traditional orientation of a 3D camera coordinate system where the $z$-axis points out the lens of the camera passing through the focal center. This frame is therefore rotated so that $i^c = -j^c_2$, $j^c = -k^c_2$ and $k^c = i^2_2$. The rotation matrix from $\mathcal{F}^c_2$ to $\mathcal{F}^c$ is given by

$$ R^c_2 = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}. \quad (3.37) $$

Image Frame $\mathcal{F}^{im}$ 

The image frame $\mathcal{F}^{im}$ is a 2D coordinate frame with origin at the top left corner of the image plane. The $i^{im}$-axis points to the right along the top edge of the image and the $j^{im}$-axis points down the image along the left edge.
3.5.2 Camera Model

The position of the camera in frame $\mathcal{F}^b$ is $p_c^b$. The components of the camera position will be written as

$$p_c^b = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}.$$

The camera is mounted with angle $\psi_c$ which defines the rotation from $\mathcal{F}^c_1$ to $\mathcal{F}^c_2$. The measurement equations are derived using the pin-hole camera model. This model is used extensively and is described in [45]. Let $q$ be a 3D point defined in frame $\mathcal{F}^c$ as

$$p_q^c = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}.$$

The point $q$ maps into the image plane through the following equations:

$$x_{qm} = f_x \left( \frac{q_x}{q_z} \right) + c_x \quad (3.38)$$

and

$$y_{qm} = f_y \left( \frac{q_y}{q_z} \right) + c_y. \quad (3.39)$$

The value $f_x$ and $f_y$ are the focal lengths in the $x$ and $y$ directions. The focal lengths are a function of the actual physical focal length and the actual size of the image element [45]. The units of $f_x$ and $f_y$ are in pixels. The parameters $c_x$ and $c_y$ are offsets used to locate the center of the image in frame $\mathcal{F}^{im}$ and are also expressed in units of pixels.

The measurement equations are derived by calculating the position of the goal and the positions of the landmarks in frame $\mathcal{F}^c$ from the current states and using Equations 3.38 and 3.39 to convert the positions to the expected pixel locations $x$ and $y$. 
Goal Point

Let the goal point be represented in frame $F^b$ by using the vector $p^b_g$. The position in frame $F^b$ was defined in Equation 3.8. The goal point can also be represented by adding two vectors as

$$p^b_g = p^b_c + p^b_{cg}. \quad (3.40)$$

The vector $p^b_{cg}$ is the vector from the camera location to the goal point location. Because frame $F^{c1}$ is aligned with frame $F^b$ and they only differ by a translation, a vector can be represented in the two frames with equal components, therefore, the following equation is equivalent to Equation 3.40

$$p^{c1}_g = p^{c1}_c + p^{c1}_{cg}. \quad (3.41)$$

The goal point is transformed into frame $F^c$ so that Equations 3.38 and 3.39 can be used. The rotation matrix $R^{c2}_{c2}$ is used to transform Equation 3.41 into frame $F^c$, therefore the rotation is defined as

$$R^{c1}_{c1} = R^{c2}_{c2} R^{c1}_{c2}.$$

This gives the following rotation

$$R^{c2}_{c2} = \begin{bmatrix}
\sin \psi_c & -\cos \psi_c & 0 \\
0 & 0 & -1 \\
\cos \psi_c & \sin \psi_c & 0
\end{bmatrix}.$$

The goal point in frame $F^c$ is defined as

$$p^{c}_{cg} = R^{c1}_{c1} (p^{c1}_{g} - p^{c1}_c), \quad (3.42)$$

which when multiplied out gives

$$p^{c}_{cg} = \begin{bmatrix}
-\frac{\cos(\beta) \sin(\alpha - \psi_c)}{\rho_g} - x_c \sin(\psi_c) + y_c \cos(\psi_c) \\
-\frac{\sin(\beta)}{\rho_g} z_c \\
\frac{\cos(\beta) \cos(\alpha - \psi_c)}{\rho_g} - x_c \cos(\psi_c) - y_c \sin(\psi_c)
\end{bmatrix}. \quad (3.43)$$
The expected measurement for the goal point is then computed by substituting the associated components of Equation 3.43 into Equations 3.38 and 3.39. This gives the following measurement equations

\[
\begin{bmatrix}
    h_{x} \\
    h_{y}
\end{bmatrix}
= \begin{bmatrix}
    h_{x} \\
    h_{y}
\end{bmatrix} = \begin{bmatrix}
    f_x \left( \cos(\beta) \sin(\alpha - \psi_c) - x_c \sin(\psi_c) + y_c \cos(\psi_c) \right) + c_x \\
    p_x \left( \cos(\beta) \cos(\alpha - \psi_c) - x_c \cos(\psi_c) - y_c \sin(\psi_c) \right) \\
    f_y \left( z_c - \sin(\beta) \right) + c_y \\
    p_y \left( \cos(\beta) \cos(\alpha - \psi_c) - x_c \cos(\psi_c) - y_c \sin(\psi_c) \right) + c_y
\end{bmatrix}.
\] (3.44)

**Landmark Points**

The derivation of the landmark measurement equations are similar to that of the goal point. The difference is in the definition of the landmark position in frame \( \mathcal{F}^b \), which for a landmark is defined in Equation 3.10. Let the position of landmark \( i \) in frame \( \mathcal{F}^b \) be defined as \( \mathbf{p}_i^b \) and let the vector from the camera to landmark \( i \) in frame \( \mathcal{F}^b \) be defined as \( \mathbf{p}_{cl,i}^b \). Similar to Equation 3.42 the relative vector between the camera and landmark \( i \) is

\[
\mathbf{p}_{cl,i}^c = R_{c1}^c \left( \mathbf{p}_{i}^c - \mathbf{p}_1^c \right)
\]

which is

\[
\mathbf{p}_{cl,i}^c = \begin{bmatrix}
    -\cos(\beta_i) \sin(\alpha - \psi_i) - x_c \sin(\psi_c) + y_c \cos(\psi_c) \\
    \zeta_c - \frac{\sin(\beta_i + \eta_i)}{\rho_i} \\
    \cos(\beta_i) \cos(\alpha - \psi_i) - x_c \cos(\psi_c) - y_c \sin(\psi_c)
\end{bmatrix}.
\] (3.45)

The expected measurement for landmark \( i \) is then computed by plugging the components of Equation 3.45 into Equations 3.38 and 3.39 which gives the following landmark measurement equations

\[
\begin{bmatrix}
    h_{i,x} \\
    h_{i,y}
\end{bmatrix}
= \begin{bmatrix}
    h_{i,x} \\
    h_{i,y}
\end{bmatrix} = \begin{bmatrix}
    f_x \left( -\cos(\beta_i) \sin(\alpha - \psi_i) - x_c \sin(\psi_c) + y_c \cos(\psi_c) \right) + c_x \\
    p_x \left( -\cos(\beta_i) \cos(\alpha - \psi_i) - x_c \cos(\psi_c) - y_c \sin(\psi_c) \right) \\
    f_y \left( z_c - \frac{\sin(\beta_i + \eta_i)}{\rho_i} \right) + c_y \\
    p_y \left( -\cos(\beta_i) \cos(\alpha - \psi_i) - x_c \cos(\psi_c) - y_c \sin(\psi_c) \right) + c_y
\end{bmatrix}.
\] (3.46)
The final measurement vector is found by stacking Equation 3.44 with the proper number of landmark measurement equations from Equation 3.46 as

\[
h = \begin{bmatrix}
h_g \\
h_{l_1} \\
h_{l_2} \\
\vdots \\
h_{l_N}
\end{bmatrix}.
\]

### 3.5.3 Measurement Jacobian

The measurement Jacobian \( H \) is used in the measurement update equations for both the state update and the covariance update as shown in Equations 3.4 and 3.5. The measurement Jacobian is the linearization of the measurement vector evaluated at the current states and is defined as

\[
H = \frac{\partial h}{\partial \mathbf{x}}. \tag{3.47}
\]

The measurement Jacobian \( H \) has a similar form to that of the motion model Jacobian \( G \) as

\[
H = \begin{bmatrix}
H_g & 0 & 0 & \cdots & 0 \\
H_{1,g} & H_{1,1} & 0 & \cdots & 0 \\
H_{2,g} & 0 & H_{2,2} & 0 & 0 \\
\vdots & \vdots & 0 & \ddots & 0 \\
H_{N,g} & 0 & \cdots & 0 & H_{N,N}
\end{bmatrix},
\]

where

\[
H_g = \frac{\partial h_g}{\partial \mathbf{x}_g} = \begin{bmatrix}
\frac{\partial h_{g_x}}{\partial \alpha} & \frac{\partial h_{g_x}}{\partial \beta} & \frac{\partial h_{g_x}}{\partial \rho_g} \\
\frac{\partial h_{g_y}}{\partial \alpha} & \frac{\partial h_{g_y}}{\partial \beta} & \frac{\partial h_{g_y}}{\partial \rho_g}
\end{bmatrix},
\]

\[
H_{i,g} = \frac{\partial h_{l_{i,x}}}{\partial \mathbf{x}_g} = \begin{bmatrix}
\frac{\partial h_{l_{i,x}}}{\partial \alpha} & \frac{\partial h_{l_{i,x}}}{\partial \beta} & \frac{\partial h_{l_{i,x}}}{\partial \rho_g} \\
\frac{\partial h_{l_{i,y}}}{\partial \alpha} & \frac{\partial h_{l_{i,y}}}{\partial \beta} & \frac{\partial h_{l_{i,y}}}{\partial \rho_g}
\end{bmatrix},
\]
and

\[ H_{l,i} = \frac{\partial h_{l,i}}{\partial x_i} = \begin{bmatrix} \frac{\partial h_{l,i}}{\partial \theta_i} & \frac{\partial h_{l,i}}{\partial \eta_i} & \frac{\partial h_{l,i}}{\partial \eta_i} \end{bmatrix}. \]

The individual results for the partial derivatives for the measurement Jacobian are first given for \( H_g \) which are the partial derivatives of the goal measurement equation \( h_g \) with respect to the goal states \( x_g \).

\[
\frac{\partial h_{g_x}}{\partial \alpha} = \frac{\cos(\beta) f_x (\cos(\beta) + \cos(\alpha)x_c \rho_g + \sin(\alpha)y_c \rho_g)}{p_x (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{g_x}}{\partial \beta} = \frac{\sin(\beta) f_x \rho_g (\cos(\alpha)y_c - \sin(\alpha)x_c)}{p_x (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{g_x}}{\partial \rho_g} = \frac{\cos(\beta) f_x (\cos(\alpha)y_c - \sin(\alpha)x_c)}{p_x (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{g_x}}{\partial \alpha} = \frac{\cos(\beta) f_y (\sin(\alpha - \psi_c) (z_c \rho_g - \sin(\beta)))}{p_y (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{g_y}}{\partial \beta} = \frac{f_y (\cos(\beta)x_c \rho_g \cos(\psi_c) + \cos(\beta)y_c \rho_g \sin(\psi_c) + \cos(\alpha - \psi_c) (\sin(\beta)z_c \rho_g - 1))}{p_y (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{g_y}}{\partial \rho_g} = \frac{-f_y (\sin(\beta)x_c \cos(\psi_c) + \sin(\beta)y_c \sin(\psi_c) - \cos(\beta)z_c \cos(\alpha - \psi_c))}{p_y (-\cos(\beta) \cos(\alpha - \psi_c) + x_c \rho_g \cos(\psi_c) + y_c \rho_g \sin(\psi_c))^2},
\]

Lastly, the values used in \( H_{l,g} \) are given next. These are the partial derivatives of the landmark \( i \) measurements \( h_{l_i} \) with respect to the goal states \( x_g \).

\[
\frac{\partial h_{l_i,x}}{\partial \alpha} = -\frac{f_x \cos(\beta + \eta) (\cos(\beta + \eta) - x_c \rho_L \cos(\alpha + \theta) - y_c \rho_L \sin(\alpha + \theta))}{p_x (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2},
\]

\[
\frac{\partial h_{l_i,x}}{\partial \beta} = -\frac{f_x \rho_L \sin(\beta + \eta) (x_c \sin(\alpha + \theta) - y_c \cos(\alpha + \theta))}{p_x (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}.
\]
\frac{\partial h_{i,x}}{\partial \rho_g} = 0

\frac{\partial h_{i,y}}{\partial \alpha} = \frac{f_y \cos(\beta + \eta) \sin(\alpha - \psi_c + \theta) (z_c \rho_L - \sin(\beta + \eta))}{p_y (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2} + \frac{f_y (\cos(\alpha - \psi_c + \theta) (z_c \rho_L \sin(\beta + \eta) - 1))}{p_y (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}

\frac{\partial h_{i,y}}{\partial \rho_g} = 0

The values used in \(H_{i,i}\) are shown below. These are the partial derivatives of the landmark \(i\) measurements \(h_{i,i}\) with respect to the landmark \(i\) states \(x_i\).

\frac{\partial h_{i,x}}{\partial \theta_i} = -\frac{f_x \cos(\beta + \eta) (\cos(\beta + \eta) - x_c \rho_L \cos(\alpha + \theta) - y_c \rho_L \sin(\alpha + \theta))}{p_x (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}

\frac{\partial h_{i,x}}{\partial \eta_i} = -\frac{f_x \rho_L \sin(\beta + \eta) (x_c \sin(\alpha + \theta) - y_c \cos(\alpha + \theta))}{p_x (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}

\frac{\partial h_{i,x}}{\partial \rho_i} = -\frac{f_x \cos(\beta + \eta) (x_c \sin(\alpha + \theta) - y_c \cos(\alpha + \theta))}{p_x (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}

\frac{\partial h_{i,y}}{\partial \theta_i} = \frac{f_y \cos(\beta + \eta) \sin(\alpha - \psi_c + \theta) (z_c \rho_L - \sin(\beta + \eta))}{p_y (-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2}
\[ \frac{\partial h_{i,y}}{\partial \eta_i} = \frac{f_y(x_c \rho_L \cos(\beta + \eta) \cos(\psi_c) + y_c \rho_L \cos(\beta + \eta) \sin(\psi_c))}{p_y(-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2} + \frac{f_y(\cos(\alpha - \psi_c + \theta) (z_c \rho_L \sin(\beta + \eta) - 1))}{p_y(-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2} \]

\[ \frac{\partial h_{i,y}}{\partial \rho_i} = -\frac{f_y(x_c \sin(\beta + \eta) \cos(\psi_c) + y_c \sin(\beta + \eta) \sin(\psi_c) - z_c \cos(\beta + \eta) \cos(\alpha - \psi_c + \theta))}{p_y(-\cos(\beta + \eta) \cos(\alpha - \psi_c + \theta) + x_c \rho_L \cos(\psi_c) + y_c \rho_L \sin(\psi_c))^2} \]

The above equations define the elements of the measurement Jacobian, \( H_k \) at each time step \( k \). This is computed with the current states. This is then used in Equation 3.3 to compute the Kalman gain of the EKF and is also used in Equation 3.5 to update the filter covariance.

### 3.6 Summary

This chapter has described the goal-based robocentric mapping algorithm. A novel approach to the state definitions was explained showing how the goal is the anchor of the local map that is created. Each additional landmark is referenced to the goal location. The motion model for the states was derived in the body frame. The motion model was derived for 3D motion of the robot. The measurement equations for a monocular camera with offset mounting location and yaw angle with respect to the robot were derived. Chapter 4 will analyze the designed EKF mapping algorithm by looking into the observability and consistency. This method will also be compared against the robocentric mapping algorithm in [50]. Simulation results will also be shown for 3D motion of the robot.
CHAPTER 4. MAPPING ANALYSIS

4.1 Introduction

The EKF mapping algorithm was derived in Chapter 3. This chapter will analyze the performance of the designed mapping algorithm. First, the observability of the system will be described. Second, the consistency of the EKF mapping algorithm will be analyzed through Monte Carlo simulation results. Third, a comparison of the EKF Mapping method from this work will be compared with the work in [50]. The chapter will conclude with simulation results of the mapping algorithm in a 3D environment.

4.2 Observability

The observability of this system is calculated using the observability analysis developed in [54]. This analysis uses the continuous-time system definition. A brief explanation of the observability criteria is given here. The local weak observability analysis is used because the camera is essentially a bearings-only measurement device, therefore, the depth of the landmarks cannot be estimated with a single measurement. The vehicle must have some motion for observability to be possible. This will be shown here. Let a system $\Sigma$ be defined as

$$
\dot{x} = f(x,u) = \left[ f_g^T(x_g,u), f_{l,1}^T(x_{l,1},u), \ldots, f_{l,N}^T(x_{l,N},u) \right]^T
$$

$$
y = h(x) = \left[ h_g^T(x_g), h_{l,1}^T(x), \ldots, h_{l,N}^T(x) \right]
$$

where the state $x \in \mathbb{R}^{3(N+1)}$ and the measurement vector $h(x) \in \mathbb{R}^{2(N+1)}$. The value $N$ is the number of landmarks and there will always be one goal point. The functions $f_g$ and $f_{l,i}$ are defined in Equations 3.24 and 3.34 respectively. The measurement functions $h_g$ and $h_{l,i}$ are defined in Equations 3.44 and 3.46 respectively.
4.2.1 Lie Derivatives

Lie derivatives are used in the observability analysis and will be briefly described here following [55]. Let the function $f : \mathbb{R}^n \to \mathbb{R}^n$ be a vector field in $\mathbb{R}^n$ and let the function $h : \mathbb{R}^n \to \mathbb{R}$ be a scalar function. The Lie derivative of $h$ with respect to $f$ can be calculated as

$$L_f h = \nabla h f = \frac{\partial h}{\partial x} f.$$ 

The Lie derivative evaluates the change in the function $h$ with respect to the function $f$. The zeroth-order Lie derivative of a function $h$ is the function itself as

$$L_f^0 h = h.$$ 

Higher order Lie derivatives are computed recursively as

$$L_f^1 h = \nabla L_f^0 h f$$

and

$$L_f^2 h = \nabla L_f^1 h f$$

such that the $n^{th}$ Lie derivative is

$$L_f^n h = \nabla L_f^{n-1} h f.$$ 

Using these definitions of Lie derivatives they can be stacked to form

$$l(x) = \begin{bmatrix} L_f^0 h \\ L_f^1 h \\ \vdots \\ L_f^{n-1} h \end{bmatrix}.$$ 

The observability matrix $\mathcal{O}$ about the local point $x_0$ is then defined as

$$\mathcal{O} = \frac{\partial l(x)}{\partial x} \bigg|_{x=x_0}$$ 

(4.2)
The observability of the system is then found by using the observability rank condition as described in [54]. This says that if the rank of $\mathcal{O}(x_0) = n$ then the system is locally weakly observable at the point $x = x_0$.

**Goal Point Observability**

The observability of the goal point states will first be shown and this will then be extended to the other landmark states. The system with only goal states to be examined for observability is

$$\Sigma_g : \dot{x} = f(x, u) = f_g(x, u),$$

$$y = h(x) = h_g(x).$$

The system $\Sigma_g$ is locally weakly observable if there is a point $x_0$ that is distinguishable from all other points within the neighborhood of $x_0$. The point $x_0$ is distinguishable from all other points if the rank $\mathcal{O}(x_0) = 3$ where $\mathcal{O}$ is defined in Equation 4.2. As will be shown below, for the system $\Sigma_g$ in this analysis, only the zeroth-order and first-order Lie derivatives will be necessary. The zeroth-order Lie derivative (or the function $h$) is used to calculate the information $h$ gives about the states $x$ under no motion. The first-order Lie derivative is used to calculate how the function $h$ changes under first-order motion using the function $f$. In other words, what does the function $h$ tell us about the state $x$ when the vehicle is moving as described by the function $f$. This is shown mathematically by

$$\dot{h} = \frac{\partial h}{\partial x} \frac{\partial x}{\partial t}$$

and by noting that $\dot{x} = f(x)$, therefore

$$\dot{h} = \frac{\partial h}{\partial x} f = L^1_f h.$$ 

The state $x$ in the system $\Sigma$ for the goal point only is

$$x = \begin{bmatrix} \alpha \\ \beta \\ \rho_g \end{bmatrix}$$
and the measurement model is defined as

\[
h = \begin{bmatrix} h_{g,x} \\ h_{g,y} \end{bmatrix}.
\]

Without loss of generality, let the camera be located at the center of frame \( \mathcal{F}^b \) with frame \( \mathcal{F}^c_2 \) aligned with frame \( \mathcal{F}^b \). Also let the image be square with no image offset, \( c_x = c_y = 0 \) and \( f_x = f_y = f \) and let the parameters \( p_x = p_y = 1 \). Using these assumptions there is no change in the motion model in Equation 3.33. The simplified measurement equation is found by using these assumptions in Equation 3.44 which gives the following equation

\[
h_g = \begin{bmatrix} -f \tan \alpha \\ -f \sec \alpha \tan \beta \end{bmatrix},
\]

which means that the zeroth-order Lie derivatives are

\[
L^0_f h_{g,x} = -f \tan \alpha
\]

and

\[
L^0_f h_{g,y} = -f \sec \alpha \tan \beta.
\]

The first-order Lie derivatives are

\[
L^1_f h_{g,x} = -f \sec^2 \alpha \left( V \sec \beta \sin \alpha \rho_g + p \cos \alpha \tan \beta + q \sin \alpha \tan \beta - r \right)
\]

and

\[
L^1_f h_{g,y} = -f \sec \alpha \sec^2 \beta \left( V \cos \alpha \sin \beta \rho_g - p \sin \alpha + q \cos \alpha \right) - f \sec \alpha \tan \alpha \tan \beta \left( V \sec \beta \sin \alpha \rho_g + p \cos \alpha \tan \beta + q \sin \alpha \tan \beta - r \right).
\]
The observability matrix $\mathcal{O}$ is then formed by stacking the rows of partial derivatives of the Lie derivatives as described in Equations 4.1 and 4.2. The observability becomes

$$\mathcal{O} = \begin{bmatrix}
\frac{\partial L^0_h}{\partial x} \\
\frac{\partial L^0_h}{\partial y} \\
\frac{\partial L^1_h}{\partial x} \\
\frac{\partial L^1_h}{\partial y}
\end{bmatrix},$$

where the rows of the observability matrix are given in the equations below.

$$\frac{\partial L^0_h}{\partial x} = \begin{bmatrix}
-f \sec^2 \alpha & 0 & 0
\end{bmatrix} \tag{4.3}$$

$$\frac{\partial L^0_h}{\partial y} = \begin{bmatrix}
-f \sec \alpha \tan \alpha \tan \beta & -f \sec \alpha \sec^2 \beta & 0
\end{bmatrix}$$

$$\frac{\partial L^1_h}{\partial x} = \begin{bmatrix}
-f \sec \alpha \left( V \sec \beta \rho_g \left( 1 + 2 \tan^2 \alpha \right) - 2 r \sec \alpha \tan \alpha \right) - f \sec \alpha \tan \beta \left( p \tan \alpha + q \left( 1 + 2 \tan^2 \alpha \right) \right) \\
-f \sec \alpha \sec \beta \left( p + q \tan \alpha \right) + V \rho_g \tan \alpha \tan \beta \\
-f V \sec \alpha \sec \beta \tan \alpha
\end{bmatrix} \tag{4.4}$$

$$\frac{\partial L^1_h}{\partial y} = \begin{bmatrix}
f \sec \alpha \left( p \sec \alpha + r \left( 2 \sec^2 \alpha - 1 \right) \tan \beta \right) - 2 \sec \alpha \sec \beta \left( V \rho_g + q \sin \beta \right) \tan \alpha \tan \beta \\
f \sec^2 \alpha \sec^2 \beta \left( r \cos \beta \sin \alpha - 2 q \sin \beta + \frac{1}{2} V \rho_g \left( \cos \left( 2 \beta \right) - 3 \right) \right) \\
f V \sec^2 \alpha \sec \beta \tan \beta
\end{bmatrix} \tag{4.5}$$

Now, to show that the system $\Sigma_g$ is locally weakly observable, the rows of matrix $\mathcal{O}$ are inspected to find when the matrix is full rank. This is similar to the discussion of observability.
in [50]. As would be expected, if $V = 0$ then the system is unobservable. This is shown noting that the entire third column of $\mathcal{O}$ is 0. When $V = 0$ there is no forward motion of the vehicle, therefore, there is no way to triangulate the position of the goal from successive measurements. There are also restrictions on the states $\alpha$ and $\beta$. To find what these restrictions are, Equations 4.4 and 4.5 are used. In particular, the last element in each of those rows is inspected. The third element is the element that describes the observability of the state $\rho_g$. Because of the terms $\tan \alpha$ and $\tan \beta$ in these elements, the third column of the observability matrix would be 0 if $\alpha = \beta = 0$, therefore, the system is unobservable when $\alpha = 0$ and $\beta = 0$ simultaneously. This happens when the vehicle is driving directly toward the goal point (traveling along the vector that connects the origin of $\mathcal{F}^b$ to the goal point). Because the goal point is perfectly in line with the vehicle, motion of the vehicle will not change the measurement that the vehicle receives and no information about the state $\rho_g$ can be gained. It is interesting to note that when $\alpha = 0$ Equation 4.5 is used to keep the system observable by having a nonzero third element in the row as long as $\beta \neq 0$. Likewise, when $\beta = 0$ then Equation 4.4 is used to keep the system observable in the same manner as long as $\alpha \neq 0$. It could also be shown [50] that the system can be observable when $\alpha = \beta = 0$ as long as $r \neq 0$. This requires the use of second-order Lie derivatives which are not shown here.

**Observability of Landmarks**

The above discussion of observability for the system $\Sigma_g$ was for a goal point only, therefore, the observability of additional landmarks is now shown. To do this, an intermediate step is used by analyzing a system made up of a goal and a single landmark with a slightly different state definition. The observability of this modified system is then used to prove the stability of the normal system with a goal and one landmark. The observability of the system with a goal and a single landmark is then extended to a system with $N$ landmarks.

Let a new state representation be defined for landmark $i$ as

$$
\mathbf{x}_i^* = \begin{bmatrix} 
\gamma_i \\
\lambda_i \\
\rho_i 
\end{bmatrix},
$$

(4.6)
where $\gamma_i = \alpha + \theta_i$ as defined in Equation 3.25 and $\lambda_i = \beta + \eta_i$ as defined in Equation 3.26. The angles $\gamma_i$ and $\lambda_i$ for landmark $i$ are equivalent to the angles $\alpha$ and $\beta$ respectively for the goal. This state definition can be seen back in Figure 3.5.

Now a system $\Sigma_1^*$ is analyzed for observability. This system has one goal point and one landmark as defined in Equation 4.6. Let the system be defined as

$$
\Sigma_1^* : \\
\dot{x} = f(x, u) = \begin{bmatrix} f_g^T(x_g, u), f_l^T(x_{l_1}^*, u) \end{bmatrix}^T \\
y = h(x) = \begin{bmatrix} h_g^T(x_g), h_l^T(x_{l_1}^*) \end{bmatrix}^T,
$$

where the function $f_g$ is used for both the goal and the landmark because the definitions of $\gamma_1$ and $\lambda_1$ are equivalent to $\alpha$ and $\beta$ so the motion model is the same for both sets of states, therefore, passing in the argument $x_{l_1}^*$ into $f_g$ will update the landmark states properly. The function $h_g$ is used for both, for the same reason. The observability of this system is analyzed by using the fact that the goal and landmark states are completely independent using the landmark state definition of Equation 4.6. This means that for the system $\Sigma_1^*$ to be observable both the goal and the landmark must be observable. The conditions for observability for the goal have already been shown. Because of the equivalent state definition in Equation 4.6 for the landmark, the landmark is observable under the same conditions, which are

\[
V \neq 0, \\
\gamma_1 \neq 0 \text{ if } \lambda_1 = 0, \\
\lambda_1 \neq 0 \text{ if } \gamma_1 = 0.
\]

Now the proposed system of this work with a goal with states as defined in Equation 3.7 and a single landmark with states defined in Equation 3.9 is analyzed for observability. This system is

$$
\Sigma_1 : \\
\dot{x} = f(x, u) = \begin{bmatrix} f_g^T(x_g, u), f_l^T(x_{l_1}, u) \end{bmatrix}^T \\
y = h(x) = \begin{bmatrix} h_g^T(x_g), h_l^T(x_{l_1}) \end{bmatrix}^T,
$$

where the functions $f_g$ and $f_l$ are the equations of motion defined in Equation 3.33 and Equation 3.34 respectively.
The same conditions under which the system $\Sigma^*$ is observable also apply to the system $\Sigma_1$. The conditions are

\begin{align*}
\alpha \neq 0 & \text{ if } \beta = 0 \\
\beta \neq 0 & \text{ if } \alpha = 0 \\
\theta_i \neq -\alpha_i & \text{ if } \eta_i = -\beta_i \\
\eta_i \neq -\beta_i & \text{ if } \theta_i = -\alpha_i \\
V & \neq 0
\end{align*}

As expected, this means that the vehicle has to be moving and that the vehicle cannot be traveling directly along one of the vectors from the origin of $\mathcal{F}^b$ to the goal or the landmark. Now the final system which is the combined system of a goal and $N$ landmarks is analyzed. This system is defined as

$$
\begin{align*}
\dot{x} &= f(x, u) = \left[ f^T_g(x, u), f^T_1(x, u), f^T_2(x, u), \ldots, f^T_N(x, u) \right]^T \\
y &= h(x) = \left[ h^T_g(x), h^T_1(x), h^T_2(x), \ldots, h^T_N(x) \right]^T
\end{align*}
$$

The observability of the system $\Sigma$ follows the same conditions that exist for $\Sigma_1$ but the conditions apply to each additional landmark. The conditions for the goal remain the same and each additional landmark will be observable as long as the vehicle is not driving directly toward the landmark. This is summarized by the following conditions:

\begin{align*}
\alpha \neq 0 & \text{ if } \beta = 0 \\
\beta \neq 0 & \text{ if } \alpha = 0 \\
\theta_i \neq -\alpha_i & \text{ if } \eta_i = \beta \forall \ i \in N \\
\eta_i \neq -\beta_i & \text{ if } \theta_i = \alpha \forall \ i \in N \\
V & \neq 0.
\end{align*}

As expected this shows that for the system $\Sigma$ to be observable, the vehicle cannot be traveling directly towards the goal or any of the landmarks. If the robot is traveling directly towards any landmark then no information about the depth for that landmark is available and the depth state for that landmark is not observable.
4.3 Consistency of EKF

In this section the consistency of the EKF used for robocentric mapping will be analyzed. The consistency of a dynamic estimator is discussed in [56] and the main points from that work are noted here. There are two main criteria for consistency:

1. The estimation errors should be zero mean

2. The filter covariance should accurately reflect the actual estimation error distribution

Criterion 1 means the estimator should be unbiased. An unbiased estimator is described by the following equation:

$$E [x (k) - \hat{x} (k|k)] = 0.$$  \hspace{1cm} (4.7)

Criterion 2 means the covariance of the estimation errors should be smaller than the covariance that is computed by the filter [57].

The world-centric EKF SLAM algorithm has been shown to be inconsistent [23], [57], [48]. An EKF derived in a local frame, however, can improve the consistency of the filter by reducing heading uncertainty, which is the major cause of inconsistency in the world-centric EKF method. The work in [50] mentions that an EKF in the robot frame can help with the inconsistencies, however, there is no consistency analysis of the proposed method shown.

The method used to analyze the consistency of the estimator is described in [56] and [23] and is also described here. The test uses $N_m$ Monte Carlo simulations of the vehicle driving through an environment with static landmarks. Each run produces an independent state vector $\hat{x}_i$, with $i = 1, 2, \ldots, N_m$. The true states $x_i$ for each run will also be known from the simulation, but the true probability density function of the errors $(x_i - \hat{x}_i)$ is not known. The filter performance is then measured by using the normalized estimation error squared (NEES) which at each time step $k$ is defined as

$$\varepsilon_k = (x_k - \hat{x}_{k|k})^T P_k^{-1} (x_k - \hat{x}_{k|k}).$$

The test assumes that the estimation errors are consistent and approximately Gaussian. Under this assumption, the single run NEES will be chi-square distributed with the degrees of freedom being $\text{dim} (x_k)$. The overall filter performance is tested by taking the average NEES value over $N_m$,
independent simulations as

$$\bar{e} = \frac{1}{N_m} \sum_{i=1}^{N_m} e_k^i.$$ 

The quantity $N_m \cdot \varepsilon_k$ is also chi-square distributed but the degree of freedom for this distribution is now $N_m \cdot \text{dim}(x_k)$. The value of $\bar{e}_k$ is then checked at each time step against the 95% confidence interval.

The confidence intervals are calculated with the total degree of freedom of the simulation, which is the number of degrees of freedom for the system, $\text{dim}(x_k)$, multiplied by the number of Monte Carlo runs $N_m$. The total degree of freedom is then used to find the critical value from the chi-square distribution. The intervals are then normalized by dividing by the number of Monte Carlo runs. As described in [56] this is for convenience which allows the results of the Monte Carlo runs to be in the neighborhood of $\text{dim} \cdot (x_k)$ independent of the number of Monte Carlo runs ($N_m$) used.

Simulation results are presented for 2D robot motion and for 3D robot motion. The landmarks used in each case are 3D. For the 2D robot motion case, the robot is restricted to motion in the $x$-$y$ plane with $\phi = \theta = 0$. The motion of the robot in the 3D case uses non-zero $p$ and $q$ to produce roll and pitch while driving. In each case the same environment is used with static landmarks. Two paths for the 2D case and the 3D case are run. In the first path the robot can see the goal in the image the entire run while in the second path the robot loses the goal in the image for some duration of time. The 2D results of this work are used to compare against the baseline consistency check described in [23] and [50]. For the consistency analysis, the algorithm is using all of the landmarks that are seen in the camera view. As new landmarks enter the image they are added to the state vector, however, only the three states associated with the goal are used to calculate the NEES value in Equation 4.7. This means that the degrees of freedom for this analysis is 150. With $N_m = 50$ and $\text{dim}(x_k) = 3$ the 95% confidence interval is $[2.36, 3.72]$.

The landmark positions are the same for each simulation run. The only changes in each simulation run are the noise vectors used to add noise to the vehicle inputs. The standard deviation values on the vehicle inputs ($\sigma_v, \sigma_p, \sigma_q, \sigma_r$) and to the measurement noise ($\sigma_l, \sigma_f$) were constant across all simulations. For the 2D simulations $\sigma_p = \sigma_q = 0$. The noise is injected in the Matlab
simulation by using the \textit{randn} command in Matlab. Each individual simulation was run using a different seed to change the noise vector in the simulation. All other parameters were identical.

4.3.1 2D Simulation Results

This section contains the results of running the Monte Carlo simulations in a 2D Matlab environment. There are two different simulation setups used for the 2D analysis and the results from each setup will be presented here. Each 2D simulation uses the parameters described in Table 4.1.

Table 4.1: Parameters used in the 2D Monte Carlo simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Starting Position</td>
<td>((0, 0, 0)^\top) m</td>
</tr>
<tr>
<td>Vehicle Starting Angles ((\phi, \theta, \psi))</td>
<td>((0, 0, -15)) deg</td>
</tr>
<tr>
<td>Goal Position</td>
<td>((300, -120, 10)^\top) m</td>
</tr>
<tr>
<td>Vehicle Velocity</td>
<td>1.0 m/s</td>
</tr>
<tr>
<td>Velocity Noise (\sigma_V)</td>
<td>0.2 m/s</td>
</tr>
<tr>
<td>Angular Velocity Noise (\sigma_r)</td>
<td>1.0 deg/s</td>
</tr>
<tr>
<td>Image Measurement Noise ((\sigma_{I_x}, \sigma_{I_y}))</td>
<td>((1, 1)) pixel</td>
</tr>
<tr>
<td>Simulation Time Step (\Delta t)</td>
<td>0.1 s</td>
</tr>
<tr>
<td>Simulation Duration</td>
<td>300 s</td>
</tr>
<tr>
<td>Number of Landmarks</td>
<td>20</td>
</tr>
<tr>
<td>Number of Monte Carlo Runs</td>
<td>50</td>
</tr>
</tbody>
</table>

Two different paths of the vehicle were tested and are described below. The first path allows the vehicle to see the goal the entire time while the second path forces the vehicle to turn such that the goal leaves the camera field of view, therefore no direct measurement of the goal is available.

Path 1 - No Goal Loss

The path overview for the 2D path with no goal loss is shown in Figure 4.1. The figure shows two views of the path, a 2D view in Figure 4.1(a) and a 3D view in Figure 4.1(b). The average error in the state \(\rho_g\) is shown in Figure 4.1(c) and the average error in depth is shown in Figure 4.1(d). For this path, the vehicle is able to see the goal in the camera image the entire time.
The path of the goal in the image is shown in Figure 4.2, which shows that the goal point is not moving much in the image frame because this is a 2D path and it is desired to keep the goal in the image the whole path. Figure 4.3 shows the value of $\bar{\epsilon}_k$ for the 50 Monte Carlo runs. The 95% confidence intervals of $[2.36, 3.72]$ are also plotted. The measure of consistency is not merely determined if the value of $\bar{\epsilon}_k$ leaves the bounds at any time but by how much the value of $\bar{\epsilon}_k$ is above the bounds. As mentioned in [23], if $\bar{\epsilon}_k$ rises significantly above the bound then the filter is considered to be optimistic, meaning the covariance computed by the filter is smaller than it should be. If the value of $\bar{\epsilon}_k$ is lower than the lower bound then the filter is conservative and the covariance is much larger than it should be. Another observation is the trend of the $\bar{\epsilon}_k$ value. In Figure 4.3 it is seen that it is not diverging over time which was shown to be the case in [23] for the EKF algorithm used in that work.

**Path 2 - Goal Loss**

The second path tested for consistency is shown in Figure 4.4. The path is only slightly different than the path shown in Figure 4.1(a) with the difference being the angular rate $r$ which causes the vehicle to turn enough to lose the goal two different times while driving. Figure 4.4(b) shows the 3D view of the path and the environment. Figure 4.4(c) shows the actual velocity and the measured velocity from a single run. Figure 4.4(d) shows the actual and measured angular velocity $r$ for a single run. The path of the goal in the image is shown in Figure 4.5. The image path is shown in blue when the goal is viewed by the camera and shown in red when outside the camera field of view. The goal leaves the field of view two separate times. Figure 4.6 shows the value $\bar{\epsilon}_k$ for the 50 Monte Carlo runs. As before, the 95% intervals are also shown for this test. The results of this test while losing the goal in the image are similar to the results from the path where the goal is not lost. The estimator remains consistent even while not seeing the goal in the camera. The trend of $\bar{\epsilon}_k$ is also similar, there is no diverging trend even when the goal is not seen as shown in Figure 4.6.
Figure 4.1: Path overview of the 2D path with no goal loss. The 2D path with landmarks and goal is shown in (a). The goal location is shown in red. The other landmarks are shown in black. A 3D view of the path and landmarks is shown in (b). This shows the differing elevation of the landmarks. The landmark positions do not change for each simulation run. The average error in the state $\rho_g$ is shown in (c). This also includes the average error bound at each time which is shown in red. The average error in distance is shown in (d). The average bound ate each time step is shown in red.
Figure 4.2: Path of the goal point in the image. As can be seen, the goal is visible within the image the entire run. The image boundary is shown by the bold black rectangle.

Figure 4.3: Average NEES value of 50 Monte Carlo simulations at each time step. The vehicle was able to see the goal in the camera image the whole run as seen in Figure 4.2.
Figure 4.4: Path overview of the 2D path that causes the goal to leave the camera field of view. The 2D path with landmarks and goal are shown in (a). The goal location is shown in red. The other landmarks are shown in black. A 3D view of the path and landmarks is shown in (b). This shows the differing elevation of the landmarks. The landmark positions do not change for each simulation run. The velocity is shown in (c). Both the true velocity and the sensed velocity are shown. The vehicle drives 1 m/s for each of the simulation runs but the noise on the sensed velocity changes with each run. The sensed velocity is plotted for a single run. The angular rate \( r \) is shown in (d). The true value and the sensed value of \( r \) are both shown.
Figure 4.5: Path of the goal point in the image. The goal is shown to leave the image and come back two separate times. When the goal is out of the image boundary it is shown in red. The image boundary is shown by the bold black rectangle.

Figure 4.6: Average NEES value of 50 Monte Carlo simulations at each time step. The vehicle lost the goal in the camera image two different times ($t = 54$ s and $t = 99$ s) along the path as shown in Figure 4.5.
4.3.2 3D Simulation Results

This section contains the results of running the Monte Carlo simulations in a 3D Matlab environment. There are two different simulation setups used for the 3D analysis and the results from each setup are presented here. Each 3D simulation uses the parameters described in Table 4.2.

Table 4.2: Parameters used in the 3D Monte Carlo simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Starting Position</td>
<td>$(0, 0, 0)^T \text{ m}$</td>
</tr>
<tr>
<td>Vehicle Starting Angles $(\phi, \theta, \psi)$</td>
<td>$(0, 0, -15) \text{ deg}$</td>
</tr>
<tr>
<td>Goal Position</td>
<td>$(300, -120, 10)^T \text{ m}$</td>
</tr>
<tr>
<td>Vehicle Velocity</td>
<td>1.0 m/s</td>
</tr>
<tr>
<td>Velocity Noise $\sigma_V$</td>
<td>0.2 m/s</td>
</tr>
<tr>
<td>Angular Velocity Noise $(\sigma_p, \sigma_q, \sigma_r)$</td>
<td>$(1.0, 1.0, 1.0) \text{ deg/s}$</td>
</tr>
<tr>
<td>Image Measurement Noise $(\sigma_{I_x}, \sigma_{I_y})$</td>
<td>$(1, 1) \text{ pixel}$</td>
</tr>
<tr>
<td>Simulation Time Step $dt$</td>
<td>0.1 s</td>
</tr>
<tr>
<td>Simulation Duration</td>
<td>300 s</td>
</tr>
<tr>
<td>Number of Landmarks</td>
<td>20</td>
</tr>
<tr>
<td>Number or Monte Carlo Runs</td>
<td>50</td>
</tr>
</tbody>
</table>

Path 1 - No Goal Loss

The path overview for the 3D path without losing the goal in the image is shown in Figure 4.7(a). The 3D vehicle motion is achieved by applying angular rates $p$ and $q$ which cause the vehicle to roll and pitch. The average error of $\rho_g$ is shown in Figure 4.7(b) along with the 95% confidence bounds. This shows that on average, the filter is performing well because the error approaches zero and the bounds are reasonable. Figure 4.7(c) shows the same average error but for the depth $d_g$. This also shows that the estimate is tracking. The actual roll and pitch of the vehicle are shown in Figure 4.7(d). Figures 4.8(a) and 4.8(b) show the average errors for $\alpha$ and $\beta$ respectively. The errors for both $\alpha$ and $\beta$ are very low. This is mostly due to the fact that the goal
Figure 4.7: Path overview of the 3D path without losing the goal. The 3D path with landmarks and goal is shown in (a). The goal location is shown in red. The other landmarks are shown in black. This shows the differing elevation of the landmarks. The landmark positions do not change for each simulation run. The average estimation error for $\rho_g$ is shown in (b). The 95% confidence bounds are shown in red. The average error in depth is shown in (c) with the 95% confidence bounds shown in red. The roll and pitch of the vehicle are shown in (d).

The vehicle does not leave the field of view during the entire run so these angles are measured at each time step. For this path, the vehicle is able to see the goal in the camera image the entire time. The path of the goal point in the image is shown in Figure 4.9. Figure 4.10 shows the value of $\tilde{\epsilon}_k$ for the 50 Monte Carlo runs. The 95% confidence intervals of $[2.36, 3.72]$ are also plotted. As mentioned above for the 2D case, the measure of consistency is not merely determined by the value of $\tilde{\epsilon}_k$ leaving the bounds at any time but by how much the value of $\tilde{\epsilon}_k$ is above the bounds. If $\tilde{\epsilon}_k$ rises significantly outside the bound then the filter is considered to be optimistic, meaning the covariance computed
Figure 4.8: Results for the 3D path without losing the goal. The average error for $\alpha$ is shown in (a) along with the 95% bounds in red. The average error for $\beta$ is shown in (b) with the 95% bounds in red.

by the filter is smaller than it should be. If the value of $\bar{e}_k$ is lower than the lower bound then the filter is conservative and the covariance is much larger than it should be. Another observation is the trend of the $\bar{e}_k$ value. The trend shows that the filter is remaining consistent by staying within the bounds.

Figure 4.9: Path of the goal point in the image for a 3D path. As can be seen, the goal is visible within the image the entire run. The image boundary is shown by the bold black rectangle.
Figure 4.10: Average NEES value of 50 Monte Carlo simulations at each time step. This is for the 3D path that allows the vehicle to see the goal in the camera image the whole run as shown in Figure 4.9.

Path 2 - Goal Loss

The second path tested for consistency is shown in Figure 4.11(a). This path is similar to the path shown in Figure 4.7 but the input angular rate $r$ is designed such that the goal point leaves the camera image. The average error of $\rho_g$ is shown in Figure 4.11(b) along with the 95% confidence bounds in red. As for the case when the goal did not leave the image, on average, the filter is performing well because the error approaches zero and the bounds are reasonable. Figure 4.11(c) shows the same average error but for the depth $d_g$. The roll and pitch of the vehicle are shown in Figure 4.11(d). This also shows that the estimate is tracking. Figures 4.12(a) and 4.12(b) show the average errors for $\alpha$ and $\beta$ respectively. In these plots it can be seen when the image leaves the field of view because the uncertainty in the angular states immediately starts to increase. The goal leaves the field of view the first time at $t = 54$ s and enters the field of view again at $t = 75$ s. The goal leaves the second time at $t = 100$ s and enters again at $t = 134$ s. This corresponds directly to the times when the uncertainty in $\alpha$ and $\beta$ start increasing and then decrease rapidly when the goal is back in the field of view. The errors for both $\alpha$ and $\beta$ are very low. This is mostly due to the fact that the goal does not leave the field of view during the entire run so these angles are measured at each time step. The path of the goal in the camera image is shown in Figure 4.13. The path of the goal in the image is shown in blue when the goal is seen in the camera image and it is shown
Figure 4.11: Path overview of the 3D path with losing the goal. The 3D path with landmarks and goal is shown in (a). The goal location is shown in red. The other landmarks are shown in black. This shows the differing elevation of the landmarks. The landmark positions do not change for each simulation run. The average estimation error for $\rho_g$ is shown in (b). The 95% confidence bounds are shown in red. The average error in depth is shown in (c) with the 95% confidence bounds shown in red. The roll and pitch of the vehicle are shown in (d).

in red when the goal is outside the camera field of view. This shows that the goal leaves the field of view two separate times but re-enters each time. The result of the consistency test is shown in Figure 4.14. This shows the value of $\bar{\varepsilon}_k$ for each time step. The result shows the value of $\bar{\varepsilon}_k$ starting high but then shifting to a lower value near the lower consistency boundary. Part of the reason for this is that with the 3D motion (non-zero $p$ and $q$), a higher process noise is needed because of the extra uncertainty due to this motion. Once the landmark comes into view the second time (at $t = 134$ s) the value of $\bar{\varepsilon}_k$ jumps down because of the decrease in the uncertainty but the process
noise remains higher than needed. The trend of $\tilde{\epsilon}_k$ is adequate to show that the filter is remaining consistent.

Figure 4.12: Results for the 3D path with losing the goal. The average error for $\alpha$ is shown in (a) along with the 95% bounds in red. The average error for $\beta$ is shown in (b) with the 95% bounds in red.

Figure 4.13: Path of the goal point in the image for a 3D path. The path of the goal is shown in blue when the goal is inside the camera field of view and is shown in red when outside the camera field of view. The goal leaves the image two separate times during each run. The image boundary is shown by the bold black rectangle.
4.4 Algorithm Performance

The previous section focused on the consistency of the EKF used in this work, however, the consistency alone does not highlight all of the performance improvements of this method over previous methods. The method outlined in this work will be compared with the work from [50]. For clarity in distinguishing between the two different methods, the method from this work will be referred to as the goal-based mapping algorithm and the work from [50] will be referenced as the robocentric mapping method. The main difference between the two methods is the landmark state definition as was explained in the derivation of the goal-based mapping method in Section 3.3. The comparison is only performed for the 2D robot motion case as the robocentric mapping algorithm is not designed for 3D vehicle motion. The 3D results of the goal-based mapping algorithm are shown and explained separately.

4.4.1 2D Algorithm Comparison

The performance of the two algorithms is evaluated by using a Monte Carlo simulation approach similar to that used in the consistency analysis. The same two paths that were used in Section 4.3 are also used for this analysis. The parameters used in the simulation are the same as in Table 4.1. Each method was run side by side to ensure that the timing, inputs and measurements.

Figure 4.14: Average NEES value of 50 Monte Carlo simulations at each time step. This is for the 3D path where the goal leaves the camera image during each run as shown in Figure 4.13.
were all the same for both methods. The average error of each of the goal states \([\alpha \ \beta \ \rho_g]^T\) is taken at each time step and the results are compared to each other. The average position error of the goal in the vehicle frame is also computed at each time step. This is done using the conversion from goal states to position in the body frame shown in Equation 3.8.

**Path 1 - No Goal Loss**

The results of the Monte Carlo simulation runs are shown in Figures 4.15 and 4.16. In each of the figures, the goal-based mapping method results are shown in black and the robocentric mapping results are shown in blue. The vehicle can see the goal during the entire run, which allows the estimate of \(\alpha\) and \(\beta\) to be very accurate as can be seen in Figures 4.15(a) and 4.15(b). The difference in the error between the two methods for \(\alpha\) and \(\beta\) is very small. The bigger difference between the two methods is the rate of convergence of the state \(\rho_g\). The error in \(\rho_g\) for each method is shown in Figure 4.15(c) which corresponds to the error in depth which is shown in Figure 4.15(d). It is easier to understand the difference in error looking at the plot of depth. This is in meters and the goal-based method performs much better than the robocentric method. The error in the goal position is shown in Figures 4.16(a) and 4.16(b). As can be seen in the zoomed in view of Figure 4.16(b), the convergence is much slower for the robocentric method.

**Path 2 - Goal Loss**

The results of the Monte Carlo simulation runs for path 2 are shown in Figures 4.17 and 4.18. The vehicle loses the goal two different times during the run. The times at which the goal is lost can be seen in Figure 4.17(a) where the estimate of \(\alpha\) diverges. This happens at \(t = 20\) s and the goal is reacquired in the image around \(t = 50\) s until \(t = 100\) s where the goal is lost again. Notice that the estimate of \(\alpha\) from the goal-based method does not diverge as rapidly as that from the robocentric method. This is due to the coupling of the landmark angles \(\theta_i\) to the goal angle \(\alpha\). Even when the goal is not seen in the image with the goal-based method, there is still some information gained about the position of the goal through the tracking of the other landmarks. This is also similar for the angle \(\beta\) as can be seen in Figure 4.17(b). During this experiment, the goal-based method again converges quicker in the state \(\rho_g\) as seen in Figure 4.17(c). There is also
Figure 4.15: Goal state error between the goal-based mapping and the robocentric mapping methods. These errors are for Path 1 which is shown in Figure 4.1. The vehicle was able to track the goal the entire run. The performance of both methods is similar for the angle states $\alpha$ and $\beta$ shown in (a) and (b), however the difference in the state $\rho_g$ is significant as can be seen in the depth $d_g$ which is shown in (d).

benefit in the state $\rho_g$ when the goal is outside the camera field of view. When the goal is first lost the error in $\rho_g$ of the robocentric method starts to increase while the error in $\rho_g$ of the goal-based method actually holds level. This is also seen in the depth in Figure 4.17(d) and in the goal position error plots of Figures 4.18(a) and 4.18(b) at $t = 20$ s and $t = 100$ s. When the goal is lost the position error starts to diverge for the robocentric method while the goal-based method is able to improve the estimate and the error slightly decreases.
Figure 4.16: The position error of the goal is shown in (a) with a zoomed in view shown in (b). Eventually the two positions converge but the robocentric mapping method converges much slower than the goal-based approach.

4.4.2 3D Algorithm Performance

The 3D simulation results also use the Monte Carlo simulation method that was used for the 2D tests. The parameters used for the 3D simulation are the same as those in Table 4.2. The path now includes roll and pitch angles as the vehicle drives.

No Goal Loss

The overview of the 3D path with no goal loss is shown in Figure 4.7. The results are shown in Figure 4.19. The angles $\alpha$ and $\beta$ are shown in Figures 4.19(a) and 4.19(b). These show little error which is expected because the goal is tracked during the entire run. Figure 4.19(c) shows the error in the inverse-depth $\rho_g$. This shows quick convergence and good tracking the entire run. The error in the depth $d_g$ is shown in 4.19(d). The position error is shown in Figure 4.19(e) with a zoomed-in view in Figure 4.19(f). The results show similar performance to the 2D results which is needed for this method to be implemented on an actual vehicle that will see 3D motion.

Goal Loss

The overview of the 3D path with goal loss is shown in Figure 4.20. These results are promising because this type of case will be encountered by an actual vehicle in the presence of
Figure 4.17: Goal state error between the goal-based mapping and the robocentric mapping methods. These errors are for Path 2 which is shown in Figure 4.4. The vehicle loses the goal at about 20 seconds and 100 seconds. Each time the goal is reacquired sometime later. The angles $\alpha$ and $\beta$ from the robocentric method are seen to diverge when the goal is not seen in the camera image in (a) and (b). There is similar behavior for the error in $\rho$ in (c) and $d$ in (d).

Obstacles. The states $\alpha$ and $\beta$ are shown in Figures 4.20(a) and 4.20(b). Each of the angles start to diverge when the goal leaves the camera field of view, however, the rate of divergence is very slow compared to the rate of divergence of the 2D robocentric mapping method shown in Figures 4.17(a) and 4.17(b). The performance in the state $\rho_g$ is shown in Figure 4.20(c) and converges to the true value. The performance in depth is shown in Figure 4.20(d). The real improvement in this method is shown in Figures 4.20(e) and 4.20(f) where it can be seen that the position estimation of the goal does not diverge when the goal leaves the field of view. This is important to the overall success of the system since this will allow the robot to reacquire the goal in the image once it is back in view.
Figure 4.18: Position error between the goal-based Mapping and the robocentric mapping algorithms. Similar to the results from path 1, eventually the two positions converge but the robocentric mapping method converges much slower than the goal-based approach.

4.5 Summary

This chapter has analyzed the performance of the the goal-based robocentric mapping algorithm. The designed filter was analyzed for observability and consistency. The observability of the proposed system was evaluated and the conditions were shown for when the states are locally weakly observable. Consistency of the filter was tested using a Monte Carlo approach and monitoring the normalized estimation error squared of the filter then comparing this to the expected value using the chi-squared distribution. The filter consistency was tested in 2D and 3D with paths in each that kept the goal in the camera field of view the entire run and also paths where the goal left the field of view.

A comparison of the goal-based mapping method and the robocentric mapping method was performed for the paths used in the consistency tests. The performance of the goal-based method was shown to improve the estimate of the goal states when compared to the robocentric method. It is noted here that the output of the robocentric mapping method is used to create a global map using additional algorithm steps. The requirement for this work is an improved position of the goal and for that reason the goal states were compared.
Figure 4.19: Errors for the goal states for the 3D path shown in Figure 4.7(a). The errors in $\alpha$ and $\beta$ are shown in (a) and (b) respectively. The error in inverse-depth is shown in (c) and the error in depth is shown in (d). Plot (e) and (f) show the position error of the goal.
Figure 4.20: Errors for the goal states for the 3D path shown in Figure 4.12(a). The errors in $\alpha$ and $\beta$ are shown in (a) and (b) respectively. The error in inverse-depth is shown in (c) and the error in depth is shown in (d). Plot (e) and (f) show the position error of the goal.
CHAPTER 5. VISUAL TRACKING

5.1 Problem Description

As described in Chapter 3, the objective of the robot is to automatically drive to a selected goal location. The goal is selected by clicking on an object in the image of an on-board camera. The goal location in the robot frame is then estimated by using an EKF to combine information from the motion of the vehicle and the measurement of the location of the goal in the image. Other landmarks in the image are also used to improve the estimate of the position of the goal. This chapter discusses the image processing required to track the goal and landmarks in the image frame while the vehicle drives towards the goal. The image processing is broken up into three main parts:

1. Feature selection
2. Feature tracking
3. Feature reacquisition

Each of the above steps are used for the goal and each landmark, however, the details of the image processing steps for tracking the goal are different from those of tracking the landmarks. Both procedures are explained below.

5.2 Goal Tracking

Tracking the goal while the robot drives to the goal creates a unique image processing problem. When the goal object is first selected, the size of the goal object in the image can be very small compared to its size in the image when the robot arrives at the goal. This means that the goal tracking algorithm needs to be robust to large changes in scale of the goal object. There is a large body of research dealing with this problem of tracking objects within an image. There are
many different papers that discuss how to track different features from image to image. Chapter 1 discussed much of the previous work in this area that is applicable to this work. Some of the more recent methods for feature detection and feature descriptions are shown in the list below:

- Scale-invariant feature transform [58].
- Speeded up robust features (SURF) [59].
- Binary robust independent elementary features (BRIEF) [60].
- Oriented FAST and Rotated BRIEF (ORB) [61].

Each of the feature description methods above can then be used with a matching routine between images to track the features. The problem with using these methods alone for this work is that the goal size changes significantly as the robot drives towards the goal. These methods track features very well from image to image but over time as the robot gets closer to the goal object the features have changed. For the purpose of tracking the goal, it is important that the change in size of the object does not cause the tracker to drift or lose the object. For this reason, a tracking algorithm involving a template matcher is used.

5.2.1 Goal Selection

The goal is selected by a user clicking on the image with a mouse pointer, this point is sent to the image tracker as the goal point. For the purposes of template matching, a template is created around the selected goal point and is stored for matching in future images. A template in this case is a smaller sub-image around a specific point in the main image. This is shown below in Figure 5.1. Let the position of the goal in frame $F^{im}$ be

$$\mathbf{p}_g^{im} = \begin{bmatrix} x_g \\ y_g \end{bmatrix}.$$ 

The user selects the point $\mathbf{p}_g^{im}$ shown by the red circle in the image. The template (shown by the red rectangle) is then created using the width and height as $dx_T$ and $dy_T$ respectively. In OpenCV the template rectangle is defined by the top-left corner of the rectangle and its width and height.
The top-left corner is shown by the black dot and the coordinates are \((x_T, y_T)\). These coordinates are defined as

\[
p_T^{im} = \begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} x_g - \frac{dx_T}{2} \\ y_g - \frac{dy_T}{2} \end{bmatrix}.
\]

Figure 5.1: Example camera image with the goal point selected. The image is shown in the green outline with the image frame, \(F^{im}\) shown at the top left corner of the image. The user selects the goal point \((x_g, y_g)\) and the template of size \((dx_T, dy_T)\) is created at the center of the selected point. The template is shown in red.

### 5.2.2 Goal Tracking

The objective of goal tracking is to find the goal object in each image as the robot drives to the goal. Once the template around the desired goal object has been created and saved as described above, the template is then used to find the goal object in each new image as it is received from the camera. There are two main steps in tracking the goal. First, there is a prediction step followed by the template matching step.
Prediction

The prediction step in goal tracking is used to calculate the expected position of the goal in the image frame. This is done by using the current states from the EKF. As described in Chapter 3, there are three states that define the estimated position of the goal in frame \( F^b \). These states are:

\[
x_g = \begin{bmatrix} \alpha \\ \beta \\ \rho_g \end{bmatrix}.
\]

The expected position of the goal in the image is calculated using the measurement equations derived in Chapter 3. These are shown here for completeness. The current states from Equation 5.1 are substituted into the following equation to compute the estimated pixel coordinates of the goal in the current image:

\[
\begin{bmatrix}
\hat{x}_g \\
\hat{y}_g
\end{bmatrix} =
\begin{bmatrix}
\frac{f_x}{p_g}\left(\frac{\cos(\beta) \sin(\alpha - \psi)}{p_g} - x_c \sin(\psi_c) + y_c \cos(\psi_c)\right) + c_x \\
\frac{f_y}{p_g}\left(\frac{\cos(\beta) \cos(\alpha - \psi)}{p_g} - x_c \cos(\psi_c) - y_c \sin(\psi_c)\right) + c_y
\end{bmatrix}.
\]

At each time step the expected position of the goal in the target frame is computed. Figure 5.2 shows an example of the change in position of the goal from one image frame to another. Let the true position of the goal in the image frame at time \( t_0 \) be defined as \( (x_{g,0}, y_{g,0}) \). This is the position selected by the user. After motion of the vehicle through one time step from \( t_0 \) to \( t_1 \) the expected position of the goal in the image frame is calculated from Equation 5.2, and due to noisy inputs into the EKF, the estimated position of the goal in the image \( (\hat{x}_{g,1}, \hat{y}_{g,1}) \) differs from the true position of the goal in the image \( (x_{g,1}, y_{g,1}) \). In Figure 5.2 the estimated path and the estimated position of the goal are shown in blue while the true path and the true position of the goal in the image are shown in red. A search region is created around the estimated position of the goal. This search region has a width and height of \( dx_s \) and \( dy_s \) respectively. Let the top-left corner of the search region be defined as:

\[
p^m_{im} = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} \hat{x}_g - \frac{dx_s}{2} \\ \hat{y}_g - \frac{dy_s}{2} \end{bmatrix}.
\]
The search region needs to be larger than the template because the template matching is performed by sliding the template around the search region and looking for the best match.

Figure 5.2: The entire image is shown in the green outline with the image frame $F^{im}$ shown at the top left corner of the image. The goal point position at time $t_0$ is shown. This is the position of the goal point when selected by the user. The estimated position of the goal point at time $t_1$ and the path of the goal point through the image from time $t_0$ to time $t_1$ is shown in blue. A search region of size $(dx_s, dy_s)$ is created at the center of the expected goal location in the image. The search region rectangle is shown in blue.

**Goal Template Matching**

To help describe how the template matching works, a new frame is defined. Let $F^s$ define the search-region frame. This frame is used for the template matching algorithm. This frame is similar to frame $F^{im}$ with the same alignment, however, the origin of frame $F^s$ is at the top-left corner of the search region. An example is shown in Figure 5.3. Let the position of the goal in frame $F^s$ be $p_{g}^s$. The position of the template in frame $F^s$ is $p_{T}^s$. After the search region is defined as shown in Figure 5.2, the template matching is performed. The template matching algorithm is described in [45] and takes the template image (in this case the smaller image defined by $p_{T}^{im}$ and the width $dx_T$ and height $dy_T$) and slides it across the larger search region image. The output of the template matching algorithm is the coordinate of top-left of the template image in frame $F^s$. 

89
which is $p_T^s$. Figure 5.3 shows an example of how the template match algorithm is used with the search region and the template. The red dot is the true position of the goal in frame $F_s$ and the blue dot is the estimated goal position in frame $F_s$. The output of the template matching algorithm is shown by the black dot which is the new position of the template $p_T^s$. The true position of the goal in frame $F_s$ is then calculated as

$$p_g^s = p_T^s + \begin{bmatrix} \frac{d_{xT}}{2} \\ \frac{d_{yT}}{2} \end{bmatrix}. $$

The goal location in the image is then calculated as

$$p_{g}^{im} = R_{im}^{si}p_{g}^s + p_{im}^{si}, \quad (5.3)$$

where $R_{im}^{si}$ is the transform from $F_s$ to $F^{im}$ which is simply

$$R_{im}^{si} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. $$

Figure 5.3: The template match takes the smaller template image (shown in red) and slides it across the larger search region (shown in blue). The output of the template match is the top-left position of the template image at the best match location in frame $F^{sr}$. This location is $p_T^{sr,i}$. 

90
Template Drift

The goal template must be updated frequently as the robot drives towards the goal because the object will grow larger in size within the image. If the template is not updated it will not accurately represent the current view of the goal object because of the change in viewing angle or the change in size. Figure 5.4 shows an example of how the goal object changes size as the robot approaches the object. The goal object is shown in the red ellipse in each image. Figure 5.4(a) is the starting image when the goal is selected by the user. At this point the robot is 60 m away from the goal. Figure 5.4(b) is taken when the robot is 25 m from the goal. The goal object is significantly larger at this point than when it was originally selected. Figure 5.4(c) is the next image and was taken with the robot being 15 m away from the goal. The last image is shown Figure 5.4(d) and is when the robot is 7 m away from the goal and the size of the goal is much larger than the original selection by the user.

Each time the template image is updated using the template matching algorithm there is a small amount of error. This error will create drift in the template position and it will start to leave the actual goal object in the image. To reduce the drift of the goal tracking in the image two additional steps are used. The first step is to use a hue-saturation-value (HSV) filter on the small template image to eliminate frame-to-frame drift from image noise. The second step is to monitor the size of the goal object in the template image and increase the size of the template when needed.

The HSV filter is used to check the pixels in the image to see if they fall in a configurable range of HSV limits. This is a very basic filter but eliminates the drift of the changing template. The HSV filter is only used on the small template image. This is important because the object could have similar HSV values that are similar to other objects in the seen. For example, in Figure 5.4 the HSV values of the barrel (which is selected as the goal object) are very similar to that of the blue sky in the background. Simply using an HSV threshold on the entire image would produce very large contours of objects and it would be difficult to pick out the goal object using that method alone. The drift is eliminated by adding an offset onto the template match pixel coordinates from Equation 5.3. The centroid of the largest contour from the HSV filtered image is used as the desired goal point centroid. Figure 5.5 shows an example of the offset that can be calculated from the center of the contour to the center of the template. Let the \( F_{TM} \) be the template frame with the origin at the top-left corner of the template image as shown in Figure 5.5. This frame is used to
Figure 5.4: Example image sequence showing how the goal object grows significantly in the image as the robot drives to the goal. Image (a) shows the initial image when the goal is selected. The robot is 60 m away. The image shown in (b) was taken when the robot was 25 m from the goal. Image (c) shows the goal when the robot is 15 m away and image (d) is when the robot is 7 m away. The goal is marked with a red ellipse in each image.

calculate the center of the HSV filter. The position of the HSV centroid in $\mathcal{F}_m$ is defined as $p_{\text{HSV}}^m$. Figure 5.5 shows the center of the current template with the red dot and the calculated position of the HSV centroid by the black dot. The required offset to remove drift is the difference between the HSV centroid and the template center. The HSV offset is defined as $p_o^m$ and is calculated as

$$p_o^m = p_{\text{HSV}}^m - \begin{bmatrix} \frac{dx_T}{2} \\ \frac{dy_T}{2} \end{bmatrix},$$ 

92
therefore, the final measurement output of the goal tracking algorithm is

\[ \mathbf{p}_{g}^{im} = R_{s}^{im} \mathbf{p}_{s}^{x} + \mathbf{p}_{s}^{im} + R_{Tm}^{im} \mathbf{p}_{o}^{Tm} , \]

where \( R_{Tm}^{im} \) is the rotation from \( \mathcal{F}_{Tm} \) to \( \mathcal{F}_{im} \) and is defined as

\[ R_{Tm}^{im} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \]

Figure 5.5: Example of how the HSV offset value is calculated. This offset will re-center the template on the object.

After the HSV filter step the template size is checked against the largest contour from the HSV filter. As can be seen in Figure 5.4, if the template started only big enough to fit the goal object when first selected (top-left image) then as the vehicle approaches the goal the template would not be big enough and then the measurement could become more noisy and move around within the goal object. It is desired to have the size of the template be big enough to see the entire object but small enough that the template matching algorithm can run efficiently. The template size is therefore started small and is increased by a percentage when the area of the largest contour of the HSV filtered image is larger than 50% of the image area. Figure 5.6 shows the template images that correspond to the full-size images from Figure 5.4. As the goal object becomes larger,
the templates become larger also. Figure 5.7 shows the corresponding HSV images from the templates shown in Figure 5.6.

Figure 5.6: The template from each image in the sequence shown in Figure 5.4. This is a zoomed in view of the templates. Each template increases in size as the goal object increases in size in the image.

Figure 5.7: The HSV images corresponding to the template images shown in Figure 5.6.

The final step in the goal tracking is to update the template image. This is done by using the final goal position in frame $\mathcal{R}^{im}$ along with the current width and height of the template ($dx_T$ and $dy_T$ respectively) to save the template image from the current image. Figures 5.8 and 5.9 show the process of tracking the goal while driving towards it. The images during four different robot
Figure 5.8: First overview of different images during the same test run with the robot driving toward a goal. The second two images are shown in Figure 5.9. Each sequence shows the original image with the outline of the search region (in blue), search region image with outline of the template image (in red), final template image and the HSV filtered image. This demonstrates the process of tracking the goal object while the object changes size during approach.

locations are shown. The first column of images is the actual image from the camera. Each camera image shows the outline (in blue) of the search region. The search region for each image is then shown in the second column. The outline of the selected template within each search region image is shown by the red outline. Then the corresponding template and HSV filtered template is shown.
5.2.3 Goal Reacquisition

During the tracking process of the goal there will be times when the goal leaves the camera field of view. This can happen when the robot has to drive around an object or if the robot is driving on rough terrain and pitches up or down to where the object is above or below the camera field of view. Once the goal leaves the camera field of view, the tracker tries to reacquire the goal as soon as possible. There are two main steps in the reacquisition process:
1. Calculate when the goal has left the field of view.

2. Locate the goal when it returns inside the field of view.

**Leaving the Field of View**

When the goal leaves the field of view it is important to have an updated template of the last view of the object before leaving the field of view. If the updated template of the goal is only half of the object then it will be more difficult to find the object once it re-enters the camera image. For this reason there is a border offset around the outside of the image. This border is shown in Figure 5.10. The reduced image size is shown by the black rectangle inside the image (green rectangle). The size of the offset is also shown in the figure. The offset along the $x$-axis is $\frac{dx_T}{2}$ and the offset along the $y$-axis is $\frac{dy_T}{2}$ which is half of the template size in the $x$ and $y$ directions. If the goal location in the image $p_{im}^g$ leaves the black rectangle then the goal will not be tracked in the image. This ensures that the last saved template of the goal will include the entire goal object in the template. This is needed for improved reacquisition when the goal enters the image at a future time.

**Locating the Goal**

Once the goal leaves the field of view the template for the goal is not updated. The estimated pixel coordinates of the goal in the image are continuously calculated using Equation 5.2. At each time step the image coordinates are checked to see if the goal is expected to be in the image. Once the location of $\hat{p}_{im}^g$ is within the valid tracking area (black rectangle in Figure 5.10) then the previously saved template image before the goal left the image is used in the template match against the current search region. This process is very similar to that shown in Figure 5.2, however, when reacquiring, the time differences could be significant meaning that it is not just a single time step from $t_0$ to $t_1$ but could be any amount of time. Because the time difference could be large, it is possible that the template of the goal is different from the current view of the goal. For this reason the score of the template match is checked against a maximum acceptable value. If the score is less than the acceptable value then the match is a success and the position of the goal in the image
Figure 5.10: The image tracking is decreased by the offset shown by the black rectangle. This area is used to ensure that the template will be updated properly as the goal leaves the image. The template image is shown in red.

is updated and the template is updated. If the score is too large then the template is not updated but is continued to be used against the search region from current images.

Figures 5.11 and 5.12 show an example of reacquiring the goal in the image. The first images in Figure 5.11 are taken just before the robot turns away from the goal to avoid the barrier that is seen in the image. As can be seen in the image the last template is saved right near the side of the image. This image is taken when the robot is 25 meters away from the goal. The robot then travels along the barrier until it can turn towards the goal again. The robot travels 15 meters along the barrier while not being able to see the goal object. After reaching the end of the barrier it is able to turn to the goal and try to reacquire the goal in the image. Figure 5.12 shows the images when the goal object is expected to be inside the image. The expected location of the goal is shown in the large image at the center of the yellow dot. The expected position does have error from the actual goal position. For this reason the search region is used around the expected position. This is shown
with the blue outline in the image. As shown in Figure 5.12, the template saved from the last view of the goal shown in Figure 5.11 is used in the template matching algorithm. The template matcher is able to reacquire the goal. In this example the robot traveled 15 meters while not seeing the goal. This took 30 seconds and the score of the template matcher (which is the output of the OpenCV function call) was 0.143. This score was lower than the maximum acceptable score value of 0.3. This value was set experimentally from example images.

Figure 5.11: Images show the last template and search regions before the robot turns to avoid the barrier in the image. The search region is shown in blue in the main image while the template image is shown in red in the search region. This template image is stored for future use when the goal is expected to be in the image again as shown in Figure 5.12.

5.3 Landmark Tracking

The landmark tracking algorithm is responsible for tracking multiple landmarks that can be sent to the EKF mapping for inclusion in the state vector. The landmark selection and tracking components are used from a ROS package called ROVIO. ROVIO stands for Robust Visual Inertial
Figure 5.12: Images when the robot has turned back toward the goal object after losing the goal from the camera field of view. The main image shows that the search region is centered around the expected goal location in the image (shown with the yellow circle). The template that was saved from when the goal was last seen (as shown in Figure 5.11) is then used in the template matcher to find the best match of where the goal object is in the current image.

Odometry and is explained in [62]. The package is available online for use in ROS [63]. The document describes a visual odometry method, however, for this work only the tracking component of ROVIO is utilized. Modifications to the ROVIO tracking algorithm were made to be suitable for this work.

5.3.1 Landmark Selection

The landmark selection process looks at the entire image for features that stand out. This is done using the OpenCV function goodFeaturesToTrack [45]. This function uses the Shi Tomasi corner detector [64] to find the best corners in the image. The function returns the quality of the corners as well. The list of available corners are then passed into the ROVIO tracker where the landmarks are then tested using a multi-level patch feature tracker [62]. The candidate features
are then tested against other features currently in the state vector from the EKF. This is done by using the projection in Equation 5.2 to compare the candidate pixel coordinates against the current landmarks pixel coordinates. The candidate landmarks that are too close to the current tracked landmarks are not used. Only a certain number of landmarks are allowed to be added into the state vector within an image. There is also a total number of landmarks limit that is imposed by the EKF. Once this limit is reached no additional landmarks are added into the tracker or the EKF.

5.3.2 Landmark Tracking

Once the landmarks have been added into the tracking algorithm then with each new image the new position of each landmark is calculated. This is done using the ROVIO tracking algorithm. In using the ROVIO tracker for this application it was noticed that the tracker would lose track of some features as the objects in the image grew larger while approaching the goal. Then the tracker would select new landmarks and not try to recover the lost landmarks. To improve the tracking for this application, a method of finding the landmarks again is required. To do this, a similar method used for the goal tracking as described in Section 5.2 is used. This involves using a template matching algorithm for each tracked landmark so that when the ROVIO tracker loses a landmark or when a landmark leaves the camera field of view it can be reacquired.

Because each landmark can be reacquired using the template matching routine, each time a landmark is successfully tracked there is a template image created with the center of the template being the landmark position in the image. Let the position of landmark $i$ in frame $F^{im}$ be

$$ p_{im}^i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

Figure 5.13 shows three landmarks with three templates that have been saved with the center at the tracked landmark position. Each template has the same size as the template used for the goal. The width is $dx_T$ and the height is $dy_T$. Like the template created for the goal, the template corner for landmark $i$ is calculated as

$$ p_{im}^{T_i} = \begin{bmatrix} x_{T_i} \\ y_{T_i} \end{bmatrix} = \begin{bmatrix} x_i - \frac{dx_T}{2} \\ y_i - \frac{dy_T}{2} \end{bmatrix}.$$
Figure 5.13: Three different landmark templates are shown by the red squares.

Figure 5.14 shows an example from an actual data set where the landmarks are selected using the method described above. The figure shows the image with the landmark positions and the template outlines in red. It also shows the actual saved template images. These template images are updated at each time step so they are ready to be used in reacquisition if needed. If the landmark is lost or leaves the camera field of view the landmark reacquisition step is used.

5.3.3 Landmark Reacquisition

The landmark reacquisition step is similar to that used for the goal. When a landmark is to be reacquired, the current estimate of the landmark position in the image frame is calculated from the measurement equations derived in Chapter 3. Using the current states from the EKF the expected image location of landmark $i$ is
Figure 5.14: An example image with the landmark features being tracked. The landmark features in the image are shown with the red circles. Each landmark has an associated template image that is saved each time the landmark is tracked. These template images are shown around the main image for each landmark.

Once the expected position of the landmark is within the search area (black rectangle in Figure 5.10), then a search region centered at the expected landmark location is created similar to the region shown in Figure 5.2 for the goal object. The previously saved template is then used with the search region in the template matching algorithm. The output is the location of the corner of
the template in the search region that best matches. The score of the match is also returned. This score is then checked against the maximum allowable score to decide if this match can be accepted.

If the score is not acceptable then the template match is run again each time step with a new search region being used with the same template. If the score continues to be too high for a certain period of time then the landmark is removed from the landmark tracking algorithm. Once removed from the tracking algorithm the landmark is also removed from the EKF state vector.

If the score is acceptable then there is one additional step for goal reacquisition. The accepted landmark position from the template match is then sent to the ROVIO selection algorithm as a candidate landmark. This is done to update the ROVIO tracking algorithm with the same landmark that had previously been lost. This step is important because it allows the ROVIO tracking algorithm to track the same landmark without having to add it as a new landmark to track. It is possible that the template matching algorithm will successfully reacquire the landmark but the ROVIO selection algorithm will not accept the candidate landmark again because the corners within the new template and image patches might not be strong enough. If this happens then the landmark is removed from tracker and also from the EKF. If ROVIO accepts the landmark again then it continues to be tracked in the ROVIO portion of the tracker.

5.4 Summary

This chapter has outlined and explained the visual tracking algorithm designed for this work. The tracker sends landmark image positions to the robocentric mapping algorithm to allow the robot to successfully track a goal object and additional landmarks in the map. The method of tracking and reacquiring the goal was explained. Actual images were shown to explain how the tracker algorithm works. The method for tracking and reacquiring the landmarks was also explained. The method has been implemented and tested successfully on the hardware explained in Chapter 2. The results from the visual tracker will not be shown here but will be part of the results from the experiments explained in detail in Chapter 7.
CHAPTER 6. NONLINEAR MODEL PREDICTIVE CONTROL

6.1 Introduction

Model predictive control (MPC) uses a mathematical model to predict the behavior of the system, given the inputs. This predicted behavior is used to solve an on-line optimization problem at each time step. The optimization problem is solved to find a control sequence from the current time to some time in the future that minimizes the cost function during that time sequence. The optimization problem is solved at each time step and the first input in the sequence is the current control input. The problem is then solved repeatedly to drive the system to the desired states. MPC is also known as receding horizon control and is similar to infinite horizon optimal control methods. MPC differs because it is not computing an off-line feedback method but is calculating the current input by solving the finite horizon optimization problem at each time step.

6.2 Literature Review

MPC has been successfully employed for many years in the area of process control for systems with slow dynamics. The slow dynamic systems allowed for the computers to solve the optimization problem in real-time despite the complexity of the problem. With the increase in speed of modern computers, these MPC algorithms are being used for many different applications. There are many papers that discuss the general MPC problem [65–68]. In recent years there has been much work in the area of nonlinear MPC in the field of robotics. The areas of interest that align with this work are the areas of robot control, obstacle avoidance and sensor constraints. The literature from each of these areas is discussed here.
6.2.1 Robot Control

There are many different methods of MPC discussed in the literature for robot control. The method of control is mostly defined by the cost function. In [69–71] a dynamic vehicle model is derived to control a vehicle to the path. The cost function is designed to minimize heading and lateral errors with respect to the path while also ensuring that the vehicle model constraints are satisfied. The designed methods are shown to stabilize the vehicle along the desired path, however, obstacles are neglected. In [72] the path control of a robot arm is designed with the MPC formulation. A desired path is planned and the error from that path is used in the cost function to drive the arm along the correct path.

Other methods of path control include active-SLAM [39, 40], where MPC based control drives a robot that is mapping the environment with SLAM. A 2D LiDAR senses the environment and the goal is to drive a path that will properly estimate line features in the environment. In [73] the MPC plans a path for a robot containing a bearings-only sensor trying to minimize the uncertainty of an object by a prescribed terminal time. The cost function design is based on the information gain in the environment. This is similar to the goal in this work, however, obstacles and sensor constraints are not considered. In [74] an observability-based cost function is used to plan the path of an unmanned aerial vehicle through an environment that includes obstacles. The cost function includes the cost to get to a desired known location and steer the vehicle such that the uncertainty of obstacle locations is minimized. This control method also leads the vehicle on a path that will avoid the obstacles. The control does not consider sensor constraints.

6.2.2 Obstacle Avoidance

MPC also has many applications for obstacle avoidance. In [75] a dynamic ground vehicle model is used in the MPC framework along with obstacle data from a LiDAR to avoid obstacles. The area around the vehicle is divided into sectors that are hard constraints within the MPC algorithm. These hard constraints cannot be violated, therefore the vehicle avoids the obstacles. In [76] the obstacles are also defined as hard constraints, but they are combined with road boundaries as locations where the vehicle cannot drive. Obstacles can also be defined in the cost functions as soft constraints. The work in [77] addresses a dynamic vehicle model for a ground robot with a
desired path. The error along the path is used in the cost function and the distance to the obstacles generates a high cost as the distance gets closer. This method of obstacles in the cost function is known as a barrier function \[78\], which generates a cost function that approaches infinity as the constraint is approached. This method is designed in \[79\] with a ground vehicle. In this work, the obstacles will be hard constraints in the MPC formulation.

### 6.2.3 Sensor Constraints

The problem of sensor constraints needs to be addressed in this work since the environment is unknown and the vehicle might need to turn such that the goal leaves the field of view. In \[80\] the sensor constraints of a camera are handled in the controller, however, the camera points down toward the ground looking for obstacles so that the vehicle will not collide with any obstacles. The camera field of view limits the available paths that the robot can take in the near term so that the obstacle information of the path will be known. The work in \[81\] is more inline with the needs for this work. The sensor constraints are for a camera field of view looking at the environment. The field-of-view constraint is implemented using a soft constraint, meaning the field of view is an input in the cost function, however the cost is a barrier function that will not allow the vehicle to turn outside the camera field of view.

### 6.3 General Problem Formulation

In the literature that discusses MPC there are many different notations. The notation in this work will be explained as well as an explanation of the general problem formulation. A discrete formulation is used here similar to that of \[65\] and \[82\]. The nonlinear discrete system model is given by

\[
x_{k+1} = f(x_k, u_k) \quad (6.1)
\]
\[
y_k = h(x_k) \quad (6.2)
\]

where \(x_k \in \mathbb{R}^n\) and \(u_k \in \mathbb{R}^m\). The value \(n\) is the number of states and \(m\) is the number of inputs into the system. The vector \(x_k\) is the state at the discrete time \(k\) and \(u_k\) is the control vector at time \(k\).
The function \( f(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \) is the system dynamics, which is for prediction to the next state, given the current state and the input. The goal of MPC is to select an optimal control sequence that minimizes a cost function over \( N \) look-ahead steps. Let the feasible control sequence at time \( k \) be

\[
\pi_k = \{ v_{k|k}, v_{k+1|k}, \ldots, v_{k+N-1|k} \}
\]

where the value \( v_{k+1|k} \) is the feasible input at one time step into the future computed from the current state at time \( k \). The state sequence corresponding to the feasible control input is

\[
\{ x_{k|k}, x_{k+1|k}, \ldots, x_{k+N|k} \},
\]

and the control input into the system is then selected as

\[
u_k = v_{k|k}\]

which means that only the first feasible input from \( \pi_k \) is actually used in the control. The control input \( u_k \) is the input that will take state \( x_k \) to state \( x_{k+1} \). After one time step the optimal control sequence is computed again and the first value in that sequence is again selected as the control input. This process is continually repeated.

For a given state \( x_k \) and a control sequence \( \pi_k \), a cost function \( V(\cdot) : \mathbb{R}^n \times \mathbb{R}^{mN} \rightarrow \mathbb{R} \) is defined as

\[
V(x_k, \pi_k) = \sum_{i=k}^{k+N-1} \ell (x_{i|k}, v_{i|k}). \tag{6.3}
\]

The function \( \ell (x, v) \) is known as the stage cost. This is the cost incurred at the predicted state \( x \) and input \( v \). Equation 6.3 calculates the total cost of moving the current state \( x_k \) to the future state \( x_{k+N|k} \) using the input sequence \( \pi_k \). Under the assumption that the functions \( f(\cdot) \) and \( \ell(\cdot) \) are time invariant the problem can be reset at each time step to be starting from \( t = 0 \). This is done by setting the initial state in the problem to be the current state. The general MPC problem is then
defined by

$$\min_{\pi} \quad V(x_0, \pi) = \sum_{i=0}^{N-1} \ell(x_i, v_i)$$

subject to

$$x_{i+1} = f(x_i, v_i),$$

$$g_{eq}(x_i, u_i) = 0,$$

$$g_{in}(x_i, u_i) \leq 0.$$  \hspace{1cm} (6.4)

Equation 6.4 contains three different constraint equations. The first constraints are the constraints on the states due to the dynamics, then there are general equality constraints \( g_{eq}(x_i, u_i) \) and general inequality constraints \( g_{in}(x_i, u_i) \). Equation 6.4 shows the general problem formulation.

In practice, there can be many different equality and inequality constraints. The formulation of the cost function and constraint functions for this work will be explained in the next sections.

The goal of this work, as was explained in previous chapters, is to drive the robot to a selected goal point in the camera image. There is no assumption that the goal will be visible during the entire path to the goal. For this reason, it is important that the robot take the path that will give the best estimate of the goal in the robot frame. The reason for this is so that if the robot loses the goal in the image frame it will have a good estimate of the goal position to reacquire the goal. The loss of the goal in the image could be due to various factors such as driving around an obstacle, an object blocking the view from the camera to the goal or the robot being pitched up or down such that the goal is above or below the camera field of view.

It is desired to use the MPC formulation shown in Equation 6.4 to drive the robot on the path that will give the best estimate of the position of the goal with respect to the robot. The robot will also have to maneuver around obstacles and continually track the object in the image frame of the camera. Once the goal object is selected in the image, the goals of the robot are:

1. Arrive at the goal location
2. Minimize the uncertainty of the goal position while driving towards it
3. Keep the goal in the camera field of view if possible
4. Avoid obstacles while driving
5. Reacquire the goal in the camera field of view frame if lost
The cost function and constraint functions are designed such that the goals listed above are achieved. First the cost function will be designed and explained, followed by the explanation of the constraint functions.

6.4 Cost Function

The generalized cost function for MPC was shown above as part of the general MPC problem formulation in Equation 6.4. The cost function

$$V(x_0, \pi) = \sum_{i=0}^{N-1} \ell(x_i, \nu_i)$$ (6.5)

is the sum of each of the stage costs $\ell(x_i, \nu_i)$ at each time step $i$. The design of the stage cost is important in the successful achievement of the listed goals. When there are no obstacles present, the stage cost will be the major factor in choosing where the vehicle drives. For this reason, the stage cost will need to achieve goals 1, 2, 3, and 5 as listed in Section 6.3.

6.4.1 Observability-based Stage Cost

An observability-based approach for path planning and control was explained in [74] and this work builds on that approach. This work also extends this method to consider sensor field-of-view constraints as will be explained below. The general approach to using observability in the stage cost will be explained here and then the formulation for this problem is derived. Observability is chosen as a method for the stage cost because of the following lemma which is shown in [74,83]. The lemma is also shown here for completeness.

Lemma 6.4.1 Suppose that there exist positive real scalars $\alpha_1$, $\alpha_2$, $\beta_1$, $\beta_2$ such that $\beta_1 I \leq \bar{O}^T \bar{O} \leq \beta_2 I$ and $\alpha_1 I \geq \bar{C}^T \bar{C} \geq \alpha_2 I$ then,

$$\left( \frac{1}{\beta_2 + \frac{1}{\alpha_2}} \right) I \leq P \leq \left( \frac{\alpha_1 + \frac{1}{\beta_1}}{1} \right) I,$$ (6.6)

where $I$ is the identity matrix, $C$ is the controllability matrix, $O$ is the observability matrix and $P$ is the covariance matrix.
Lemma 6.4.1 means that to minimize the upper and lower bounds on the covariance matrix \( P \), the two values \( \beta_1 \) and \( \beta_2 \) need to be maximized. These are the maximum and minimum singular values of the observability matrix \( O \). This is accomplished by maximizing the determinant of \( O^T O \) which will maximize the singular values.

The EKF in this work was described in Chapter 3. The observability of the filter was shown in Section 4.2, however, for the control of the robot using observability, only two states are considered. The state of the goal will be reduced to

\[
g_c = \begin{bmatrix} \alpha \\ \rho_g \end{bmatrix}, \tag{6.7} \]

where the subscript \( c \) implies the state vector is for control. The reason for the omission of the state \( \beta \) is because this state is the elevation angle to the goal and is not controllable, therefore it is omitted for computing the desired path of the robot.

The observability matrix is computed for the reduced state vector for the goal. Because the state \( \beta \) is omitted from the state vector, only the measurement equation defined as \( h_{g,x} \) is used for control purposes, using the assumptions as described in Section 4.2 which are

- Camera located at the origin of frame \( F^b \)
- \( F^{c2} \) is aligned with frame \( F^b \)
- The image is square with no offset, \( c_x = c_y = 0 \)
- The focal lengths are equal, \( f_x = f_y = f \)
- The pixel size parameters are \( p_x = p_y = 1 \)

Under these assumptions the measurement equation is

\[
h_{g,x} = -f \tan \alpha. \tag{6.8} \]
From this, the observability matrix is defined as

$$\mathcal{O} = \begin{bmatrix} \frac{\partial L_0 h_{g,x}}{\partial x} \\ \frac{\partial L_1 h_{g,x}}{\partial x} \end{bmatrix},$$

where the two rows are from Equations 4.3 and 4.2.1 with $\beta = 0$, this gives

$$\mathcal{O} = \begin{bmatrix} -f \sec^2 \alpha & 0 \\ -f \rho_g V \sec \alpha \left(1 + 2 \tan^2 \alpha\right) & -f V \sec \alpha \tan \alpha \end{bmatrix}. \quad (6.10)$$

The value that will be maximized is

$$\det \left( \mathcal{O}^T \mathcal{O} \right) = f^4 V^2 \sec^6 \alpha \tan^2 \alpha. \quad (6.11)$$

For constant values of $f$ and $V$ this equation shows that to minimize the upper and lower bound of the covariance matrix the desired $\alpha$ is $\frac{\pi}{2}$. When $\alpha = 0$, the value in Equation 6.11 is 0 and there is no information about the state $\rho_g$ gained as the vehicle drives directly towards the object. Since the value of $f$ is a constant value for a given camera it can be removed from the equation as it will not change the control solution in the optimization problem. The velocity of the robot is also constant for this work so the value of $V^2$ in Equation 6.11 will not affect the solution either, therefore it will also be removed from the cost function. The cost function becomes

$$\ell = -\sec^6 \alpha \tan^2 \alpha. \quad (6.12)$$

This is the function used in the optimization problem. Because it is a minimization problem the negative value is included in Equation 6.12.

Figure 6.1 shows the stage cost as a function of the angle $\alpha$. The cost goes very low (this is desired) very rapidly as $\alpha$ goes to $\frac{\pi}{2}$. The red lines show a field of view of 60 deg which is $\pm 30$ deg.

Using this cost function alone as the stage cost $\ell(x_i, v_j)$ for robot control will try and drive the state $\alpha$ to $\frac{\pi}{2}$, however, using a camera with a limited field of view on a fixed mount on the robot,
Figure 6.1: Plots of the cost function value vs. the angle $\alpha$. Figure (a) shows the stage cost $\ell$ versus changing values of $\alpha$. This shows how the cost function has the lowest cost at $\alpha = \pm 90 \text{ deg}$. Both figures show the camera field of view angle in red. Figure (b) shows only the cost within the field of view.

it is not possible to achieve an $\alpha$ of $\frac{\pi}{2}$ while still viewing the object. For this reason a method of handling the viewing constraints of the camera is needed in the cost function.

The goal is to minimize Equation 6.12 using the observability matrix. When the goal is outside the field of view of the camera there is no measurement of the goal so the states $\alpha$ and $\rho_g$ actually become unobservable and the observability based cost function is invalid and could be set to 0 because there is no measurement available. This also agrees with what would happen in the EKF. If the goal was being tracked and then leaves the field of view not to return, the covariance would grow unbounded through the prediction equation of the covariance. This is what Lemma 6.4.1 also shows since the values of $\beta_1$ and $\beta_2$ would be 0, the upper bound on the covariance would be infinite. Using Equation 6.12 as currently defined would drive the robot such that the goal would leave the camera field of view rapidly.

One way to make sure that the goal stays in the field of view is to apply a hard inequality constraint to the MPC problem. This constraint would be

$$|\alpha| \leq \alpha_{FOV} \forall i,$$
however, this will not work for all environment scenarios. For example, if there was an obstacle that required the robot to turn such that $\alpha$ had to leave the field of view then this hard constraint would be violated, therefore, a different method of keeping the goal in the field of view is required. The goal will be kept in the field of view by penalizing the cost function when getting close to the edge of the field of view. This is accomplished using a blending function that will zero out the return value of the observability-based function. This method acts like a soft constraint for the field of view. When the robot drives with no obstacles present then the goal will be in the field of view, but when there is a requirement to drive around an obstacle where the goal leaves the field of view then this allows the goal to leave the field of view without violating any hard constraints.

The stage cost function with blending is defined as

$$\ell = - \left( \sec^6 \alpha \tan^2 \alpha \right) \sigma(\alpha), \quad (6.13)$$

where

$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha - \alpha_0)} + e^{M(\alpha + \alpha_0)}}{(1 + e^{-M(\alpha - \alpha_0)})(1 + e^{M(\alpha + \alpha_0)})}. \quad (6.14)$$

The new stage cost is shown in Figure 6.2. When the angle $\alpha$ is leaves the field of view (shown by the vertical red lines) the cost tends toward zero. For this example $M = 40$ and $\alpha_0 = 30$ deg.

![Figure 6.2: Stage cost from Equation 6.13 with changing $\alpha$. When the angle $\alpha$ is leaves the field of view the cost goes to 0.](image-url)
There are two parameters in the blending function that define the transition cutoff point and the transition rate; these are $\alpha_0$ and $M$ respectively. An example of how these values change the output of the cost function is shown in Figure 6.3. In Figure 6.3a $\alpha_0 = 30$ deg, the green line shows $M = 40$ and the blue line shows $M = 200$. In Figure 6.3b $M = 40$ and the green line shows $\alpha_0 = 30$ deg while the blue line shows $\alpha_0 = 40$ deg.

![Graph](a) ![Graph](b)

Figure 6.3: Plots of the cost function value vs. the angle $\alpha$. Figure (a) shows two examples with different values of $M$ with $\alpha_0 = 30$ deg. $M = 40$ for the green line and $M = 200$ for the blue line. Figure (b) shows two different values of $\alpha_0$ with $M = 40$. $\alpha_0 = 30$ deg for the green line and $\alpha_0 = 40$ deg for the blue line.

Goal 5 from Section 6.3 still needs to be addressed by the cost function. If the goal leaves the field of view at any time, then the cost function needs to bring the goal back into the field of view. This is done with a penalizing cost when $\alpha$ is outside the field of view. The cost function shown in Figure 6.3 will not penalize for being outside the field of view because the cost is 0. This will not bring the goal back into the field of view. Instead, this is accomplished by adding a quadratic cost of $\alpha$ onto the cost function. The new stage cost becomes

$$\ell = -\left(\sec^6 \alpha \tan^2 \alpha\right) \sigma (\alpha) + \alpha^2.$$  \hspace{1cm} (6.15)

The new cost function is shown in Figure 6.4. One important note of this cost function is that the minimum value of Equation 6.15 is still within the field of view defined by the value of $\alpha_0$. 

115
Figure 6.4: Stage cost from Equation 6.15 with changing $\alpha$. The value of $\alpha_0 = 30$ deg and $M = -40$. When the angle $\alpha$ is outside the field of view the cost is increased.

Figure 6.5 shows the cost function $\ell(|\alpha|)$ plotted against changing values of $\alpha$. There are 10 different plots showing how the cost function changes with different values of $M$ in the blend function. The values of M range from $-200$ to $-40$. Figure 6.5(b) shows a zoomed in view with labels. The smaller the value of $M$ the slower the transition is from 1 to 0. The sensor field of view is shown in the figures as the vertical red lines. It is clearly seen in Figure 6.5(a) that the minimum of the cost function is within the bound of $\pm \alpha_{FOV}$. In this example $\alpha_{FOV} = 30$ deg.

Figure 6.5: Cost function shown for different values of $M$. 

116
Figure 6.6 shows the derivative of the cost function with respect to $\alpha$. This is to show that using the derivative, there are three possibilities for minimum or maximum values (where the derivative is equal to 0). As can be seen in Figure 6.6(a), all three zeros of the derivative lie within the range $[-\alpha_{FOV}, \alpha_{FOV}]$.

![Figure 6.6: Derivative of cost function shown for different values of $M$.](image)

Figures 6.5 and 6.6 show that the minimum value of the cost function is guaranteed to be in the range $[-\alpha_{FOV}, \alpha_{FOV}]$. This is true for values of $M$ within the range $[-200, -35]$.

### 6.5 Input Constraint

One of the constraints in the MPC problem will be the constraint on the input value in

$$x_{k+1} = f(x_k, u_k).$$

There are two reasons for this constraint: first is that there are physical limitations to the robot that will only let it turn at the maximum angular rate and second is to reduce the rate at which the goal and the landmarks are traveling through the image. If the robot is spinning fast then the visual tracking is much more difficult and the image can be blurred. This limit also ensures that even
while turning, the goal and landmarks are still tracked properly. This constraint is defined as

\[ |u_k| \leq u_{\text{max}}. \quad (6.16) \]

### 6.6 Obstacles

As the robot drives towards the goal it needs to check for obstacles that could cause a collision along the desired path. Obstacles are implemented in the MPC as hard constraints in the optimization problem. A 2D LiDAR detects obstacles. A typical 2D LiDAR scanner will have an angular resolution \( \leq 1.0 \) deg. With a large field of view (typically in the range of 180 deg to 360 deg) there are many returns from the LiDAR. It is desired to reduce the number of obstacles that are sent to the MPC optimization to keep the problem tractable. For this reason an obstacle detection routine is designed to keep the number of obstacle constraints, low yet still represent the obstacle to avoid collision. This method is explained below.

#### 6.6.1 Obstacle Detection

The 2D LiDAR has a circular scan that returns the range of objects seen due to reflections of the LiDAR. Figure 6.7 shows the LiDAR mounted on the robot. The LiDAR frame \( \mathcal{F}_l \) is also shown along with the body frame \( \mathcal{F}_b \). The LiDAR frame is aligned with the body frame but the origin of the LiDAR frame is centered at the origin of the LiDAR as shown in the figure. The obstacles are detected in \( \mathcal{F}_l \).

To reduce the number of obstacles sent as constraints to the MPC algorithm, the angular field of view is divided into sectors. Only one range is returned from each sector. The full angular field of view of the LiDAR is \( \beta_l \). The angular range of the LiDAR is divided up into \( N_s \) sectors as shown in Figure 6.8. The sector angle is

\[ \beta_s = \frac{\beta_l}{N_s}. \quad (6.17) \]

Within each sector there will be multiple scan returns from the LiDAR. Figure 6.9(a) shows a single sector with multiple LiDAR returns in the sector. The red circle represents an obstacle within the sector. The LiDAR resolution between scans is \( \theta_{\text{res}} \). The range returned by scan \( i \) is \( r_i \).
Figure 6.7: LiDAR on the vehicle. The LiDAR frame $F^l$ is shown in green while the body frame $F^b$ is shown in black.

The yellow circles mark the location of the return of the LiDAR measurements. There is no range measurement returned if no object is sensed. Let the vector $p_{sj}$ be the vector of LiDAR returns within the $j$-th sector. This is defined as

$$p_{sj} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{bmatrix}, \quad (6.18)$$

where $k$ is the number of scans within each sector. Only one range from each sector is returned. The range that is returned is the smallest range within the sector. The range from sector $j$ is

$$r_{sj} = \min (p_{sj}). \quad (6.19)$$

The angle associated with the range $r_{sj}$ is the angle of the middle of the sector and is defined as

$$\theta_j = (j-1)\theta_s + \frac{\theta_s}{2}. \quad (6.20)$$

Both the angle and range of sector $j$ is shown in Figure 6.9(b).
Figure 6.8: The LiDAR field of view is split up into $N_s$ number of sectors. Each sector has the same angular range $\beta_s$ as shown. The blue circle represents the center of the LiDAR.

The final step in the obstacle detection process is to convert the range and angles from each sector into $(x,y)$ positions in frame $\mathcal{F}^b$. Let the position of the LiDAR in frame $\mathcal{F}^b$ be

$$p^b_l = \begin{bmatrix} x_l \\ y_l \end{bmatrix}. \tag{6.21}$$

The position of the obstacle returned in sector $j$ in frame $\mathcal{F}^l$

$$p^l_{o,j} = \begin{bmatrix} r_{xj} \cos \theta_j \\ r_{xj} \sin \theta_j \end{bmatrix}. \tag{6.22}$$

The position of the obstacle in frame $\mathcal{F}^b$ is then calculated as

$$p^b_{o,j} = p^b_l + R^b_i p^l_{o,j}, \tag{6.23}$$

where the rotation matrix $R^b_i$ is the rotation from frame $\mathcal{F}^l$ to frame $\mathcal{F}^b$ and is the identity matrix.
Figure 6.9: A single sector is shown with an obstacle (red circle) inside the sector. There are multiple returns on the obstacle within the sector as shown in (a) but only the midpoint of the sector is returned with the smallest range \( r_{s,j} \) as shown in (b).

### 6.6.2 Obstacle Constraints

Once the obstacles have been detected then the positions of the obstacle points \( p_{o,j}^b \) need to be constraints in the MPC problem. This is done with an obstacle boundary around the robot with radius \( R_v \). This radius is the closest distance the robot can get to any obstacle. Figure 6.10 shows the robot with a single obstacle return. The location of the obstacle return point is \((x_o, y_o)\) in frame \( F^b \) and the position of the robot is at \((0, 0)\). The distance \( D \) to the obstacle from the center of the robot is therefore

\[
D = \sqrt{x_o^2 + y_o^2}.
\] (6.24)

It is noted that a \( D \) is used for the distance to an obstacle so it is not confused with \( d \) which is the distance from the vehicle to the goal position. The robot is considered safe from an obstacle if \( D \geq R_v \). This is the constraint that needs to be checked for all positions of the robot against all sensed obstacles. This is now generalized to multiple obstacles and multiple positions of the robot along the candidate path from the MPC algorithm. To do this, the positions of the vehicle must be calculated along each candidate path.
Figure 6.10: Robot shown with a single obstacle.

The MPC algorithm solves for the input vector $\pi_k$ that minimizes the cost function. The cost function uses the goal point states $x_g$ as defined in Equation 3.7. The states used to check the distance to the goal point, however, are different. Let the future position of the vehicle in frame $\mathcal{F}^b$ at time $t = k$ be $(x_{v,k|0}, y_{v,k|0})$ and let the future heading of the vehicle with respect to the current heading be $\psi_{v,k|0}$. Now using the same input vector $\pi_k$ as used for the propagation of the goal states, the future vehicle position and heading are propagated to check the future distances to currently sensed obstacles. To do this, the 2D kinematic equations for the vehicle motion are needed. It is assumed that the future positions are always calculated with respect to the current frame $\mathcal{F}^b$. Using the current velocity $V_k$ and the control input $v_{k|0}$, the equations to propagate the position and orientation of the vehicle are

$$
\begin{bmatrix}
    x_{v,k+1|0} \\
    y_{v,k+1|0} \\
    \psi_{v,k+1|0}
\end{bmatrix}
= 
\begin{bmatrix}
    x_{v,k|0} + (V_k|0 \cos \psi_{v,k|0}) \Delta t \\
    y_{v,k|0} + (V_k|0 \sin \psi_{v,k|0}) \Delta t \\
    v_{k|0} \Delta t
\end{bmatrix}.
$$

(6.25)
Figure 6.11 shows a candidate path (in blue) with two obstacles. The path is shown with the discrete positions of the vehicle after each time step (blue circles) and the distance from each vehicle position to each obstacle is shown with the red lines. As shown in the figure, the distance $D_{1,0}$ is the distance from the vehicle position at $k = 1$ to obstacle 0, and the distance $D_{2,1}$ is the distance from the vehicle position at $k = 2$ to obstacle 1. This is generalized as $D_{k,j}$, the distance from the vehicle position at $t = k$ to obstacle $j$ and

$$ D_{k,j} = \sqrt{(x_{v,k} - x_{obs,j})^2 + (y_{v,k} - y_{obs,j})^2}. $$

(6.26)

The general constraint equations for obstacles are defined as

$$ D_{k,j} > R_v. $$

(6.27)
6.6.3 MPC Problem Statement

Using the above derivations and discussion, the MPC problem is to solve

\[
\min_{\pi} \quad V(x_0, \pi) = \sum_{i=0}^{N-1} \ell(x_i, \nu_i)
\]

subject to

\[
x_{i+1} = f(x_i, \nu_i),
\]

\[
|\nu_j| \leq v_{\text{max}}
\]

\[
D_{i,j} > R_v.
\]

6.7 Analysis

The requirement for the MPC is to drive the robot to the goal location. The guarantee of arrival to the goal will be proven by using intermediate theorems that build on each other to show that using the defined MPC algorithm in Equations 6.28 to 6.31, the robot will drive to the goal.

In the discussion that follows, the values of \(\alpha_{FOV}\) and \(-\alpha_{FOV}\) are defined as the field of view boundaries for the camera. The value \(u_{\text{max}}\) is the maximum value of the control input at time \(k\), \(u_k = v_{k|0}\), which is the angular rate about the \(k^b\)-axis. This value limits how fast the robot can turn. The variable \(\nu\) is the intermediate feasible control input in the control sequence \(\pi\). The first theorem proves that the goal can stay in the field of view while satisfying the control input constraint, and that if the goal is in the field of view the robot will arrive within some small distance of the goal.

Remark

It is noted that the theorems and proofs will be stated using the states \([\alpha, d]^T\), where \(d\) is the distance from the robot to the goal and \(d = \frac{1}{\rho}\). This will make some of the arguments more intuitive.

**Theorem 6.7.1** Let the robot start with initial states \([\alpha_0, d_0]^T\) where \(|\alpha_0| \leq \alpha_{FOV} \leq \frac{\pi}{2}\) and \(d_0 > d_b\), where \(d_b = \frac{V \sin \alpha_{FOV}}{u_{\text{max}}}\) is the radius of a ball that is centered at the goal position. For all \(k > 0\) such that \(d_k > d_b\) there exists a control input \(u_k\) such that \(|u_k| < u_{\text{max}}\) and \(|\alpha_{k+1}| \leq \alpha_{FOV}\). Furthermore,
if \( u_k \) is selected so that \( |\alpha_{k+1}| \leq \alpha_{FOV} \) \( k = 0, 1, 2, \ldots \) then the robot is guaranteed to enter the ball at a finite time \( K_b \).

**Proof:** The discrete equations of motion for the states \( \alpha \) and \( \rho \) were derived in Chapter 3 and are defined in Equation 3.33. These equations are simplified here for the 2D case. Letting \( \beta = 0 \) and noting that \( d_k = \frac{1}{\rho_k} \) gives

\[
\begin{bmatrix}
\alpha_{k+1} \\
\rho_{k+1}
\end{bmatrix} =
\begin{bmatrix}
\alpha_k + \left( \frac{V \sin \alpha_k}{d_k} - u_k \right) \Delta t \\
d_k - (V \cos \alpha_k) \Delta t
\end{bmatrix}.
\] (6.32)

Since

\[
d_k > \frac{V \sin \alpha_{FOV}}{u_{max}}
\]

and \( |\alpha_k| \leq \alpha_{FOV} \) this means that

\[
d_k > \frac{V \sin |\alpha_k|}{u_{max}},
\]

which implies that

\[
u_{max} > \frac{V \sin |\alpha_k|}{d_k},
\]

therefore, there exists some value \( \hat{\epsilon}_k > 0 \) such that

\[
u_{max} = \frac{V \sin |\alpha_k|}{d_k} + \hat{\epsilon}_k.
\]

Let

\[
u_k = \frac{V \sin \alpha_k}{d_k} + \epsilon_k \text{sgn} (\alpha_k),
\]

where

\[0 \leq \epsilon \leq \hat{\epsilon}_k,
\]

then

\[
|\nu_k| = \left| \frac{V \sin \alpha_k}{d_k} + \epsilon_k \text{sgn} (\alpha_k) \right| \leq \frac{V \sin \alpha_k}{d_k} + \hat{\epsilon}_k
\] (6.33)

which implies that \( |\nu_k| = \nu_{max} \). From Equation 6.32

\[
\alpha_{k+1} = \alpha_k + \left( \frac{V \sin \alpha_k}{d_k} - u_k \right) \Delta t.
\]
Now substituting in $u_k$ as defined in Equation 6.33 gives

$$\alpha_{k+1} = \alpha_k - \varepsilon_k \text{sgn} (\alpha_k) \Delta t.$$ 

Therefore

$$|\alpha_{k+1}| \leq |\alpha_k| \Leftrightarrow |\alpha_k - \varepsilon_k \text{sgn} (\alpha_k) \Delta t| \leq |\alpha_k| \Leftrightarrow 0 \leq \varepsilon_k \leq \frac{|\alpha_k|}{\Delta t}.$$ 

Therefore, for any $\varepsilon_k$ such that

$$0 \leq \varepsilon_k \leq \min \left\{ \hat{\varepsilon}_k, \frac{|\alpha_k|}{\Delta t} \right\}$$ 

then

$$|u_k| \leq u_{\text{max}}$$ 

and

$$|\alpha_{k+1}| \leq |\alpha_k|.$$ 

Furthermore, since $|\alpha_0| \leq \alpha_{FOV}$ then this implies that $|\alpha_k| < \alpha_{FOV}, \forall \ k$ where $d_k > d_0$. This then implies that $\cos \alpha_k > 0, \forall \ k$ where $d_k > d_b$ which can be substituted in Equation 6.32 to give

$$0 < d_{k+1} = d_k - V \cos \alpha_k \Delta t < d_k - V \cos \alpha_{FOV} \Delta t$$

which means that $d_k$ decreases by at least $V \cos \alpha_{FOV} \Delta t$ at each time step, therefore, there is some time $K_b$ such that $d_k < d_b$.

Figure 6.12 shows two plots of the required angular rate $u$ as the robot travels to the goal with a constant velocity $V = 0.5 \text{ m/s}$, $u_{\text{max}} = 15 \text{ deg/s}$ and $\alpha_{FOV} = 30 \text{ deg}$. Figure 6.12(a) shows the angular rate plotted with distance $d$ and Figure 6.12(b) shows the angular rate plotted with inverse-depth $\rho$. The radius of the ball for arrival is shown in each plot with the vertical red line. Each plot shows multiple runs with different values of $\alpha$ where $\alpha$ is held constant over the entire run.
Figure 6.12: Plots of the control input \( u \) for different values of a constant \( \alpha \). Figure a plots with distance and Figure b plots with inverse-depth. The radius of the ball is shown by the vertical red line. The robot is shown to arrive at the ball without breaking the constraint \( |u| < u_{\text{max}} \).

To set up the discussion to prove that the control from Equation 6.28 will keep \( \alpha \in [-\alpha_{FOV}, \alpha_{FOV}] \) the properties of a desired cost function are described. Let the function \( \Phi(|\gamma|) \) be a stage cost function, where \( \gamma \in [0, 2\pi] \). The purpose of specifying the properties of the stage cost function is to define a cost function that will drive the robot to the goal while properly handling the limited sensor field of view. The sensor field of view is considered to be symmetric as in Theorem 6.7.1. Property 1 will classify the necessary requirements of the stage cost function \( \Phi(|\gamma|) \).

**Property 1**

The function \( \Phi(|\gamma|) \) has the following properties.

1.1 \( \Phi(|\gamma|) : [0, \pi] \rightarrow \mathbb{R} \).

1.2 \( \Phi(|\gamma|) \) is continuous.

1.3 \( \Phi(|\gamma|) \) is continuously differentiable.

1.4 \( \Phi(|\gamma|) \) has a global minimum value at \( \gamma_{\text{min}} \) where \( \gamma_{\text{min}} \in (0, \gamma_{FOV}) \) and \( \gamma_{FOV} < \frac{\pi}{2} \).

1.5 \( \Phi(|\gamma|) \) is strictly increasing \( \forall \gamma > \gamma_{\text{min}} \).
One example of a family of functions that meets the properties listed in Property 1 is the cost function defined in Equation 6.15.

The following lemma shows that when there are no obstacles, if $\alpha_0$ is inside the field of view then $\alpha_k \forall k \in [1, 2, \ldots, N]$ will be inside the field of view.

**Lemma 6.7.2** If $|\alpha_0| \leq \alpha_{FOV}$, $d_k > d_b, k \in [1, 2, \ldots, N]$ where $d_b$ is defined in Theorem 6.7.1 and the control input sequence $\pi_0 = \{v_{0|0}, v_{1|0}, \ldots, v_{N-1|0}\}$ is calculated by solving Equation 6.28 then the corresponding angle state sequence $|\alpha_k| \leq \alpha_{FOV}, k \in [1, 2, \ldots, N]$.

**Proof:** The minimization problem in Equation 6.28 will solve for the input sequence $\pi_0$ that minimizes the sum of the stage cost functions defined in Property 1. If the current state at time 0 is $[\alpha_0, d_0]^T$ with $d_0 > d_b$ then for any $|\alpha_0| \leq \alpha_{FOV}$ there is enough control authority (even under the constraint in Equation 6.30 as shown in Theorem 6.7.1) such that $|\alpha_1| \leq \alpha_{FOV}$. Let the optimal control sequence at time 0 be defined as

$$\pi_0^* = \{v_{0|0}, v_{1|0}, \ldots, v_{N-1|0}\},$$

(6.34)

where the input $v_{k|0}$ is the input that takes $\alpha_k$ to $\alpha_{k+1}$. The proof will consider two distinct cases of the initial angle $\alpha_0$.

**Case 1:** $|\alpha_0| \in (\alpha_{\min}, \alpha_{FOV}]$

In this case the angle $|\alpha_0|$ starts to the right of the minimum value location $\alpha_{\min}$ while still being inside the field of view. Now suppose that one of the input values $v_{k|0}, k \in [1, N]$ is a value such that $|\alpha_{k+1}| > \alpha_{FOV}$ which means that $\ell(|\alpha_{k+1}|) > \ell(|\alpha_k|)$. This is because from Property 1.5, $\ell(|\alpha|)$ is increasing $\forall |\alpha| > \alpha_{\min}$. If this is true then the sequence $\pi_0^*$ cannot be optimal because if the input were computed as $v_{k|0} = \frac{V \sin \alpha_k}{d_k}$, which is feasible then, $|\alpha_{k+1}| = |\alpha_k|$ and $\ell(|\alpha_{k+1}|) = \ell(|\alpha_k|)$. The total cost is the summation of the stage costs as shown in Equation 6.28, therefore the minimum value is the summation of each of the minimum stage costs. Using $v_{k|0} = \frac{V \sin \alpha_k}{d_k}$ therefore generates a cost function with lower cost proving that the control sequence $\pi_0^*$ is not optimal if $|\alpha_0| \in (\alpha_{\min}, \alpha_{FOV}]$ and any $|\alpha_k| > \alpha_{FOV} \forall k \in [1, 2, \ldots, N]$. 

128
Case 2: $|\alpha_0| \in [0, \alpha_{\text{min}}]$

Suppose that the input $\nu_{k|0}, k \in [1,N]$ is a value that forces $|\alpha_{k+1}| > \alpha_{\text{min}}$. However, if $\nu_{k|0}$ could be chosen such that $|\alpha_{k+1}| > \alpha_{\text{min}}$ then using a similar argument as in Case 1, $\ell(|\alpha_{k+1}|) > \ell(|\alpha_{\text{min}}|)$ so the sequence would not be optimal. This shows that when $|\alpha_0| < \alpha_{\text{min}}$ then $|\alpha_k| \leq \alpha_{\text{min}}, \forall k \in [1,2,\ldots,N]$.

Now using Lemma 6.7.2 it will be shown that using the control input $u_k = \nu_{0|k}$ repeatedly the robot will arrive inside the ball defined in Theorem 6.7.1.

**Theorem 6.7.3** If $|\alpha_0| \leq \alpha_{\text{FOV}}$ and $d_0 > d_b$ where $d_b$ is defined in Theorem 6.7.1 and the control input is calculated as defined in Lemma 6.7.2 then $|\alpha_k| \leq \alpha_{\text{FOV}} \forall k \in [1,2,\ldots,K_b]$ where $K_b$ is defined in Theorem 6.7.1.

**Proof:** Under the assumption of no obstacles and that the vehicle model is correct, the vehicle will use the input $u_0 = \nu_{0|0}$. From the result in Lemma 6.7.2, because $|\alpha_0| \leq \alpha_{\text{FOV}}$ then $|\alpha_1| \leq \alpha_{\text{FOV}}$.

Now, by induction suppose that $|\alpha_k| \leq \alpha_{\text{FOV}}$ and $d_k > d_0$, then using $u_k = \nu_{0|k}$ gives $|\alpha_{k+1}| \leq \alpha_{\text{FOV}}$ until some time $K_b$ when the vehicle will have arrived inside the ball.

Now it is shown that when $\alpha$ is outside the field of view it will enter the field of view in some finite time $K_{\text{FOV}}$. The following theorem describes this scenario.

**Theorem 6.7.4** If $|\alpha_0| \geq \alpha_{\text{FOV}}$ and $d_0 > d_b$ where $d_b$ is defined in Theorem 6.7.1 and the control input is calculated by solving Equation 6.28 subject to constraint 6.30 then there exists a time $K_{\text{FOV}}$ such that $|\alpha_k| \leq \alpha_{\text{FOV}} \forall k > K_{\text{FOV}}$ where $K_{\text{FOV}}$ is the time when the goal enters the field of view.

**Proof:** Let $|\alpha_0| > \alpha_{\text{FOV}}$ and let the optimal input sequence completed at time 0 be given by $\pi_0^*$. Suppose that a given input $\nu_{k|0}$ in the sequence is a value such that $|\alpha_{k+1}| \geq |\alpha_{k}|$ then, using Property 1.4 and 1.5 $\Phi(|\alpha_{k}|) \geq \Phi(|\alpha_{k-1}|)$, however, because $d_0 > d_b$ there exists a feasible control input $\nu_{k|0}$ such that $\alpha_{k+1} < \alpha_{k|0}$ which, using the properties of $\Phi$ implies that $\Phi(|\alpha_{k+1}|) < \Phi(|\alpha_{k}|)$ which contradicts the claim that the sequence $\pi_0^*$ is optimal. In fact, when $|\alpha_k| > \alpha_{\text{FOV}}$, the control input $\nu_{k|0}$ must be chosen such that $|\alpha_{k+1}| < |\alpha_k|$ or it will not be the minimum value as a result of Property 1.5. Then, by a similar argument to that used in

129
Theorem 6.7.1, the decrease in $|\alpha_k|$ is bounded away from zero, therefore there exists a time $K_{FOV}$ where $|\alpha_k| \leq \alpha_{FOV} \forall k \geq K_{FOV}$.

The following lemma will show the existence of an input such that the vehicle will not collide with a single obstacle. The conditions under which the input exists will be given.

Lemma 6.7.5 For a vehicle with a boundary $R_v$, maximum turn rate of $u_{max}$, constant velocity $V$, and a minimum turning radius of $R_{min} = \frac{u_{max}}{V}$, there exists a control input $u_k$ that satisfies the constraint $|u_k| \leq u_{max}$ that will steer the robot to not collide with a single obstacle with radius $R_{obs}$ if the initial condition of the vehicle satisfies

$$D_{k,1} > \sqrt{(R_{min} + R_v + R_{obs})^2 - R_{min}^2 - R_{obs}}.$$  

The distance $D_{k,1}$ is the distance from the robot at time $t = k$ to the boundary of the obstacle as defined in Equation 6.26.

Proof: To prove that the input $u_k$ exists and satisfies $|u_k| \leq u_{max}$ the worst-case scenario is analyzed. Let the robot at time $k$ be in the configuration shown in Figure 6.13 where the robot is heading straight towards the obstacle with a constant velocity. Using the Pythagorean theorem and solving for the distance $d_{min}$ gives

$$D_{min} = \sqrt{(R_{min} + R_v + R_{obs})^2 - R_{min}^2 - R_{obs}}.$$  

As can be seen from Figure 6.13 the distance $D_{min}$ is the minimum distance required by the robot to still turn and avoid the obstacle. This distance is worst-case because if the robot were not heading directly towards the obstacle then it could actually get closer to the obstacle and still avoid it. If the distance from the vehicle to the obstacle satisfies $D_{k,1} > D_{min}$ then the input $u_k$ can be chosen such that $u_k \leq u_{max}$ and the vehicle will avoid the obstacle.

Lemma 6.7.5 described the conditions for which an input will exist to allow the vehicle to avoid a single obstacle. Now the following theorem will show how using the MPC algorithm for control will drive the vehicle so that a single obstacle is avoided and the robot arrives at the goal.

Theorem 6.7.6 Let the robot be in an environment with a single obstacle with radius $R_{obs}$ and let the vehicle start in a position such that $D_{0,1} > D_{min}$ where $D_{0,1}$ is the distance from the vehicle to
the obstacle boundary at time \( k = 0 \) and \( D_{\text{min}} \) is defined in Equation 6.7. Also, let \( d_0 > d_b \) where \( d_b \) is defined in Theorem 6.7.1 and let \( |\alpha_0| < \alpha_{r,\text{FOV}} \), then using the control input calculated by solving Equation 6.28 subject to constraints 6.30 and 6.31, the distance \( D_{k,1} > R_v \forall k \). Furthermore, using the control input will also drive the vehicle such that the distance to goal \( d_k < d_b \) at time \( K_b \) as defined in Theorem 6.7.1.

**Proof:** Because \( D_{0,1} > D_{\text{min}} \) there exists an input such that the vehicle will avoid the obstacle and also, \( D_{0,1} > R_v \). Let the optimal input sequence be \( \pi_0^* \). Suppose that a given input \( \nu_k \) from \( \pi_0^* \) is a value such that \( D_{k,1} \leq R_v \), however, this conflicts with the design of the MPC algorithm of Equations 6.28 to 6.31 because this would violate the constraint in Equation 6.31. Because the control input exists, the controller will select a sequence \( \pi_0 \) such that \( D_{k,1} > R_v \forall k \in [1,2,\ldots,N] \). Now by induction, suppose that \( D_{k,1} > R_v \), then using \( u_k = \nu_{0,k} \) gives \( D_{k+1,1} > R_v \) until the vehicle has passed the obstacle. Now to examine the arrival of the vehicle within the ball defined by \( d_b \) in Theorem 6.7.1. There are two cases to examine while driving around the obstacle. The first is keeping the goal in the field of view while avoiding the obstacle. The second is losing the goal while avoiding the obstacle.
Case 1: Goal remains in the field of view

In this case $|\alpha| \leq \alpha_{FOV} \forall k$, then using the result from Lemma 6.7.2 and Theorem 6.7.1 the vehicle will arrive within the ball and $d_k < d_b$.

Case 1: Goal leaves the field of view

In this case, at some time $k^* \ |\alpha| > \alpha_{FOV}$. Because the obstacle is a circle with radius $R_{obs}$ there exists a finite time when the vehicle will be able to turn back to the goal and not collide with the obstacle. At this point, using the result from Theorem 6.7.4 the vehicle will turn such that at time $k_{FOV} \ |\alpha| \leq \alpha_{FOV}$. Then using the result from Lemma 6.7.2 and Theorem 6.7.1 the vehicle will arrive within the ball and $d_k < d_b$.

It has been shown that the vehicle is guaranteed to avoid a single obstacle if the starting distance from the boundary of the obstacle is greater than $D_{min}$ as defined in Equation 6.7 is satisfied. An environment with multiple obstacles is now examined. The method explained here is similar to the method in [74] where an environment with multiple obstacles is analyzed and the distance between obstacles is limited based on the minimum turn radius of the airplane. This work will also include the arrival of the robot within the ball of radius $D_b$.

To set up the problem space of multiple obstacles, let the position of obstacle $j$ in frame $F^i$ be given as

$$p_{o,j}^i = \begin{bmatrix} p_{o,jx}^i \\ p_{o,jy}^i \end{bmatrix} \tag{6.35}$$

and let the position of the robot be

$$q_r^i = \begin{bmatrix} q_{r,x}^i \\ q_{r,y}^i \end{bmatrix}. \tag{6.36}$$

The radius of obstacle $i$ is given by $R_{obs,i}$ and the minimum turn radius of the vehicle is $R_{min} = \frac{u_{max}}{V}$. Let the distance between the center of the vehicle at time $k$ and the boundary of obstacle $j$ be defined as

$$D_{k,j} = \sqrt{(q_{r,x}^i - p_{o,jx}^i)^2 + (q_{r,y}^i - p_{o,jy}^i)^2} - R_{obs,j}. \tag{6.37}$$
Also let the distance between the boundaries of two obstacles, obstacle \(i\) and \(j\) be defined as

\[
D_{o,ij} = \sqrt{(p_{o,ix}^i - p_{o,jx}^j)^2 + (p_{o,iy}^i - p_{o,jy}^j)^2} - (R_{obs,i} + R_{obs,j}) .
\] (6.38)

This distance is the minimum distance between the two boundaries of the obstacles. The following Theorem defines the environment in which the vehicle is guaranteed to avoid the obstacles. Let the minimum distance for avoidance of obstacle \(j\) be \(D_{min,j}\) which is defined in Equation 6.7.

**Theorem 6.7.7** If there are \(N_{obs}\) number of obstacles in the environment and the distance between each obstacle satisfies \(D_{o,ij} > \max(D_{min,i}, d_{min,j}), i = 1,2,\ldots,N_{obs}, j = 1,2,\ldots,N_{obs}, i \neq j\) and the initial position of the robot is such that \(D_{0,j} > D_{min,j}\), \(j = 1,2,\ldots,N_{obs}\), where \(D_{o,ij}\) is the distance from obstacle \(j\) to \(k\) given in Equation 6.38 and \(D_{k,j}\) is the distance from the robot to the closest boundary to obstacle \(j\) from Equation 6.37, then the control command computed in Equation 6.28 with constraints 6.30 and 6.31 will guarantee that the robot arrives within a ball of radius \(d_b\) as defined in Theorem 6.7.1 and also guarantees that the robot does not collide with any of the obstacles in the environment.

**Proof:** Consider the environment and vehicle configuration shown in Figures 6.14-6.19. Let the initial position of the robot be \(q_0^i\) such that \(D_{0,j} > d_{min,j} \forall j = 1,2,\ldots,N_{obs}\). In each figure the field of view of the camera is shown by the black lines extending from the camera on the robot. The locations of \(\alpha_{min}\) and \(-\alpha_{min}\) are shown by the green lines. There are two obstacles shown by the solid red circles. The red circle around each obstacle is the closest point that the center of the robot can get to the obstacle. This distance is shown as \(R_v\). The goal is the green circle. The black circle around the goal is the ball of radius \(d_b\). As the robot travels from position \(q_0\) to \(q_A\) as shown in Figure 6.14 it is traveling on the desired path that will keep the goal at the angle \(-\alpha_{min}\). This is the case that is proven in Lemma 6.7.1, however as the robot nears point \(q_A\) it needs to avoid obstacle \(O_1\). The robot will start to avoid this obstacle (Theorem 6.7.6) and at point \(q_A\) the goal leaves the field of view entirely as shown in Figure 6.15. At this point the robot will avoid the obstacle (Theorem 6.7.6) while trying to minimize the cost function in Equation 6.28. This means that the robot will drive as close to the obstacle as allowed because this will minimize the angle \(\alpha\). This is the path shown from point \(q_A\) to \(q_B\). When the robot arrives at point \(q_B\) shown in
Figure 6.16 the goal has entered the field of view again and is at the angle $-\alpha_{\text{min}}$ which happens as shown in Theorem 6.7.4. From this point, the robot will drive to the keep the goal at the $-\alpha_{\text{min}}$ location. The process is then repeated for obstacle $O_2$ where at point $q_C$ shown in Figure 6.17, the goal has left the field of view. The robot then travels from $q_C$ to $q_D$ while minimizing the angle $\alpha$ until the goal enters the field of view again and at point $q_D$ as shown in Figure 6.18. At this point the goal is at the angle $\alpha_{\text{min}}$ and remains there until the robot arrives at the ball of radius $d_b$ as shown in Figure 6.19 and proven in Theorem 6.7.1. This process of avoiding obstacles and reacquiring the goal in the field of view can be repeated for as many obstacles as necessary as long as the environment satisfies $d_{o,ij} > \max \left( d_{\text{min},i}, d_{\text{min},j} \right)$. The constraint on the distance $d_{o,ij}$ ensures that the vehicle will always be farther than the minimum distance from each obstacle when it has to start to turn to avoid it. Then, using Theorem 6.7.6 it is shown to avoid the obstacle. The vehicle is then able to avoid each obstacle as shown in Theorem 6.7.6. In this environment, the robot will then arrive with the ball of radius $d_b$ of the goal as defined in Theorem 6.7.1.

![Figure 6.14: Robot motion sequence for a robot starting at point $q_0$ and driving toward the goal shown by the green circle. The environment contains two obstacles that obstruct the ideal path of the robot. The robot is shown at initial position $q_0$ while the goal is still at the desired angle $-\alpha_{\text{min}}$.](image)
Figure 6.15: Robot motion sequence showing the robot at position $q_A$ where the goal leaves the field of view because of the obstacle in the desired path.

Figure 6.16: Robot motion sequence showing the robot at position $q_B$ when the goal is back in the field of view and at the desired angle $-\alpha_{\text{min}}$.

6.8 Simulation Results

A simulator was developed using Matlab to simulate the MPC derived in this Chapter. The simulation shows the behavior of the vehicle in various scenarios including with and without obstacles. The simulations that follow in this section are with known states, there is no estimation of the goal in these simulations. This is to show the behavior of the designed controller independent of the goal estimation. The simulation uses the Matlab function *fmincon* which is a function for nonlinear multivariable constrained optimization. There are five different simulation environments
Figure 6.17: Robot motion sequence as the robot drives around the second obstacle in the environment and arrives inside the ball around the goal. The robot is at location $q_C$ when the goal leaves the field of view because of the obstacle.

Figure 6.18: Robot motion sequence as the robot drives around the second obstacle in the environment. The robot is passing the obstacle at position $q_D$ with the goal in the field of view.

that are shown here with the robot starting with the same position and heading in each experiment. The goal location also remains the same for each experiment. The difference with each experiment is the location and types of obstacles in the environment. The different experiments are:

1. Environment with no obstacles.

2. Single circular obstacle and the goal stays in the field of view

3. Two circular obstacles and the goal leaves the field of view
4. Multiple circular obstacles and a barrier and the goal leaves the field of view

5. Multiple circular obstacles that are placed closer than the minimum obstacle distance of $D_{o,ij} > \max(D_{\text{min},i}, D_{\text{min},j})$ as described in Theorem 6.7.7

For these experiments, the robot is traveling on a 2D flat surface and the robot has a simulated LiDAR that detects circular obstacles and wall barriers. The properties that remain constant for each simulation are shown in Table 6.1.

### 6.8.1 No Obstacles

This is a very simple simulation to show how the robot behaves when there are no obstacles. The results of this simulation are shown in Figure 6.20. As can be seen in Figure 6.20(a) and Figure 6.20(b), immediately after starting, the robot turns to control the angle $\alpha$ to be at the minimum value $\alpha_{\text{min}}$. The control input is shown to always be within the constraint $|u_k| \leq u_{\text{max}}$ for the entire run. The control input arrives at $u_{\text{max}}$ as the robot enters the ball as was shown in Theorem 6.7.1. Because there are not obstacles, the robot can maintain this angle almost the entire path until it is inside the ball around the goal. As expected, the cost (shown in Figure 6.20(c)) will be at a minimum while $\alpha = \alpha_{\text{min}}$ and then when $\alpha \neq \alpha_{\text{min}}$ the cost will increase. The path of the robot while approaching the goal is shown in Figure 6.20(d). The goal location and the ball around the goal are shown in green.
Table 6.1: Parameters used in the Matlab MPC simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Starting Position ((x, y))</td>
<td>((0,0) \text{ m})</td>
</tr>
<tr>
<td>Vehicle Starting Yaw ((\psi))</td>
<td>(-5 \text{ deg})</td>
</tr>
<tr>
<td>Target Position ((x, y))</td>
<td>((50,0) \text{ m})</td>
</tr>
<tr>
<td>Vehicle Velocity</td>
<td>(1.0 \text{ m/s})</td>
</tr>
<tr>
<td>Maximum Angular Velocity ((u_{max}))</td>
<td>(17 \text{ deg/s})</td>
</tr>
<tr>
<td>Camera Field of View (\alpha_{FOV})</td>
<td>(61.5 \text{ deg})</td>
</tr>
<tr>
<td>Cost Function (\alpha_0)</td>
<td>(30.75 \text{ deg})</td>
</tr>
<tr>
<td>Ball Radius around Goal (d_b)</td>
<td>(1.7 \text{ m/s})</td>
</tr>
<tr>
<td>Simulation time step (\Delta t)</td>
<td>(0.1 \text{ s})</td>
</tr>
<tr>
<td>Look Ahead Steps ((N))</td>
<td>(30)</td>
</tr>
<tr>
<td>Vehicle Boundary ((R_v))</td>
<td>(0.5 \text{ m})</td>
</tr>
<tr>
<td>LiDAR Angular Range</td>
<td>(90 \text{ deg})</td>
</tr>
<tr>
<td>LiDAR Angular Resolution</td>
<td>(4.5 \text{ deg})</td>
</tr>
</tbody>
</table>

6.8.2 Single Circular Obstacle

This simulation has a single circular obstacle with a radius of \(0.5 \text{ m}\) and a position of \((8.0, -3.6) \text{ m}\). Figure 6.21(a) shows the angle \(\alpha\) and how it stays in the field of view over the entire run. The control input \(u\) is shown in Figure 6.21(b) and it is seen that when the robot is sensing the obstacle starting at \(t = 5 \text{ s}\) the input is changing as it is trying to find the best route around the obstacle. The noise seen in the input can be partially attributed to the fact that only discrete points are sensed on the outside of the obstacle. Two zoomed in views of the obstacle sensing are shown in Figure 6.21(e) and Figure 6.21(f). The current optimal path \(\pi_k\) is shown by the black line in front of the vehicle. The dots along the line are the positions at time \(k\). The obstacle is shown in red with the keep out zone around the obstacle shown in cyan. The keep out zone is the area around the obstacle that is enlarged by the vehicle boundary \(R_v\). As seen in Figure 6.21(e), the path generated from \(\pi_k\) enters the keep out zone around the obstacle. This would seem to be an error, but this is due to the fact that the information about the obstacle are the sensed points (shown by the magenta circles around the obstacle) so while the optimal path enters the keep out zone in the future, it is still at least a distance of \(R_v\) away from the closest sensed point on the obstacle. Figure 6.21(f) shows that as the robot drives around the obstacle that it does
Figure 6.20: Simulation run with no obstacles. The angle $\alpha$ starts inside the camera field of view and once driving the robot controls $\alpha$ to $\alpha_{\text{min}}$. This angle is achieved until the robot is inside the ball around the goal. The angle $\alpha$ is shown in Sub-figure (a). The control input $u_k$ is shown in Sub-figure (b) while the cost is shown in Sub-figure (c). The path the robot took to the goal is shown in Sub-figure (d).

stay out of the keep out zone around the obstacle. The path of the robot to the goal is shown in Figure 6.21(d) while the cost is shown in Figure 6.21(c). The cost is not minimal while the robot drives around the obstacle, but it is able to avoid the obstacle and then bring the cost back to the minimum.
Figure 6.21: Simulation run with a single obstacle of radius 0.5 m and a position of $(8.0, -3.6)$ m. The robot successfully drives around the obstacle while keeping the goal inside the field of view the entire time. Sub-figure (a) and (b) show the angle $\alpha$ and the control input $u$ over the run. Sub-figure (c) shows the cost from the MPC problem at each time step. Sub-figure (d) shows the path of the robot to the goal. Sub-figure (e) shows a zoomed in view at $t = 5.7$ s and Sub-figure (f) shows a view at $t = 8.0$ s.
6.8.3 Two Circular Obstacles

This simulation example shows the response of the vehicle to two different obstacles. These obstacles each have a radius of 0.5 m. The location of the first obstacle is (8.0, −3.4) m and the location of the second obstacle is (20.0, 2.3) m. The results from this simulation are shown in Figure 6.22. The angle $\alpha$ is shown in Figure 6.22(a) along with the input $u$ in Figure 6.22(b). The cost over the whole simulation is shown in Figure 6.22(c) and the path of the vehicle while driving to the goal is shown in Figure 6.22(d). This simulation shows two different responses around an obstacle. The first response of the robot keeps the goal in the field of view. The response of the vehicle while sensing the first obstacle is shown in Figure 6.22(e). Here the vehicle had enough room around the obstacle to make the turn to the left of the obstacle to keep the cost function as low as possible which keeps $\alpha$ in the field of view. The response around the second obstacle is shown in Figure 6.22(d). Here the vehicle was too far outside the obstacle to be able to turn back inside to keep $\alpha$ inside the field of view. The main reason for this, is that the control input has to satisfy the constraint $|u_k| < u_{max}$. As the robot goes around the vehicle it keeps as close to it as possible to keep the cost as low as possible. Once past the second obstacle the vehicle is able to keep the angle $\alpha$ at $\alpha_{min}$ until entering the ball shown by the green circle in Figure 6.22(d).

6.8.4 Barrier Obstacle

This simulation shows multiple circular obstacles and a barrier obstacle. The barrier obstacle is shown because this can represent a more realistic type of obstacle that may be encountered by the actual vehicle. Figure 6.23 and Figure 6.24 show the results from this simulation. Figure 6.23(d) shows the layout of the obstacles along with the path that the robot took in the simulation. The circular obstacles have a radius of 0.5 m and the barrier is a line with zero thickness. The vehicle boundary is shown around the circular obstacles and around the barrier in cyan. The angle $\alpha$ is shown in Figure 6.23(a) while the input $u$ is shown in Figure 6.23(b). The cost at each time step over the whole simulation is shown in Figure 6.23(c). The vehicle approaches the first obstacle and is able to turn and keep the angle $\alpha$ inside the field of view as shown in Figure 6.24(a). The robot then approaches the second and third obstacles (shown in Figure 6.24(b) and Figure 6.24(c)) and is once again able to turn such that $\alpha$ remains in the field of view. The vehicle then approaches
Figure 6.22: Simulation run with two obstacles of radius 0.5 m and a positions of $(8.0, -3.4)$ m and $(20.0, -2.3)$ m. The robot successfully drives around the first obstacle while keeping the goal inside the field of view and then the angle $\alpha$ is forced to leave the field of view while driving around the second obstacle. Sub-figure (a) and (b) show the angle $\alpha$ and the control input $u$ over the run. Sub-figure (c) shows the cost from the MPC problem at each time step. Sub-figure (d) shows the path of the robot to the goal. Sub-figure (e) shows a zoomed in view at $t = 6.0$ s and Sub-figure (f) shows a view at $t = 19.9$ s.
the barrier as shown in Figure 6.24(d) and the robot is able to avoid the barrier and follow as close to it as possible as shown in Figure 6.24(e). It is interesting to note that in Figure 6.24(e) it is seen that the end of the optimal path (shown in black) starts to curl back in toward the barrier. This is because at that current time the MPC look ahead path is reaching farther than the sensed obstacle so it is trying to get $\alpha$ back into the field of view. Finally, as shown in Figure 6.24(f) the vehicle reaches the end of the barrier and is able to turn back so that $\alpha$ enters the field of view. The vehicle is then able to drive unobstructed to arrive within the ball at the goal.

### 6.8.5 Multiple Circular Obstacles

One last simulation example is shown here to show how the robot handles multiple circular obstacles that don’t satisfy the constraint $d_{o,jk} > \max \left( d_{\min,j}, d_{\min,k} \right)$ described in Theorem 6.7.7. There are six total circular obstacles with a radius of 0.5 m in this simulation. The results from the simulation are shown in Figure 6.25 and Figure 6.26. The layout of the obstacles is shown in Figure 6.25(d). The angle $\alpha$ is shown in Figure 6.25(a) and the control input $u$ is shown in Figure 6.25(b). The cost at each time step is shown in Figure 6.25(c). The robot approaches the first group of obstacles and avoids the first and second obstacle as shown in Figure 6.26(a). Notice that in Figure 6.26(a) the optimal path computed from MPC (shown in black) is trying to drive around the second obstacle and it has yet to sense the third obstacle in the group. Then, in Figure 6.26(b) the robot senses the third obstacle, therefore the path will not let the robot go between the two obstacles. These obstacles are too close for the robot to fit between them. Figure 6.26(c) then shows the robot driving around the last of the obstacles in the first group. The robot then approaches the second group of obstacles and avoids the first one as shown in Figure 6.26(d). Figure 6.26(e) shows the first and second obstacles of the second group, but the MPC path does not go in between the two obstacles. This is because of the control input constraint. The robot would not be able to turn fast enough to make it through without a collision, therefore it continues forward. Figure 6.26(f) shows the robot being able to navigate between the last two obstacles. This is possible because the spacing between them is large enough that even with the control input constraint of Equation 6.30 the robot can make the turn without a collision. It is noted that under this configuration the robot was able to successfully avoid all obstacles even though the obstacle spacing constraint from Theorem 6.7.7 was not met, however, this does not prove that it will work in all configurations.
Figure 6.23: Simulation data for a run with circular obstacles of radius 0.5 m and positions of (8.0, −3.4) m, (20.0, 3.0) m and (22.0, 2.0) m. The barrier has a start point of (23.0, −2.0) m and an endpoint of (30.0, 2.5) m. The robot successfully drives around the first obstacle while keeping $\alpha$ within the field of view. The robot also avoids the second and third obstacles with $\alpha$ in the field of view. Then the robot comes to the barrier and is forced to follow the barrier which causes $\alpha$ to leave the field of view. Once past the end of the barrier the robot is able to turn back towards the goal and arrives within the ball around the goal. Sub-figure (a) and (b) show the angle $\alpha$ and the control input $u$ over the run. Sub-figure (c) shows the cost from the MPC problem at each time step. Sub-figure (d) shows the path of the robot to the goal.
Figure 6.24: Zoomed in views at different times during the simulation shown in Figure 6.23. The simulation time of each picture is labeled. The dark blue line is the path history of the robot while the black line with dots represents the optimal path computed by the MPC algorithm at that time. The dots represent the discrete positions of the robot after using the computed inputs. The obstacles are shown in red with the vehicle boundary around the obstacle shown in cyan. The LiDAR returns from the obstacles are shown in magenta.
Figure 6.25: Simulation data for a run with circular obstacles of radius 0.5 m at positions of (7.0, −2.0) m, (8.0, −3.4) m, (8.5, −5.3) m, (20.0, −8.0) m, (22.0, −9.4) m and (23.5, −11.8) m. The robot successfully avoids all of the obstacles and arrives at the ball around the goal. The robot is not able to keep the angle $\alpha$ inside the field of view because of the layout of the obstacles. The robot is able to successfully avoid obstacles that don’t meet the requirement from Theorem 6.7.7 in this scenario. Sub-figure (a) and (b) show the angle $\alpha$ and the control input $u$ over the run. Sub-figure (c) shows the cost from the MPC problem at each time step. Sub-figure (d) shows the path of the robot to the goal.
Figure 6.26: Zoomed in views at different times during the simulation shown in Figure 6.25. The simulation time of each picture is labeled. The dark blue line is the path history of the robot while the black line with dots represents the optimal path computed by the MPC algorithm at that time. The dots represent the discrete positions of the robot after using the computed inputs. The obstacles are shown in red with the vehicle boundary around the obstacle shown in cyan. The LiDAR returns from the obstacles are shown in magenta.
6.9 Summary

This chapter has described the MPC method designed to drive the robot to the goal location. The vehicle model is used to predict future locations of the robot. The cost function was designed and is based on the observability of the goal. MPC solves for the path of the robot that will minimize the uncertainty of the goal position estimate in the robot-body frame. The properties of the cost function were described. The cost function explicitly accounted for the constraint on the sensor field of view. The controller also takes in obstacles that are sensed from a 2D planar LiDAR. The obstacles are hard constraints in the MPC problem. The maximum angular rate of the vehicle is an input constraint.

The arrival of the robot to the goal was examined. Different theorems were shown to prove that the robot will arrive at the goal under certain conditions. The robot will arrive within some small distance of the goal based on the velocity of the robot and the maximum turn rate of the robot. The conditions for obstacle avoidance were also given. Simulation examples of different environments were presented to show the behavior of the robot under certain scenarios.
CHAPTER 7. EXPERIMENTAL RESULTS

7.1 System Overview

The system described in this work has been implemented on the hardware presented in Chapter 2. This chapter will discuss the results of experiments run on the Packbot robot platform. The full system is shown in Figure 7.1. The overall system block diagram is shown in Figure 7.2 including the software blocks and hardware items. The dark blue blocks indicate the Packbot and different sensors used throughout the experiments. The Linux laptop is shown in the block diagram as the orange box with the dashed outline. Each of the sensors in the system connects to the Linux machine via a USB cable. The Windows laptop is shown as the gray block with the dashed outline and it communicates with the Linux machine and the Packbot through two separate Ethernet connections.

7.1.1 Robot Operating System

The Robot Operating System (ROS) is implemented on the Linux laptop. ROS allows for many different software applications to run independently on the same machine or to run distributed across multiple platforms. It also provides a simple communication framework using a publish-subscribe format. An independent software application is called a ROS node. A ROS node is defined by the inputs and outputs and then simply subscribes to the desired messages for the inputs and publishes the output messages. Each of the blocks within the Linux machine shown in Figure 7.2 is a stand-alone ROS node. The functions of each of the nodes will be described here.

7.1.2 Sensor Interface

ROS nodes communicate using ROS topics. Each topic is specified to be a certain ROS message. The topic is simply an instantiation of a ROS message. For example, there is a ROS
message defined for an IMU message, therefore a ROS node is required to talk to the IMU and generate an IMU message that can be sent out to the system as a ROS topic. Then any other node in the system can access the IMU topic simply by subscribing to that specific topic. For this reason, sensor driver nodes are required for each sensor used. These sensor drivers are used to communicate with the sensor and translate the raw data into a message defined within the ROS system.

Each of the sensor driver nodes are shown in Figure 7.2 as the light blue blocks. ROS is an open-source framework where many members of the ROS community collaborate to generate code that is shared with others in the robotics community. Because of this, there are many ROS nodes that have been created to interface with certain sensors. Each of the sensor drivers shown in the system block diagram was created by others and posted for use. These nodes will briefly be described here.
Figure 7.2: Overall system block diagram. The diagram shows hardware and software blocks. The two computers are shown in dashed outlines. The orange box is the Linux laptop and the gray block is the Windows laptop. The dark blue blocks are hardware (sensors and robot) and the light blue blocks within the Linux box are the required software drivers used to interface to the sensors. The green blocks are the software applications that run certain algorithms. The two computers communicate over an Ethernet connection. The windows machine also talks to the Packbot over an Ethernet connection.

**IMU Driver**

The IMU is a KVH 1725 IMU. This driver was created by the Terrestrial Robotics Engineering and Controls (TREC) group at Virginia Tech [84] and has been modified slightly for use with the KVH 1725. The output IMU ROS topic is available at 100 Hz.

**Phidget Encoder Driver**

The encoder quadrature lines connect to the Phidget encoder board which plugs into the Linux laptop. The Phidget drivers are from the Phidget ROS package [85]. This package was
developed by the board manufacturer for Phidget hardware. This package supports the high-speed encoder board installed on the Packbot. The output ROS topic is available at 125 Hz.

**Camera Driver**

The camera driver for the Xtion PRO LIVE camera is the OpenNI 2 ROS package [86]. This driver supports the image acquisition for both the monocular camera and the RGB-D camera, however, only the monocular image is subscribed to by other nodes. The camera image ROS topics are available at a rate of 30 Hz.

**Laser Driver**

The Hokuyo laser driver is a supported ROS package that was built by Hokuyo [87]. This driver generates a laser scan at a rate of 40 Hz.

**GPS Driver**

The Trimble Ag 162 GPS sensor outputs a standard NMEA message on the serial line through a RS-232 to USB device into the Linux laptop. This NMEA message is then parsed by a NMEA ROS package [88]. This package returns two NMEA sentences: a GGA message and a VTG message. The GGA NMEA message contains information about the position of the receiver. This includes the latitude, longitude, and altitude of the receiver as well as the mode and number of satellites. The VTG NMEA message contains information about the velocity and heading of the receiver. This provides information for a map with a known origin to produce a position \((x, y, z)\) in the local map frame. The logged position data is in the local map coordinates.

### 7.1.3 ROS Node Architecture

The green blocks in Figure 7.2 represent the nodes associated with the estimation and control of the vehicle. Each sensor, estimation, and control node will be described here.
Velocity Node

The Phidget driver nodes output the delta counts and the total counts of the track encoder at a maximum rate of 125 Hz. If the encoders do not move, then no message of counts is sent. The Velocity node, as labeled in Figure 7.2, takes in the encoder counts from both the left and right track and uses that information to calculate the linear velocity of the track. The encoders are incremental encoders so each time the encoder is powered on, the counts start at 0. The counts correspond to an angular distance traveled by the track drive wheel where the encoder is mounted. It is necessary to convert the angular motion of the encoder into a linear velocity of the track. A simple equation to convert angular velocity into linear velocity is

\[ V_{\text{track}} = R_d \omega_d, \]  

where \( R_d \) is the radius of the drive wheel, \( \omega_d \) is the angular velocity of the drive wheel and \( V_{\text{track}} \) is the linear velocity of the track. This conversion requires the knowledge of \( \omega_d \). Since this value is not returned directly from the encoder, an algorithm is necessary that computes \( \omega_d \) from the encoder counts. The Velocity node algorithm is a method developed in [89]. This algorithm saves a history of encoder counts over time. A third-order polynomial function is then fit to the saved encoder history data. This produces a function to describe the position of the encoder in time. This gives a continuous function that has a continuous first derivative. The angular velocity of the encoder is then calculated by taking the derivative with respect to time and substituted in the desired time for the velocity. The output is the angular velocity \( \omega_d \). The track velocity is then calculated with \( \omega_d \) as shown in Equation 7.1.

The linear track velocity \( V_{\text{track}} \) is calculated for each track on the vehicle. The linear velocity of the vehicle is then calculated as

\[ V = \frac{V_{\text{track},l} + V_{\text{track},r}}{2}, \]  

where \( V_{\text{track},l} \) is the linear track velocity of the left track and \( V_{\text{track},r} \) is the linear track velocity of the right track. A similar method calculates the angular velocity of the vehicle around the body \( z \)-axis, however, with a skid-steer type vehicle like the Packbot this measurement can be very noisy due to
slippage variation on different surfaces. For this reason, only the linear velocity of the robot along the \( i^b \)-axis is used. This calculated velocity is passed to the EKF node as shown in Figure 7.2.

**Tracker Node**

The Tracker node is responsible for tracking the goal and other landmarks in the image frame. The algorithm associated with this node is described in detail in Chapter 5. As seen in Figure 7.2 there are three inputs into this node and one output. The image input comes from the camera driver and is published at a rate of 30 Hz. The Tracker node also receives the current states from the EKF. This is important because the current states are required in the Tracker to calculate the expected locations of the goal and other landmarks that are currently being tracked. The Tracker then updates the goal and landmark positions in the current image and sends those measurements out to the EKF. It is noted that the states message and the camera image message run asynchronously. The Tracker updates the expected image locations each time a states message is received, but only updates the current tracking estimate when a new image is received from the camera driver. The third input into the Tracker is the Remove State message from the EKF node and will be explained in the next section.

**Extended Kalman Filter**

The EKF was explained in detail in Chapter 3. The EKF node receives inputs from the Velocity node and from the IMU driver. Both of these inputs are used in the prediction step. These inputs are received asynchronously, therefore, the body angular rates \((p, q, r)\) are set inside the filter when the IMU message is received and a prediction step is run. When a velocity measurement \( V \) is received from the Velocity node the internal input value in the EKF for the velocity is set and another prediction step is run. It is noted that the inputs are not set to 0 after each prediction step but are held constant between samples. This allows for the prediction steps to be run separately between the two messages. The EKF node also receives image measurements from the Tracker node. When a Tracker node message is received, the EKF runs the measurement update step. This message is sent at the same rate as the camera image, which is 30 Hz.
The number of states in the EKF is set based on the number of measurements that are received from the Tracker node. In theory, the number of states can continue to increase indefinitely with the EKF being able to handle the increased number properly. In practice, however, there is a limitation to how many states can be used in the EKF due to the computational requirements of the computer running the EKF algorithm. For this work, the robot is trying to arrive at the goal point, not trying to create a map of the environment, therefore, certain states can be removed from the filter if they are no longer needed by the vehicle to improve the estimate of the goal location in the body frame. For this reason, there is an input/output line in the system diagram going from the Tracker node to the EKF node. This means that either the Tracker or the EKF can remove states from the state vector. When a node removes a state from the state vector it has to tell the other node that the state has been removed to maintain correspondence between the current image measurements and the current state vector.

The Tracker node can only remove a state from the current state vector if that state is inside the field of view of the camera and the Tracker is unable to reacquire the landmark for a certain period of time. For this to happen, the landmark could have either left the field of view from the robot motion and then re-entered at a later time or the tracker simply could have lost track of the landmark while it was still in the field of view. In either case, the Tracker will try and reacquire using the template matching described in Chapter 5. If the template matching fails, then the state is removed and the message is sent on to the EKF node.

The EKF can also remove states from the state vector. This happens when it is decided that a landmark is going to be behind the robot. This type of landmark will most likely not be useful to the robot because the robot is constantly trying to find the path that will lead to the goal and will most likely not be turning back towards the landmark. This landmark is also not useful because the limited field of view of the camera. This is checked by removing state \( i \) when \( |\alpha + \theta_i| > \alpha_{\text{max}} \) and \( |\beta + \eta_i| > \beta_{\text{max}} \). Once the landmark is removed then the Remove State message is sent to the Tracker Node.

**Obstacle Detection Node**

The Obstacle Detection node takes in the laser scan from the laser driver and calculates the position of any obstacles that might be in front of the robot. The method of calculation was
described in detail in Chapter 6. The Obstacle Detection node runs at 40 Hz which is the rate at which it receives laser scans from the laser driver node. The output of this node is used by the MPC block.

**Model Predictive Control Node**

The MPC algorithm described in Chapter 6 runs in the MPC node block. This node is responsible for calculating the drive commands that are forwarded to the Packbot for control. It takes the current states and obstacles as inputs and then solves the MPC optimization problem. This node is processor intensive, therefore this node runs at 5 Hz with the number of look-ahead steps \( N = 40 \). The output of this node goes to the Packbot UDP node.

After testing the Packbot in environments with obstacles, the data was analyzed and found that the optimal control sequence \( \pi \) became significantly noisy when the look-ahead path was first starting to come in contact with the obstacle. After further testing it was found that the optimal path was oscillating between driving to the left or to the right of the obstacle. During most of the tests the vehicle would oscillate for a small time period before settling on a direction around the obstacle. There were some instances when the oscillation would happen until the vehicle was close enough to the obstacle that the input constraint would not allow the path to jump from one side of the obstacle to the other without violating the constraint.

To overcome this problem, an inertia cost on the input is used. This is done by penalizing the current input vector \( \pi_k \) if it is different than the previous optimal input vector \( \pi_{k-1} \). The norm of the difference of the two input vectors is used with an inertia gain in the cost function. The new cost function then becomes

\[
V(x, \pi) = \sum_{i=0}^{N-1} \ell(x_i, v_i) + R_I \|\pi^* - \pi\|, \tag{7.3}
\]

the value \( R_I \) is the inertia cost gain and \( \pi^* \) is the optimal input sequence from the last time step and \( \pi \) is the current input sequence. Adding this into the cost function helps stabilize the MPC algorithm when approaching obstacles.
**Packbot UDP Node**

The Packbot User Datagram Protocol (UDP) node is simply a translator node. It receives the current command from the MPC node and puts it on the wire (Ethernet) to send to the Windows machine for control. The desired linear and angular velocities of the robot are then sent as fast as they are received from the MPC node.

### 7.1.4 Packbot Control

The Packbot Control software is a Windows executable that accepts the desired velocities sent via UDP from the Packbot UDP node. These commands are then forwarded on to the Packbot and the Packbot hardware controls the track velocities internally.

### 7.2 Results

Many experiments were run with the robot, ranging from simple flat ground scenarios with no obstacles where the goal never left the field of view to off-road environments with obstacles where the goal had to leave the field of view. The results from two experiments in particular will be explained in detail here. Both experiments included obstacles resulting in the goal leaving the field of view, but the first experiment was run on flat ground while the second experiment was run off road.

#### 7.2.1 Flat Ground

This experiment was run on a paved track with multiple obstacles and many large barriers around a portion of the track. A view of the environment is shown in Figure 7.3. There are two individual obstacles up close to the robot that were placed to ensure that the robot would have to avoid them and lose the goal from the field of view of the camera. This environment and conditions were chosen because of the barriers, since they would require the robot to estimate the position of the goal for a significant amount of time while driving around the barriers. The parameters used for this experiment are shown in Table 7.1.

157
Figure 7.3: Initial view for the paved track experiment. The goal was selected by the user to be the blue can near the right side of the image. Two obstacles were placed in the desired path of robot. These are the white and green barrels near the middle of the image. The concrete barriers are seen in the background.

The path of the vehicle and the layout of the obstacles and barriers are shown in Figure 7.4. The path of the vehicle is red when the goal is outside the field of view and the path is blue when the goal is inside the field of view. The start position of the vehicle is shown by the black circle, while the goal position is shown by the green circle. The barriers are shown in black and the obstacles are shown as the red circles. The two first obstacles were placed such that the robot would have to maneuver early and lose the goal early in the experiment. The robot was able to drive around the obstacles and quickly reacquire the target as can be seen by the blue path that shows up after the first obstacles. The robot then reaches the barriers and once again has to avoid the obstacles and reacquire the target. It is noted that cones were placed between the barriers near the corner so the robot would not make it through the barriers and reacquire the goal earlier. This is part of the reason that the path oscillates as the robot drives along the barrier because the look-ahead path from the MPC controller is trying to turn back toward the goal whenever possible and for a brief moment between the barriers the MPC can start making it through the barrier until the cone is sensed and it has to continue forward. Once the robot reaches the corner of the barrier it is able
Table 7.1: Parameters used for the flat ground experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Starting Position</td>
<td>$(-26.1,72.2,0.0)^T$ m</td>
</tr>
<tr>
<td>Goal Position</td>
<td>$(23.4,69.3,0.0)^T$ m</td>
</tr>
<tr>
<td>Image Measurement Noise ($\sigma_{x_i}, \sigma_{y_i}$)</td>
<td>(4,4) pixel</td>
</tr>
<tr>
<td>Process Noise Parameters $\delta_1, \delta_4$</td>
<td>50.0</td>
</tr>
<tr>
<td>Process Noise Parameters $\delta_2, \delta_3$</td>
<td>5.0</td>
</tr>
<tr>
<td>Process Noise Parameters $\delta_p, \delta_q$</td>
<td>50.0</td>
</tr>
<tr>
<td>Initial Depth Values $\rho_0$</td>
<td>0.1</td>
</tr>
<tr>
<td>Initial Variance $[\sigma_\alpha, \sigma_\beta, \sigma_{\rho_0}]$</td>
<td>[0.05,0.05,0.1]</td>
</tr>
<tr>
<td>Maximum Number of States</td>
<td>51</td>
</tr>
<tr>
<td>MPC $dt$</td>
<td>0.2 s</td>
</tr>
<tr>
<td>Desired Linear Velocity</td>
<td>0.5 m/s</td>
</tr>
<tr>
<td>Max Angular Rate $u_{max}$</td>
<td>0.25 rad/s</td>
</tr>
<tr>
<td>$\alpha_0$</td>
<td>0.45 rad/s</td>
</tr>
<tr>
<td>Look Ahead Steps $N$</td>
<td>40</td>
</tr>
<tr>
<td>Inertia Cost Gain $R_I$</td>
<td>40.0</td>
</tr>
</tbody>
</table>

to turn towards the goal again and the goal is reacquired. From then on, the robot is able to drive with the angle $\alpha$ at the desired location until arriving at the goal.

A close-up view of the obstacles and barriers is shown in Figure 7.5. This shows in more detail the path that the robot took around the initial two obstacles and how the MPC wanted to drive between the barriers until the cones were detected.

The state estimate $\rho_g$ is shown in Figure 7.6. The true value for the inverse depth is shown in blue while the estimate is shown in black and the 95% confidence bounds are shown in red. As can be seen from the image, the initial uncertainty is very large and drops fairly quickly once motion of the robot creates some parallax between measurements. The convergence of the state is fairly rapid and the state estimate does stay within the bounds for the entire experiment.

Figure 7.7 shows the range to the goal for the experiment. The true range is shown in blue while the estimated range is shown in black and the 95% confidence bounds are shown in red. As expected, the true range value stays within the confidence region like the state $\rho_g$. The goal leaves the field of view the first time at about $t = 10$ s, then enters the field of view after driving around the obstacles at $t = 24$ s. At this point the vehicle has traveled close to 7 meters creating
Figure 7.4: Path of the robot during the paved test track experiment. The initial position of the robot is given by the black circle and the position of the goal is the green circle. The obstacles are shown with the red circles. The barriers are shown in black and the path of the robot is colored blue when the goal is in the field of view of the camera. The path is red when the goal is not visible in the camera field of view. The robot successfully navigates to the goal location while avoiding the obstacles and the barriers.

A new measurement with a significantly new viewpoint. When this measurement is used there is a significant drop in the error in $\rho$ and range. Figure 7.8 shows the range error.

Figure 7.9 shows the two angle states $\alpha$ and $\beta$ for the goal during the run. The angle $\alpha$ is shown by the black line with $\beta$ shown by the blue line. The negative side of the edge of the field of view ($-\alpha_{FOV}$) is shown by the red horizontal line. The goal is outside the field of view when $\alpha < -\alpha_{FOV}$. As explained in Chapter 5, the goal has to be a certain number of pixels inside the field of view before the goal can be reacquired (due to the template matching step). The goal is reacquired at $t = 67$ s which is just after $\alpha$ enters the field of view. Only slight changes in $\alpha$ and $\beta$ are noticed when the goal is reacquired. There is a more noticeable change in $\beta$ after reacquisition, but the estimates of both angles are close when the goal comes back into the field of view. This is due to the other landmarks that are being tracked while the goal is not visible. The other landmarks positions are mathematically coupled to the goal position, and the measurements
Figure 7.5: Zoomed-in view of path around the first two obstacles and the path as the vehicle approached and drove along the barriers.

Figure 7.6: Plot of the state $\rho_g$ for the goal for the experiment. The estimate $\rho_g$ is shown in black and the true value is shown in black with the 95% confidence intervals shown in red. The true value of $\rho_g$ was calculated using the GPS data.
Figure 7.7: Plot of the estimated range and the actual range. The estimated range is plotted with the 95% confidence intervals. The actual range is calculated from the logged GPS data.

Figure 7.8: Plot of the range error vs. time for the entire run. The 95% confidence intervals are also shown.
of these landmarks help reduce the error in the position estimate of the goal. The error bounds on the angle states are not shown because there is no truth angle to compare against.

The control input $u_k$ at each time step is shown in Figure 7.10 with the blue line and the measured angular rate of the vehicle from the IMU is shown by the black line. As seen in the plot, as soon as the robot starts to move, the initial command is high to make sure that $\alpha$ is near the minimum value from the cost function. There is some oscillation in achieving the desired angular rate which is due to the low-level Packbot control that takes place on the Packbot platform itself. The commanded angular rate does not exceed the constraint of $|u_k| \leq u_{\text{max}}$, but because of the error in the low-level control, the actual rate of the robot does get higher than the constraint. The angular rate feedback from the IMU is the same value that is used in the EKF as the inputs into the prediction step.

The linear velocity command and feedback are shown in Figure 7.11. The commanded value is shown in blue with the feedback shown in black. The feedback is calculated from the two encoder signals received from each track. This is the velocity $V$ that is used as an input into the EKF for the prediction step.
Figure 7.10: Plot of the desired and actual angular rate. The black line shows the measured angular rate from the IMU. The blue line is the commanded value at each time step from the MPC algorithm.

Figure 7.11: Commanded and actual linear velocity. The commanded value (shown in blue) is constant through the entire experiment at 0.5 m/s. The measured value (shown in black) is calculated from the encoder feedback from the left and right tracks.
An image sequence is shown in Figures 7.12 and 7.13. This shows different images from the robot camera at the times listed below each image. Each image shows the estimated positions of the goal and landmarks and also the tracked goal and landmark locations from the visual tracker. The estimated states are shown in the green circle outlines while the goal estimate is shown by the thicker green circle. Landmark estimates that fall outside of the smaller acceptable score for the template matcher are shown by the blue circles. These landmarks continue to be estimated but are not reacquired by the tracker until they are inside the offset boundary and eligible for template matching. The tracked position of the goal from the visual tracker is shown by the red circle. The positions of the other tracked landmarks from the visual tracker are shown by the yellow circles. Figure 7.12(a) shows the initial selection of the goal and the additional landmarks before any movement of the robot. Figure 7.12(b) shows the robot starting to maneuver around the first obstacle and at this point the goal is already out of the field of view of the camera. Figure 7.12(c) shows the robot continuing out and around the second obstacle. This shows how additional landmarks are added as old landmarks leave the field of view. At $t = 37\ s$ as shown in Figure 7.12(d) the robot has successfully gone around the first two obstacles and has reacquired the goal in the image. The robot then approaches the barrier and Figure 7.13(a) shows the view after the robot turned to avoid the barrier. The goal is out of the image again, but the other landmarks are continuing to be used to improve the estimate of the goal location even though it is not seen. Figure 7.13(b) shows the view after the robot turned the corner around the barrier and successfully reacquired the goal in the image. The last two images, Figure 7.13(c) and Figure 7.13(d) show the robot arriving at the goal without any more obstacles. It is noted that the size of the goal object changes in size significantly during the test. As seen in Figure 7.12(a), the goal is very small and completely encircled by the green circle estimate in the image. In Figure 7.13(d) the goal is much larger as would be expected. Because of this change in size, it is difficult for the visual tracker to keep tracking the same location on the object. There is a significant difference in the tracked goal location from Figure 7.13(b) to Figure 7.13(c) to Figure 7.13(d). This change in the tracked position on the goal object is part of the reason for the error in the inverse-depth state $\rho_g$ for the goal estimate.
7.2.2 Rough Terrain

For the off-road experiment the robot starts in a dirt field with a significantly steep hill between the robot and the goal and the robot must drive down the hill to get to the goal. The initial view from the robot is shown in Figure 7.14. The goal selected by the user is the blue barrel on the trailer behind the utility vehicle near the left side of the image. There are multiple obstacles in the path of the robot with one being near the beginning of the test (the white barrel in Figure 7.14) and then multiple obstacles close together near the end of the experiment. Once the robot drives down the hill then it arrives on a dirt road surface until it reaches the goal. This environment was specifically chosen to test the algorithms in terrain where the mapping would
be more difficult due to the motion of the vehicle. This also makes the landmark tracking more difficult due to the changes in accelerations of the vehicle. The parameters used for this experiment are shown in Table 7.2. It is noted that there are two different commanded velocities used for this experiment. This is because of the steep hill that the robot must drive down. The commanded velocity is reduced from 0.4 m/s to 0.2 m/s when a slope above a specific threshold is detected using the IMU acceleration values.

The path of the vehicle and the layout of the obstacles and barriers are shown in Figure 7.15. The path of the robot is drawn in blue when the goal is visible in the camera field of view and the path is red when the goal is not visible in the camera field of view. The start position of the vehicle
is shown by the black circle, the goal location is shown by the green circle and the obstacles are shown with the red circles. The obstacles shown in the plot were placed in their locations so the robot would have to maneuver around them.

Figure 7.14: Initial view from the robot camera for the off-road test environment. The goal selected by the user is the blue barrel on the trailer behind the utility vehicle near the left side of the image. The white barrel is an obstacle placed near the robot. Other obstacles are placed on the flat surface at the bottom of the hill before reaching the goal.

In this experiment there are three locations where the robot loses the goal out of the camera field of view. The first location is shown in Figure 7.16 and occurs when the robot encounters the first obstacle. As shown in the plot, the robot is able to quickly maneuver around the obstacle and reacquire the goal in the image. The next location where the robot loses the goal is due to a steep pitch in the terrain and the goal is above the top of the camera field of view. The third location is shown in Figure 7.17. Before the robot approaches the first of the four obstacles it is tracking the goal in the left side of the camera. When the first obstacle is detected it turns to the left and moves the goal to the right side of the camera image. Next, the second and third obstacles are detected and the goal is required to leave the image so that no collision occurs with the obstacles. Finally
Table 7.2: Parameters used for the rough terrain experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Starting Position</td>
<td>(139.9, −33.0, 3.8) m</td>
</tr>
<tr>
<td>Goal Position</td>
<td>(78.1, −18.6, 0.0) m</td>
</tr>
<tr>
<td>Image Measurement Noise ((\sigma_{I_x}, \sigma_{I_y}))</td>
<td>(4, 4) pixel</td>
</tr>
<tr>
<td>Process Noise Parameters (\alpha_1, \alpha_4)</td>
<td>50.0</td>
</tr>
<tr>
<td>Process Noise Parameters (\alpha_2, \alpha_3)</td>
<td>5.0</td>
</tr>
<tr>
<td>Process Noise Parameters (\alpha_p, \alpha_q)</td>
<td>50.0</td>
</tr>
<tr>
<td>Initial Depth Values (\rho_0)</td>
<td>0.1</td>
</tr>
<tr>
<td>Initial Variance ([\sigma_\alpha, \sigma_\beta, \sigma_{\rho_0}])</td>
<td>([0.05, 0.05, 0.1])</td>
</tr>
<tr>
<td>Maximum Number of States</td>
<td>51</td>
</tr>
<tr>
<td>MPC (dt)</td>
<td>0.2 s</td>
</tr>
<tr>
<td>Desired Linear Velocity Flat</td>
<td>0.4 m/s</td>
</tr>
<tr>
<td>Desired Linear Velocity Hill</td>
<td>0.2 m/s</td>
</tr>
<tr>
<td>Max Angular Rate (u_{max})</td>
<td>0.25 rad/s</td>
</tr>
<tr>
<td>(\alpha_0)</td>
<td>0.45 rad/s</td>
</tr>
<tr>
<td>Look Ahead Steps (N)</td>
<td>40</td>
</tr>
<tr>
<td>Inertia Cost Gain (R_I)</td>
<td>40.0</td>
</tr>
</tbody>
</table>

...there is enough space between the third and fourth obstacles so the robot can turn between them and reacquire the goal in the image.

Figure 7.18 shows the elevation profile of the run and also the pitch and roll angles of the robot while driving to the goal. As can be seen in the elevation profile, there is a steep slope that the robot drives down in route to the goal. The roll and pitch are calculated using the filtered acceleration values from the IMU. The pitch is a minimum angle while traveling down the steep slope and is a maximum driving up a small ridge after the hill.

The inverse-depth state \(\rho_g\) is shown in Figure 7.19. The state estimate is shown in black with the true inverse-depth value calculated from the GPS measurements shown in blue and the 95% confidence bounds shown in red. The state tracks extremely well until the robot arrives on the flat surface where the state starts to move towards the confidence bounds. This is also shown in the range plot in Figure 7.20. At about \(t = 135\) s the estimate starts to have a constant offset from the true value and does get right to the edge of the 95% confidence bound. The error is shown in Figure 7.21.
Figure 7.15: Path of the robot during the off-road experiment. The initial position of the robot is given by the black circle and the position of the goal is the green circle. The obstacles are shown with the red circles. The path of the robot is colored blue when the goal is in the field of view of the camera. The path is red when the goal is not visible in the camera field of view. The robot successfully navigates to the goal location while avoiding the obstacles and the barriers.

The angle estimates of the states $\alpha$ and $\beta$ are shown in Figure 7.22 with $\alpha$ shown in black and $\beta$ shown in blue. The horizontal field of view and vertical field of view angles are also shown. The states appear to be more noisy than those from the flat test but that is because of the rough terrain on which the robot is driving. It causes the vehicle to roll and pitch which changes these states rapidly since they are in the body frame and not the inertial frame. This plot also shows the three different times where the goal leaves the field of view which is when $|\alpha| > \alpha_{FOV}$ or $|\beta| > \beta_{FOV}$.

The commanded angular rate from the MPC algorithm is shown in Figure 7.23. The actual feedback is measured from the IMU and is shown in black. The angular rate measurement feedback is more noisy during the off-road experiment due to the accelerations of the vehicle while driving. This value is used as the input $r$ into the EKF for the prediction step. Because the vehicle is driving off-road, the angular rates $p$ and $q$ are also shown for this experiment. Figure 7.24a shows the
angular rate about the body \( \mathbf{i}^b \) axis \( p \) and Figure 7.24b shows the angular rate \( q \) about the body \( \mathbf{j}^b \) axis.

The commanded and actual linear velocities of the robot are shown in Figure 7.25. The feedback velocity is calculated from the two encoders. The commanded velocity drops to 0.2 m/s when the robot is on a sustained incline of more than 15 deg.

The path of the goal in the camera image is shown in Figure 7.26. The area in the plot is the size of the camera image. The black outline shows the area where the goal can be successfully tracked. This is due to the size of the template used in the template matching step in the image processing as explained in Chapter 5. The camera measurements are moving significantly in the image due to the motion of the vehicle on the rough terrain.

Figures 7.27 and 7.28 show selected images from this experiment. As in Figure 7.12, this shows different images from the robot camera at the times listed below each image. The estimated states are shown in the green circle outlines. The goal estimate is shown by the thicker green circle. Landmark estimates that fall outside of the smaller acceptable score for the template matcher are shown by the blue circles. These landmarks continue to be estimated but are not reacquired by the tracker until they are inside the offset boundary and eligible for template matching. The tracked
Figure 7.17: Zoomed-in view of the robot path around the second set of obstacles on the hill test.

Figure 7.18: Plots of the elevation profile of the experiment in Figure (a) and the roll and pitch during the run in Figure (b). The roll and pitch angles are calculated using filtered accelerations from the IMU.

position of the goal from the visual tracker is shown by the red circle. The positions of the other tracked landmarks from the visual tracker are shown by the yellow circles. Figure 7.27(a) shows the initial selection of the goal and the additional landmarks before any movement of the robot.
Figure 7.19: Plot of the state $\rho$ for the goal for the experiment. The 95% confidence intervals are also shown.

Figure 7.27(b) shows the view after the robot has turned to go around the first obstacle. At this point the goal is already out of the field of view of the camera. Figure 7.27(c) shows the view after the robot has driven around the obstacle and reacquired the goal in the image. At this point the robot is about to start down the steep hill towards the goal. In Figure 7.27(d) the robot is at the bottom of the hill, but it is pitched down enough that the goal is above the vertical field of view of the robot. Figure 7.28(a) shows the robot after leveling off from the hill and the goal is again being tracked by the visual tracker with the various obstacles being seen ahead. Figure 7.28(b) shows the robot maneuvering around these obstacles and at this point, the robot is about to turn between the two cones to minimize the time that the goal has to leave the field of view. Figure 7.28(c) and Figure 7.28(d) shows the robot arriving at the goal without anymore obstacles between the robot and the goal.

7.3 Summary

This chapter presented the experimental setup used to test the designed algorithms in this dissertation. The software architecture was described and each ROS node was explained. An
Figure 7.20: Plot of the estimated range and the actual range. The estimated range is plotted with the 95% confidence intervals. The actual range is calculated from the logged GPS data.

iRobot Packbot platform was used to test the algorithms in two outdoor scenarios. A test was performed on flat terrain where the vehicle was required to lose the goal because of multiple obstacles. The vehicle properly reacquired the goal in the image and the state estimates were presented. The second test was in rough terrain where the robot had significant pitch and roll. In this test the goal left the camera field of view due to driving around obstacles and also due to large pitch driving on the rough-terrain. In each experiment, the robot successfully arrived at the goal location.
Figure 7.21: Plot of the range error vs. time for the entire run. The 95% confidence intervals are also shown.

Figure 7.22: Plot of the angles $\alpha$ and $\beta$ for the goal. The horizontal and vertical field of view angle are also shown in red and green respectively.
Figure 7.23: Plot of the desired and actual angular rates. The black line shows the measured angular rate from the IMU. The blue line is the commanded value at each time step from the MPC algorithm.

Figure 7.24: Plots of the angular rates $p$ and $q$. Figure (a) shows the plot of $p$ and figure (b) shows the plot of $q$. 
Figure 7.25: Linear velocity of the robot. The commanded value is shown in blue and the actual is shown in black. The actual velocity is calculated from the encoder feedback. When the robot is driving down a hill with a sustained slope of more than 15 deg then the commanded velocity drops to 0.2 m/s.

Figure 7.26: Camera measurement locations of the goal in the image frame for the entire run. The black outline is the area where the goal can be successfully tracked. This area is defined by the width of the template in the template matching algorithm. During this experiment the goal leaves the field of view through the left side of the image, the right side of the image and out the top of the image due to maneuvering around obstacles and due to the pitch of the terrain on which the robot drives.
Figure 7.27: First image sequence from the robot camera while driving to the goal point on the rough-terrain experiment. The time for each image is shown.
Figure 7.28: Second image sequence from the robot camera while driving to the goal point on the rough-terrain experiment. The time for each image is shown.
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

This work describes the algorithms developed for ground robotic vehicles to drive to a user-selected goal point without a priori map information about the environment and without GPS. The robot calculates the position of the goal and additional landmarks in the robot-body frame using a monocular camera for measurements. A novel approach was used to improve the position estimate of the goal. The position of each of the landmarks is linked to the goal, therefore the goal is considered to be the anchor point of the states. Defining the states in this way was shown to improve the position estimate over existing robocentric formulations. A 3D state formulation was also developed including 3D motion of the vehicle. This allows the vehicle to roll and pitch and still track the positions of the landmarks.

The path of the robot is calculated with a model predictive control (MPC) approach. The cost function into the MPC uses an observability-based cost that is used to minimize the uncertainty of the position of the goal in the robot-body frame. The designed MPC algorithm also takes into account the field-of-view constraints of the monocular camera which keeps the robot driving a path where the goal will stay in the field of view whenever possible. The MPC algorithm also avoids obstacles while driving to the goal. Obstacles are used in the MPC formulation as hard constraints and are detected using a 2D planar LiDAR. Proofs were presented to show that the MPC controller with the designed cost function does bring the robot within some small distance of the goal and that it will avoid obstacles under the stated conditions.

A visual tracker was also developed and presented that tracks the goal after the user selects the object and tracks the position of additional landmarks. This tracker is used with the EKF mapping to improve the tracking of the landmarks and to make reacquisition possible by using the estimates from the EKF to predict where the landmarks would be in the image.

Both simulation results and experimental results were presented to show that the method is successful in a real-world scenario. The experiments shown were done using an iRobot Packbot.
with two different environments. The first was a flat ground scenario with obstacles and the second was on more rugged terrain with obstacles also. The results from the flat terrain showed the ability of the robot to avoid small obstacles and still continue to track the position of the goal. The flat terrain experiment also showed the ability of the robot to lose the goal for a significant amount of time and still reacquire the goal when back in the field of view. The rugged-terrain environment experiment showed the ability of the algorithms to work in a 3D environment and still track and estimate the position of the goal. In both scenarios the robot lost sight of the goal, yet was able to reacquire the goal and arrive at the goal while avoiding multiple obstacles.

This work could also be extended to other types of vehicles such as aerial vehicles or underwater vehicles. These vehicles could be guided through unknown terrain by having the user select intermediate goal points from the vehicle camera. By successively choosing new goals the vehicle could inspect an area aided by the user.

8.1 Future Work

This section will focus on three main areas of the system and thoughts on future work within each area will be presented. The three areas are mapping, MPC and visual tracking.

8.1.1 Mapping

The EKF designed and presented in this work uses the robot-body frame as the frame of reference to calculate a local map, which contains the positions of the goal and landmarks. The motion of the vehicle (linear and angular velocities) are treated as inputs into the system for prediction purposes. The system could possibly be improved by including these velocities as states in the EKF framework for the purpose of improving the estimate of states that are outside the camera field of view. If these states were included in the EKF state vector then measurement updates from the camera could help improve the velocity states by adding additional information. This would be particularly helpful under rough terrain situations where there is rapid movement. As was presented in Chapter 7, the experiment in the rough terrain had very noisy input data from the IMU, therefore the camera measurements could help improve the state estimate, which would
then improve the prediction of landmark positions because the velocities used in the prediction would be from the current states and not just from the IMU.

Another area of future work dealing with mapping is the idea of changing the anchor landmark. In the presented work, the goal point is always the anchor point, therefore it is the landmark to which all of the other landmark positions are referenced. It was shown that using the goal as the anchor point helped improve the position estimate of the goal. When the goal is outside of the field of view, the anchor point is not being measured. The plot of consistency from Chapter 3 showed that the system remained consistent, however, the level of consistency did show a shift towards the upper bound of consistency. This shift might be mitigated by changing the anchor point to a landmark that is currently in the camera field of view. This is similar to the idea of relative navigation [90] where the anchor landmark would act as a relative key-frame. Once the goal was back in the field of view, the key-frame would then be shifted back to the goal. This shift of anchor landmarks would require the current covariances to be transformed properly, but could improve the overall performance of the filter.

8.1.2 MPC

The MPC algorithm developed in this work uses the goal states in the observability-based cost function, however, when the goal is outside the field of view of the camera, the robot will try and turn to get the goal back in the field of view of the camera as quick as possible. Using the other landmark states inside the observability-based cost function could help improve the state estimate of the goal and the other landmarks. This could help the robot choose the best path while trying to reacquire the goal in the field of view. The robot might take a different path around the obstacle if it were trying to improve the estimate of the goal and all of the landmarks while not seeing the goal.

One failure mode of the MPC algorithm is that it is a local algorithm that could get caught in a dead end or be diverted away from the goal without being able to get back. In the current method, there is no history or map created from the localization data. Creating a map would be more of a traditional SLAM approach where the states from the robot-body frame are converted into an inertial frame that could be used for high-level planning. A high-level planner could run in
the background to help the robot make decisions based on more information that would be gathered as the robot traveled toward the goal.

### 8.1.3 Visual Tracking

The visual tracking piece of this work uses feedback from the EKF to find the area where the template matcher should look for the goal or the landmark. This was necessary for the tracker to be able to reacquire landmarks and the goal. This could be improved by also using the current covariance of the goal or the landmark in the decision of where to search for the landmark. The covariance of a given landmark could be transformed into the image frame of the camera and the tracker could start searching at the mean value while intelligently searching the area covered by a given confidence bound. This would be helpful in situations where the landmarks look significantly different due to large changes in viewpoint which would require more searching to ensure that the same object was identified.
REFERENCES


autonomous systems.” In Decision and Control, 2007 46th IEEE Conference on, pp. 2980–2985. 106


189


