Implementation of a Leader-follower robot team using iRobot Create platforms

Presented by:

Francis Hamilton

University of Cape Town

Prepared for:

Robyn A. Verrinder

Dept. of Electrical and Electronics Engineering

University of Cape Town

Cape Town, October 2015

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature: .....
Francis Hamilton

Cape Town
October 14, 2015

Abstract

Leader-follower formation control is one of the most versatile and scalable formation control architectures. This study iterates through the necessary steps to achieve leader-follower formation control on a pair iRobotCreate robotic platforms through the use of visual feedback provided by a Microsoft Kinect. Image identification and measuring algorithms are developed to achieve the visual feedback requirements for the selected formation controller. The entire system is accurately modelled including the iRobot's motion models and the sensor noise models. An EKF state estimator is designed and shown to produce accurate states from noisy measurements on the implemented system. A Lyapunov proven error based controller is selected from literature and shown to drive the relative errors to zero. The entire system is simulated and shown to accurately keep the formation amidst sensor noise. The implementation results show the validity of the controller and state estimators with the exception of a final error in the leader's state in a certain formation scenario.

Acknowledgements

I would like to thank my supervisor Robyn Verrinder for not only providing me with guidance and advice through out my thesis but for always being there to help guide me through my final year.
To my friends and room mates for always being there to provide me with entertainment and support through out my university degree. I would like to personally thank Jonathan Schoeman for lending his ever valuable Kinect for educational purposes.
Finally to my family for their unwavering support and encouragement through out my life.

Contents

Declaration i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures viii

List of Tables xiii

List of Symbols xiv

1 Introduction 1

1.1 Background of this study 1

1.2 Objectives of this study 2

1.2.1 Problems to be investigated 2

1.2.2 Significance of this study 3

1.3 Scope and Limitations 3

1.4 Plan of development 4

2 Literature Review 6

2.1 Introduction 6

2.2 Robotic Vehicles 7

2.3 Formation Control Motivation 9

2.4 Formation Control Architectures 10

2.4.1 Behavioural mention artificial potentials 10

2.4.2 Virtual Structure 11

2.5 Leader-follower 13

2.5.1 Control Schemes 13

2.5.2 State estimation and Sensors 16
2.6 Literature Conclusion 17

3 System Overview and Design **19**
3.1 System Overview 19
3.2 Robot Operating Software 21
3.3 System Design 23
3.3.1 Hardware/Mechanical 23
3.3.2 Software 27

4 Image Processing Algorithm Design **32**
4.1 Preprocessing Investigation 33
4.2 Leader Measurement 35
4.2.1 Landmark Measurement 36
4.3 Final Algorithm 38
4.4 Optimisation 38

5 Control Design **40**
5.1 System Modelling 40
5.1.1 iRobotCreate's Dynamics 40
5.1.2 Leader-follower Formation Dynamics 41
5.2 Controller Selection 41
5.3 Localisation 43
5.3.1 Follower pose state 44
5.3.2 Leader-follower relative state 45

6 Final Design Overview and Experimental Methodology **47**
6.1 Final Design Overview 47
6.2 Experimental Methodology 47
6.2.1 Simulations 49
6.2.2 Implementation 50

7 Simulation Experiments **52**
7.1 Controller 52
7.2 State Estimators 53
7.2.1 Follower Pose Estimators 53
7.2.2 Leader Pose Estimator 53
7.3 Final Control System 53

8 Implementation Results **54**
8.1 Zero desired polar angle with stationary leader 54
8.1.1 In line start 54
8.1.2 Angle offset start 55
8.2 Zero desired polar angle with moving leader 55
8.2.1 In line start 55
8.2.2 Angle offset start 55
8.3 Non Zero desired polar angle 58
8.3.1 $\phi = 10°$ with stationary leader 58
8.3.2 $\phi = 10°$ with moving leader 59

9 Discussion **61**
9.1 Image Processing Algorithm 61
9.2 State Estimators 62
9.2.1 Follower' Pose Estimate 62
9.2.2 Relative Coordinate State's Estimate 63
9.3 Formation Control 63
9.4 Comparison to literature results 64
9.5 Final System 64
9.5.1 Fulfilment of primary objectives 64
9.5.2 Fulfilment of secondary objectives 65

10 Conclusions **66**
10.1 Primary Objectives 66
10.2 Secondary Objectives 67

A Software Design **68**

B Control **69**
B.1 Motion Model Theory 69
B.2 Sensor Model Theory 72
B.3 Actuator and Sensor Noise Model 73
B.3.1 State Transition Measurement 73
B.3.2 Vision Measurement of State 77
B.4 Controller Proof 78
B.5 Kalman Filtering 80

C Simulation Results **86**

D Implementation Results **92**
D.1 Zero desired polar angle with stationary leader 92
D.1.1 In line start 92
D.1.2 Angle offset start 92
D.2 Zero desired polar angle with moving leader 95
D.2.1 In line start 95
D.2.2 Angle offset start 95
D.3 Desired polar angle = 10 95
D.3.1 Stationary leader 95
D.3.2 Moving leader 95

References **100**

List of Figures

2.1 Mars Science Laboratory Curiosity Rover [9] 8

2.2 H.E.R.A.L.D system [11] 8

2.3 Dive cycle of a glider[14] 9

2.4 Wave glider[15] 9

2.5 Workings of the Virtual structure 11

3.1 Design One Setup 24

3.2 Issue One 24

3.3 Issue Two 24

3.4 Improved and Final Design 25

3.5 Landmark Design 26

3.6 Kinect Splitter 27

3.7 Final follower setup 28

3.8 RGB calibration 29

3.9 IR depth calibration 29

3.10 RGB publishing rate 31

3.11 Depth publishing rate 31

4.1 Canny Edge Detection 33

4.2 Movement of sphere 33

4.3 HSV Thresholding with Morphological Operations 34
4.4 Movement of sphere 34
4.5 Analytical explanation of xmid and xright to calculate phi and gamma 37
4.6 Landmark Measurement 37
4.7 Improved Design 39

5.1 Relative Coordinates 42

6.1 Final Design Algorithm 48

7.1 Linear leader velocity 52
7.2 Circular leader trajectory 52
7.3 Both state estimators for leader follower control(low noise) 53
7.4 Both state estimators for leader follower control(high noise) 53

8.1 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 56
8.2 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 56
8.3 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 57
8.4 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 57
8.5 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 60
8.6 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 60

A.1 Cosine Rule 68

B.1 Instantaneous Center of Curvature [67] 70
B.2 Wheel Encoder [67] 72
B.3 Absolute Wheel Encoder [67] 72
B.4 Sensor Model 73
B.5 Start position of the step tests 75

B.6 Measuring the data 75
B.7 Markers after step tests 75
B.8 Error for change in position state 76
B.9 Error for change in angular state 76
B.10 Wheel Encoder [67] 82
B.11 Absolute Wheel Encoder [67] 82

C.1 Controller's Simulink Model 87
C.2 Linear Leader Velocity:Errors 88
C.3 Linear Leader Velocity:Follower velocities 88
C.4 Circular Leader Trajectory:Errors 88
C.5 Circular Leader Trajectory:Follower velocities 88
C.6 EKF high noise 89
C.7 EKF medium noise 89
C.8 EKF low noise 89
C.9 UKF high noise 89
C.10 UKF medium noise 89
C.11 UKF low noise 89
C.12 EKF high noise 90
C.13 EKF medium noise 90
C.14 EKF low noise 90
C.15 Error in relative coordinate ρ 91
C.16 Error in relative coordinate ϕ 91
C.17 Error in relative coordinate γ 91

D.1 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 93
D.2 Difference in Odom and Est Pose(Blue=x,Red=y) 93

D.3 Difference in Odom and Est Pose(Black=theta) 93
D.4 Relative error from desired(Black=ρ) 93
D.5 Relative error from desired(Green=ϕ,Blue=γ) 93
D.6 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 94
D.7 Difference in Odom and Est Pose(Blue=x,Red=y) 94
D.8 Difference in Odom and Est Pose(Black=theta) 94
D.9 Relative error from desired(Black=ρ) 94
D.10 Relative error from desired(Green=ϕ,Blue=γ) 94
D.11 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 96
D.12 Difference in Odom and Est Pose(Blue=x,Red=y) 96
D.13 Difference in Odom and Est Pose(Black=theta) 96
D.14 Relative error from desired(Black=ρ) 96
D.15 Relative error from desired(Green=ϕ,Blue=γ) 96
D.16 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 97
D.17 Difference in Odom and Est Pose(Blue=x,Red=y) 97
D.18 Difference in Odom and Est Pose(Black=theta) 97
D.19 Relative error from desired(Black=ρ) 97
D.20 Relative error from desired(Green=ϕ,Blue=γ) 97
D.21 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 98
D.22 Difference in Odom and Est Pose(Blue=x,Red=y) 98
D.23 Difference in Odom and Est Pose(Black=theta) 98
D.24 Relative error from desired(Black=ρ) 98
D.25 Relative error from desired(Green=ϕ,Blue=γ) 98
D.26 Topview (Red=leader,Blue=Odom Pose,Green=Est Pose) 99
D.27 Difference in Odom and Est Pose(Blue=x,Red=y) 99

D.28 Difference in Odom and Est Pose(Black=theta) 99

D.29 Relative error from desired(Black=ρ) 99

D.30 Relative error from desired(Green=ϕ,Blue=γ) 99

List of Tables

4.1 Canny Edge Detection Circle Detection 34

4.2 HSV Thresholding with Morphological Operations Circle Detection . 34

8.1 In line start results 54

8.2 Angle offset start results 55

8.3 In line start results 58

8.4 Angle offset start results 58

8.5 Zero Angle offset start results 59

8.6 Zero Angle offset start results 59

D.1 Gain Sets 92

List of Symbols

c	—	Speed of light
f_{ad}	—	Radar A/D sampling frequency
f_c	—	Radar centre transmit frequency
γ	—	Chirp rate of linear FM waveform
λ	—	Wavelength
τ	—	Pulse width

Chapter 1

Introduction

1.1 Background of this study

We as humans perform formation control on a daily basis and do not even realise it. When we walk along a crowded pavement and there is definitive bi-directional flow whereby each person goes with the flow rather than against it, as this would not be optimal for travelling to their destination. Each human on that street is actually sensing the other humans and changing his or her course based on what the others are doing so that the flow of walking traffic is not impeded. When viewing these types of daily occurrences the collective motion is so fluid it almost appears as if it is one body moving but it is actually the aggregate result of each human's individual decision.

This decentralised approach for individuals to perform a collective goal through their interactions is apparent in nature as well as the world of robotics. Moving in formation has many advantages over conventional centralised systems, for example, it can reduce the system cost, increase the robustness and efficiency of the system while providing redundancy, reconfiguration ability and structural flexibility. Formation control integrates many different aspects of robotics in order to achieve the desired functionality of the system. It not only requires control laws to keep the formation but different types of sensors, actuators, localisation methods etc. to support the controller which often complicates the design considerably and so it is still a very much active research topic.

Developments in the field of formation control have led to its use in search and

rescue situations [1], exploration and mapping of disaster zones such as Fukushima [2], reduced fuel consumption in small satellite clustering [3] and almost other area of robotics [4][5][6]. There are many different types of formation control architectures which govern how agents in the system interact with one another. There are strengths and weaknesses to each and are mainly dependant on the situation for them to be applied in.

The leader-follower architecture, which this study is based on, is most applicable to smalls teams of robots where no communication exists between the robots.In this formation scheme robots orientate themselves relative to one or more real leaders. The approach assumes the leader can globally position itself i.e. it forms the trajectory of itself and thus the formation. There are some drawbacks to this architecture but these will be explored in this study through the use of two iRobotCreate platforms with visual feedback.

1.2 Objectives of this study

1.2.1 Problems to be investigated

The objective of this study is to iterate through all the necessary steps to implement leader-follower formation control on a pair of iRobotCreate platforms with visual feedback from a Microsoft Kinect.
The primary objectives can be defined more explicitly as follows:

- Development of feature identification and measurement software algorithms
- Modelling of the system
- Selection of an appropriate controller
- Development of localisation algorithms
- Simulate the final designed system
- Implement final designed system in real time

The secondary objectives are as follows:

- Design of appropriate markers for visual tracking

- Optimisation of the feature identification and measurement software algorithms

This study will be mainly centered around the control aspect of leader-follower formation control so as make it adaptable to particular applications. For this reason the developed algorithms' processing times will not be a major design constraint but rather the efficacy of the different developed control methods being the basis for the design.

1.2.2 Significance of this study

The objectives mentioned above will allow the study to draw conclusions and thus make recommendations about the steps taken in order to implement the final designed system. The methods and algorithms developed will help guide researchers in any field where formation control is needed such as a pair of autonomous underwater vehicles. The visual identification would have to be improved and modified for applications where the control is not the center of the design.

1.3 Scope and Limitations

The scope of this project is develop all the necessary subsystems, outlined in the Objectives above, in order to achieve leader-follower formation control. The final system design scope and experimental testing will be constrained by the following limitations:

1. No provided Netbooks for the iRobotCreates
   There is only one Netbook for the two iRobotCreates but it was being used by a master's student and did not have a charger at the time. A charger was not able to be sourced in time as it was a special type of charger plug.

2. Author's laptop weight
   The author's laptop, ASUS G74SC, is an old gaming laptop which weighs around 4.4kg this prevents it from being on top of the follower iRobotCreate as it exceeds the maximum payload specifications and will cause the motion of the robot to be unreliable.

3. Kinect
   Due to the Mechatronics group's Microsoft Kinect being used for a different student's thesis a Kinect had to be sourced outside the university. Since it was much higher than the thesis budget one was borrowed from the author's friend. Thus it could not be modified at all so as to have it powered from the iRobotCreate's battery and not the a wall socket.

4. No aerial webcam
   A webcam to monitor the motion of the iRobotCreates from above in real time was unable to be sourced by the time of the thesis hand in.

Due to the limitations of the project the leader will not perform circular motion and will be stationary for most of the experiments. The leader will still be able to move but only transverse linearly and will have to be carefully pulled by hand as the robot's dynamics will be very different when pulled by a human hand. The limitations also force the follower iRobotCreate to be cabled to the authors laptop which is not on the robot due to unavailability of the Netbooks and the author's laptop weight. Since the follower robot had to be cabled to the author's laptop the cabled connection to the Kinect is not an additional problem. A major constraint is not having an aerial webcam to monitor the robots in real time so as to have a ground truth to compare the results against. There are quite a few limitations imposed on the design and experimental testing but solutions will be produced in the report as to produce the best possible design with the above constraints.

1.4 Plan of development

The report starts out with a review of the current literature available for formation control, mainly focusing on leader-follower formation control. The review is aimed at outlining the motivation for formation control in any robotics application and broadly understanding the different types of formation control architectures before focusing on the methods and problems associated with the chosen leader-follower architecture.

Following this, the system overview is presented where the different hardware and types of software to be used in the design are outlined and explained. The mechanical and software design will be based around the system overview where it explains the necessary steps needed in order to implement the following algorithms in the

design sections.

Next, the image identification and measuring algorithms will be developed to achieve the visual feedback requirements for the controller design in the following section.

Since the control design is the main contribution of this thesis, its chapter will be the longest and so as to make the chapter flow better the design will be presented with referencing to the relevant theory in the appendix when necessary. In any control design certain design steps must always be followed so as to ensure the validity of the final controller. Firstly the dynamics and parameters of the system to be controlled must be determined through "step test" data, these tests will show how noisy the robots actuators and sensors are etc. Then before implementing any control law it must first be mathematically shown that the closed loop system is stable and performs the desired outcomes. So a controller will be chosen from the available literature, it will be mathematically analysed and shown to be stable and achieve formation of the system. Since any Leader Follower formation control law requires state estimation for the pose of the mobile robots. Two state estimators will be designed and shown mathematically to produce accurate states from noisy measurements and actuators but only one will be chosen for the final system. The entire system will be simulated in the computational program Matlab to further validate the control law and then finally it will be physically implemented. Data sets will be obtained through experiments, outlined in the experimental methodology chapter, and will be presented in the results chapter.

Finally, the results will be discussed and conclusions will be drawn based on the outcomes in the results compared against the objectives. Recommendations will be made for future work on this thesis's topic based on the conclusions drawn.

Chapter 2

Literature Review

2.1 Introduction

Multi-agent systems refer to a network of nodes which achieve a predetermined goal through their interactions with one another. Formation control is a part of this multi-agent system and is instrumental in these multiple agents keeping or switching their chosen formation goal topology. Formation control is most apparent in nature where large numbers of animals group together to form a functional formation to provide options which which would not be available individually such as to improve their chance of survival. Flocks and other synchronous group behaviours such as schools of fish are both intriguing and beautiful to watch. At first glance the motion of birds flocking appears to be one moving body as the collective motion is so fluid but it is actually the aggregate result of all the individual birds decisions based on its local perception of the world [7]. Much research has been developed in the area of formation control and thus there are multiple architectures available in the literature. One example is behaviour formation control which can be applied to the phenomenon of the flocking of birds. The central objective of the different architectures is to produce robust and scalable control laws that are relatively simple on an individual basis but enable agents at the group level to perform with greater functionality and intelligence.

This review aims to provide the reader firstly with an understanding of the current state of robotic vehicle applications and a motivation of why formation control improves these applications. This will be followed by an overview of the formation control architectures other than leader-follower available in current literature. Many

of the different control schemes and approaches are common to all the architectures so they will be grouped according to the volume of literature available for each. Finally chosen core studies relating to the specific leader follower architecture will be analysed and supplemented with related studies to give the reader an in depth understanding of the different control schemes, image processing and navigation algorithms needed to achieve formation control.

2.2 Robotic Vehicles

Robotic land vehicles are generally the most popular and are used in a range of applications, from cleaning households to the exploration and performing of tasks which would be impossible or too dangerous for humans. The trend is for these robotic vehicles to be completely autonomous as compared to their remotely operated counterparts (ROVs) as they do not require constant monitoring and control from a human operator. Robotic vehicles are not only utilised on land but also in the air, ocean and even in space. Autonomous or unmanned aerial vehicles (UAVs) have progressed slightly more in the field of robotics compared to that of unmanned underwater vehicles (UUVs) due to the nature of the harsh environment they have to navigate in.

Autonomous or Unmanned ground vehicles (UGVs) have been used to locate landmines in countries such as Angola where explosive remnants of war are a daily hazard [8]. They have also been used to explore and inspect areas deemed fatal for humans such as the Fukushima nuclear plant disaster in 2011 [2]. However, not all applications are associated with tragedies, as UGVs have been in the frontier of exploration such as the Mars rover Curiosity which has given new insights into the Martian planet through the navigation of its hazardous terrain [9].

UAVs are most common in applications where areas are inaccessible by UGVs. The terrain of these area are often full of unavoidable ground obstacles such as fallen debris in an underground mine where it would be very difficult for the search and rescue(SAR) ground robotic teams to navigate through [1]. SAR robotic teams are tending towards utilising both UGVs and UAVs so in order to adapt to a variety of situations for example using vision acquired through the UAV to improve the UGV's path planning [10]. An interesting and growing research area is the integration of both a UGV and UAV into one robotic platform, as shown in figure 2.2, for

Figure 2.1: Mars Science Laboratory Curiosity Rover [9]

the inspection of unsafe environments, as this combination would allow the robot a much more thorough inspection and improved navigation of its surroundings [11].

Figure 2.2: H.E.R.A.L.D system [11]

The ocean is truly the frontier of our planet with less than 10 percent of it explored and mapped while the rest is just referred to as the deep sea. Studying the ocean currents and weather gives scientists a greater understanding of how our planet operates as after all it covers more than 70 percent of our planet. Up until fairly recently most underwater vehicles were remotely operated via an umbilical cable from the surface by a human operator [12]. However, this required a vessel on the surface of the water while the underwater ROV was beneath, making exploring the ocean very inefficient and expensive. Technological advancements such as using artificial bladders to control the buoyancy [4] of the robot allowed UUVs to be more predominate in ocean exploration and research. These gliders are able to autonomously navigate the ocean's depth, as shown in figure 2.3, collecting valuable data such as the effects on blooming rates of phytoplankton in the Southern Ocean due to climate change

[13]. Wave glides convert the swell of the ocean into forward thrust through the use of fins below the surface, as shown in figure 2.4, to transverse the ocean's surface. The Southern Ocean Carbon and Climate Observatory, in S.A, makes use of these types of water vehicles to determine the levels of iron, gas fluxes and other properties of the ocean's surface layer.

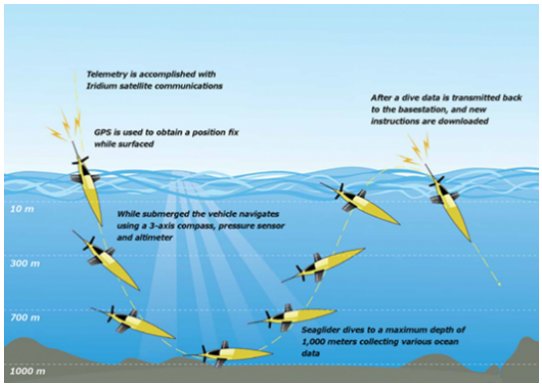

Figure 2.3: Dive cycle of a glider[14]

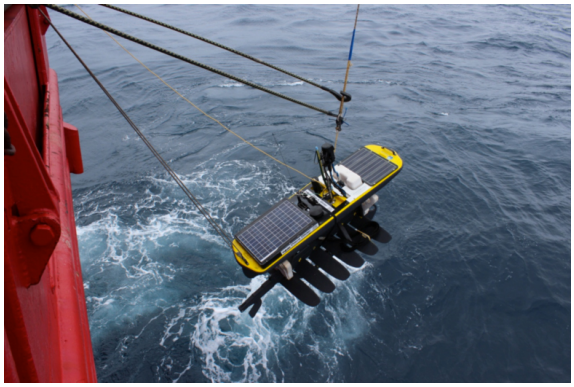

Figure 2.4: Wave glider[15]

2.3 Formation Control Motivation

The above examples of different types of robotic platforms that perform specifically designed tasks show the need, especially autonomous vehicles, to have and architecture governing how they interact with other another. Nature shows us that there is a definite advantage to performing tasks as a decentralised coordinated group rather than one powerful unit. A pivotal example is the reason why birds fly in a V shaped formation, this is to do with decreasing the drag force experienced by the group through alternating a leader to bear the majority of the work. This type of formation behaviour has been incorporated into optimal path planning for UAVs to decrease fuel consumption [5]. Robotic agents acting in this way improve the robustness of the entire system and enable the group to achieve tasks that would otherwise be impossible with just one agent. The advantages of incorporating formation control are apparent in any robotic system such as a group of reconnaissance robotic agents acting in formation allows them to more accurately cover a far greater search area. Another example of formation control is how a group of decentralised micro quadcopters, with relatively simple hardware, to outperform a larger more powerful quadcopter in terms of attitude control [6]. There are numerous areas of how

achieving formations can improve the functionality and efficiency of multiple mobile robotic agents, the reader is referred to an older study [16] for further examples.

2.4 Formation Control Architectures

2.4.1 Behavioural mention artificial potentials

Behaviour-based formation as mentioned earlier is most imminent in nature, studies on flocking and schooling show that these group behaviours emerge from a combination of individuals needing to stay together yet remain a certain distance from others. Behaviour-based control methods provide the robots with actions in reaction to sensor data and so researchers have used these behaviours to better understand grouping in both simulated agents and robots [17]. One of the more important early works in the field was done by Craig Reynolds, who managed to model and then simulate the flocking of birds at a very impressive level [7]. His work was improved recently by others[18] as his simulation did not fully take into account the motion of the animals but rather treated them like particles. In behavioural control, several desired behaviours are encoded into each agent, and the final control laws are derived from a weighting of these behaviours. Reynolds and others weight the following behaviours in order of preference:

- Attraction to distant neighbours up to a maximum distance
- Repulsion from neighbours too close
- Alignment or velocity matching with neighbours

Researchers have encoded additional behaviours to enable a higher level of functionality such as moving-to-a-target, initial formation-keeping, and avoiding-obstacles while maintaining formation [19][20]. There is little research into large scale behavioural formation control except for some simulated results regarding the optimisation of the encoded behaviours. One such study used convex optimization to simulate one hundred robots forming different formations [21].

Behaviour based formations are often controlled through machine learning type control strategies such as fuzzy logic [22][23] or neural-network [24][25] where robot reactions are decided through reasoning of various types of qualitative behaviours

using fuzzy logic or through prediction using a neural network which is trained by a database representing quantitatively the behaviours. Machine learning based controllers are also applicable in other formation control schemes but the general drawback of these controllers is the suffering from the local minimum problem due to its totally reactive nature. I.e. getting trapped in front of an obstacle or wandering indefinitely. There are algorithms that help solve this issue such as the random walk (RW), multi-potential field (MPF) or wall following (WF) method [22]. In essence the idea is to either generate random planar movements (RW) or to have a higher level control action based on the distance to the goal (MPF/WF).

2.4.2 Virtual Structure

A virtual structure is a collection of particles (Robots) which maintain a semi-rigid geometric relationship to each other and to a frame of reference [26]. The huge disadvantage of this method is the high bandwidth needed for communication between the robots. The reason being is the robots will update their position based on the virtual structure matrix they received. The concept is best illustrated in figure 2.5

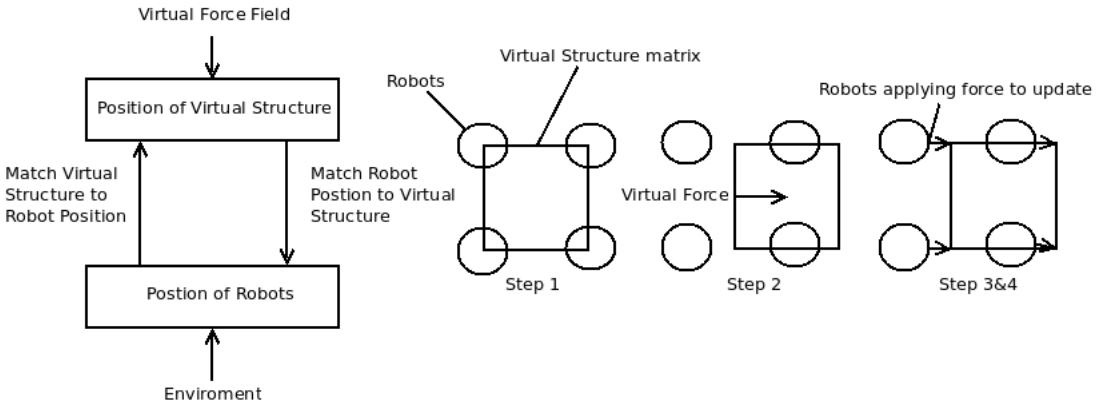

Figure 2.5: Workings of the Virtual structure

The virtual structure representing the formation is controlled by the operator via the virtual field force. This movement of the virtual structure will be communicated to all the robots and their positions will be updated according to the virtual structure matrix. This method of formation control can only be utilized in applications such as space where high bandwidth communications between robots is available [27]. It would not work in applications such as sea exploration where communication is not always available. However, the advantage to this method is that it is powerful un-

der the right conditions and is relatively straight forward to implement.The virtual structure matrix is often formed from either a specific shape where each robot is a point in the shape [28] or through the use of a virtual leader [29] to set-up up the desired matrix.

In [29] they use the iterative learning control (ILC) approach to tackle the formation control problem of a non-linear multi-agent system with a switching interaction topology. They first formed a consensus based approach for generating the virtual structure matrix from the relative positions of the robots to the virtual leader. A consensus is reached when all agents in the system agree upon some predefined quantities of interest, by this definition all formation control architectures are consensus based. In order to use ILC their defined consensus problem must be laid out over finite time intervals to which they used a distributed D-type iterative learning scheme. The reader is referred to chapter 9 of [30] for a good explanation of ILC but the general idea is that the next input signal is computed based on the current error so that the following error is reduced. They showed, in simulations only, that over a certain number of finite time iterations the formation is kept and can be switched. Study [29] is a good example of how ILC can be incorporated into formation control schemes.

Study [28] makes use of a partial behavioural based approach but for the virtual structure method through using a two layer control scheme. The first layer is responsible for the guidance of the UAVs in the ellipse shaped virtual structure. It ensures the virtual structure reaches its destination and bypasses encountered obstacles by either changing the virtual structure shape or the heading. The second layer ensures each UAV stays within the elliptical shape in space and maintains a set minimum separation distance from the other UAVs. Each control layer uses model predictive control (MPC) where by both layers' responsibilities are weighted in a cost function to determine all control actions taken. The authors show how the shape of the virtual structure does not have to be elliptical but it is the easiest to manipulate. A promising improvement of their work would be to allow the virtual structure to be separated in two to improve bypassing obstacles.

The above studies and most others use what is defined as rigid-based formation scheme where by each agent's position is described by a recti-linear relative separation to the virtual leader in the matrix, this recti-linear approach is by far the most common and easiest to understand. The contrast to this rigid-based forma-

tion is called a flexible formation scheme whereby the agent's position is defined by a curvilinear relative separation to the virtual leader. A flexible formation is described through the curvature, hence curvilinear, of the group's trajectory which allows the distances between the robots to differ slightly during turning thus providing a much better motion manoeuvrability [31]. An applicable study [32], shows how non-holonomic wheeled robots can be instead described through a flexible virtual structure formation and achieve better results in terms of manoeuvrability whilst carrying out their navigation task.

2.5 Leader-follower

In this formation scheme robots orientate themselves relative to one or more real leaders. This type of formation scheme is most applicable to groups of up to four or five agents as referencing to one leader with a lot of followers to keep in mind is quite cumbersome. The approach assumes the leader can globally position itself i.e. it forms the trajectory of itself and thus the formation. Leader-follower has the advantage in that it is easy to understand and the formation is maintained even if the leader is perturbed by a disturbance. The drawback is if a follower is disturbed the formation will be lost and there is also a single point of failure, the leader [16]. In the leader-follower formation literature it is generally assumed that the followers' actuators and sensors have no noise so as to allow the control design to be easier to understand and to try mainly focus on the stability of the system i.e. formation errors reaching zero. However, no practical system would obey these assumptions and therefore state estimators will have to be incorporated into the control design so as to have a far better state estimate in the presence of a noisy system. This section will first review the current control schemes used to achieve leader-follower formations, with ground vehicles, following that how some studies have incorporated different types of state estimators and sensors for better practical implementations.

2.5.1 Control Schemes

A similar method to the virtual structure consensus approach for leader-follower is through the use of artificial potentials [27][33]. The idea behind it is the distance between the robot and the others forms the potential function. The sum of

all potential functions affecting the individual robot must be driven to a minimum potential through controlling the robots position to some prescribed inter-spacing formation. In [27] a Lyapunov function, to prove stability and robustness, as the sum of the vehicle's kinetic and artificial potential energy. Study [33] follows a similar approach except they incorporate obstacles as another potential function for the leader to adjust the group's trajectory. The above two artificial potential studies and most others assume the robots are holonomic and only show results through simulations. Practical studies generally only use artificial potentials for obstacle avoidance as obstacles are stationary with no orientation [34].

As mentioned earlier the most common type of land vehicle is the differential drive vehicle whereby each robot is controlled through a velocity and angular velocity command. Since each robot's state is described by its two axes position and orientation the robot is considered non-holonomic, meaning the total controllable degrees of freedom are less than the total degrees of freedom. The reader is referred to the *System Modelling* section in the appendix for an in depth derivation of the system dynamics before reading on. The system is non-linear in both the control of the follower robots and the leader-follower relative dynamics, so a non-linear control law must be designed.

The easiest way to circumvent this non-linearity would be to linearise the system around a desired operating point [35][36][37]. In [35] they first show how a feedforward controller can be designed where, when the reference trajectory is known, the follower robot will keep to this trajectory. Nonholonomic systems usually have feedforward control where system inputs are calculated from the known trajectory. However, the use of the open-loop control only (just feedforward) is practically useless because it is not robust to errors in initial system states and other disturbances during operation. Closed loop is therefore added for practical use. Using feedback an error kinematic model is designed then a zero error and input operating point is chosen to linearise the system. Following that normal linear methods are shown to demonstrate adequate results for following a reference trajectory(leader). In [37], on which [35] is based, they take it one step further to explicitly show stability through the use of an indirect Lyapunov function. They show that the system is only locally stable (due to the linearisation) and it is not asymptotically stable.

The first types of control techniques used to achieve formations with non-holonomic robots was to linearise the system using the standard input-output state linearisation

method and then use common linear control tools to achieve the desired characteristics [38][39][40][41]. In [40] they first show a sufficient condition for the observability of the leader through the use of range/bearing only measurements via an omnidirectional camera. They design a controller through input-output state linearisation and show how it can be extended to more than one follower however no obstacle avoidance is incorporated. In [38] they deal with obstacles in quite a clever way, they treat a detected obstacle, by the leader, as a follower robot and apply the same type of controller used for the leader-follower formation control to navigate around the obstacle. However they do not use a decentralised approach as follower robots' commands are sent via a wireless network and their poses are measured through a ceiling camera. All studies show how the relative polar coordinates tend to zero as time elapses but they can never be zero only bounded due to the non-holonomic nature of the problem [42].

Often the leader's future trajectory is known with some probability based on the leader's past motion i.e. if it has been driving straight for five iterations it will most likely carry on going straight. This type of future knowledge for the tracking of this trajectory allows control techniques such as the receding horizon (RH) method to become very dominant in the literature [34][42][43]. The main drawback of this method is the high computational requirements but the relatively low complexity makes this technique very viable for researchers. Methods such as sliding mode control [44] or neural [45] often do not have as much literature available due to the high complexity of these methods. Study [34] uses input-output state feedback linearisation and incorporates standard MPC with this now linear model to achieve formations. They used an artificial potential function for detected obstacles and incorporated it into the MPC cost function to derive the final control commands. They showed their formation maintaining in simulations performed adequately but the switching of formations was very poor due to the controller not being globally stable from the linearisation. In study [42] they deal with the very high computational burden of the RH method by combining it through the error posture with a very well known stable tracking control method [46]. This combination decreases the amount of predictive steps in the RH tracking method and thus the overall computation time is decreased. They also develop a waypoint navigation method for the followers which takes into account their velocity constraints when maintaining or switching formations.

The most common and least complex to understand approach available in litera-

ture is deriving a relative error based tracking model and from there determining control commands which will drive the errors to tend towards zero though the use of direct Lyapunov theory [47][48][49]. Study [48] first derived the error based model based on desired relative polar coordinates, then defined control laws based on the backstepping control approach. For an excellent and well cited study on the backstepping control approach the reader is referred to the an older study [50]. Study [48] then validated these control laws to show stability through the use of a direct Lyapunov function. Study [47] first constructed a quadratic Lyapunov candidate function and from there derived conditions, though the use of gains, for what the input commands should be in order for stability and errors to converge towards zero. The results showed only the bound of the relative distance error could be decreased arbitrarily via the tunable gains whereas the rest are ultimately bounded. The tunable gains showed how the controller can prioritize the convergence of the relative polar angle over the relative orientation or vice versa.

2.5.2 State estimation and Sensors

The investigation into sensors which can extract information from the immediate surroundings of a dynamic object such as a robot has been a topic of research for many years and an area of extreme interest in the robotics community. There are many types of sensors used not only in formation control but also for the 3D mapping of an environment. The most common ones found in formation control are ones which can return range/bearing measurements, namely: sonar [51], laser [52], mono/stereo cameras [53] and more recently RGB-D sensors [54]. Not many formation control studies focus on the sensors needed for implementation but similar studies into SLAM show the efficacy of the different sensors for robotic navigation. The common practice in robotics is to use stereo cameras to retrieve range/bearing measurements of the desired object through the method of stereo triangulation [55]. RGB-D devices such as the Microsoft Kinect which use a similar technique to stereo triangulation just with infrared have become more popular for indoor navigation. There are many different ways to identify the leader in an image, the most common method in formation control studies is to use markers [47][35][38] however, most robotic navigation studies utilise feature detectors and descriptors [56] to identify known objects in an image. Any system will need software in order to link and control all the hardware. Since formation control is still a very actively researched

topic computational programs such as Matlab or Octave are used over programming languages such as C++ or Python due to the support available and the prototyping speed [57].

Any sensor employed will have noise in its reading and so in any practical system a state estimator must be utilised to more accurately determine the present state from multiple noisy measurements. In 1960,a significantly famous paper was published by R.E. Kalman [58] described an iterative solution to linear discrete filtering problems, one which was based on the properties of probability theory and conditional Gaussian variables, known as the Kalman Filter. The method described in this paper provided a means for estimating a new state estimate which is based on the priori state as well as a Kalman gain and in making this new state estimation, minimizes the mean of the squared error. The Kalman Filter assumes a linear model but since most systems are non-linear an adaptation of the original Kalman filter was developed, the Extended Kalman Filter (EKF). A Kalman Filter literature survey [59] showed the EKF is by far the most common in formation control studies as well as in SLAM studies. Another Kalman Filter known as the Unscented Kalman Filter (UKF) uses the unscented transform to pick a minimal set of sample points around the mean so that the filter can avoid poor performance when the state transition and observation models are highly nonlinear. Studies use the UKF when the robot's actuators and sensors are not purely Gaussian as the UKF is accurate up to second order moments in the probability distribution function. There are other state estimators such as adaptive versions of the different Kalman Filters and Information Filters [60] but they are far less common in formation control literature.

2.6 Literature Conclusion

Incorporating inter-agent governing architectures in any robotic system allows it to perform with a greater functionality and intelligence in almost any situation. The type of formation control architecture to use is however heavily dependent on the situation it is being applied in. It would be impossible to fully implement the virtual structure architecture with AUVs due to the extreme inter-communication bandwidth constraints in underwater navigation however for formation control of multiple UAVs it would be more applicable. Behavioural based is definitely the most interesting and promising formation control scheme but the complexity and unpredictability cause researchers to tend towards easier control schemes such as

Virtual Structures or Leader-follower. Virtual Structure is by far the most powerful formation control technique under the right conditions as it allows the switching of formations to be a lot more fluid and controlled. Leader-follower is the most versatile of them all as it is not as complicated to implement and the control laws developed are easily scalable for more followers up to a point [57]. As mentioned earlier the majority of the formation control literature mainly focuses on only the theoretical approach to designing control laws not practical implementations as it is such a broad and still a very active area of research. As a result subsystems such as state estimators are not incorporated into the studies' controller design. After reviewing the current literature the formation techniques which should be investigated first in the design aspect of this study is using an error based Lypunov proven controller with an EKF for state estimation.

Chapter 3

System Overview and Design

This chapter first explains the different hardware and software needed to integrate all the different sensors and actuators. It software side explores the open source library Robot Operating System (ROS) and how its works in relation to the different hardware needed. Through understanding the system overview, a mechanical and software design is presented to illustrate how to utilise the system to achieve the requirements.

3.1 System Overview

The iRobotCreate was developed from the ever popular iRobot Roomba as a learning robotic platform for students. What made it popular in the robotics community was a company known as Willow Garage designed a system known as the Turtlebot which incorporated the iRobotCreate as the base platform. The hardware used in this thesis will be similar to the Turtlebot except for the single axis gyro it has everything the Turtlebot uses and so code for the Turtlebot can be used. The hardware is as follows:

Mobile Robotic Base:

The iRobotCreate is an affordable mobile robot platform for educators, students and developers. The iRobotCreate has over 30 sensors which react to both internal and external events. It is built for indoor use only. The main sensors to be utilised on the mobile base are the left and right wheel encoders for localisation.

- iRobotCreate 4400
- 3000 mAh Ni-MH battery pack
- A DB25 connection which can provide unregulated battery power
- 1.2m serial cable with 0.8m USB extender

Microsoft Kinect

The Kinect camera is a low cost RGB-D sensor built specifically for the Xbox console. It delivers a RGB image and a depth image in parallel video rate. Since Microsoft has not released any official hardware specifications for the Kinect the information available has been reversed engineered. The OpenKinect community[61] explains the Kinect has one RGB camera and a range camera based on structured light. According to PrimeSense, the manufacturers of the Kinect and other RGB-D devices, the projected IR points are processed by a PS1080A micro processor to produce a depth image, I refer the reader to the PrimeSense website. The following Kinect specifications are from the PrimeSense PS1080A documentation [62] and unofficial sources [63]:

- Colour Camera = 640x480 @30FPS
- Depth(IR) Camera = 320x240 @30FPS
- Depth z resolution(@2m) = 1cm
- Spatial x/y resolution(@2m) = 3mm
- Field of View = 58 °H,45 °V
- Range = 0.8m - 3.5m
- Motor tilt range = +-27 °
- Power Supply = 12V, 2A
- 4m USB and power cable

Authors Laptop G74SX

- i7 @ 2.2Ghz
- weight =4.4 Kg

3.2 Robot Operating Software

This section is aimed at providing the reader with enough background knowledge of ROS to understand the following *Software Design* section. ROS provides the means to communicate with the iRobotCreate and receive data from the Kinect. It is not only applicable to using just the "Turtlebot" package but can be used for a range of other robotic applications. The description from their website best describes its use and is as follows:
"ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license."
The latest ROS version at the time of writing(ROS Indigo) and Ubuntu 14 LTS was used in this thesis. The following ROS description is a summerised version of all the ROS tutorials needed for this thesis's application so as to give future students a head start.
ROS is built up by stacks which in turn are built up by packages. The goal of packages is to create minimal collections of code for easy reuse where as the stacks allow users to create and share more complex systems. A stack delivers the functionality of a system such examples are:

- Turtlebot stack, which provides communication to the iRobotCreate Base
- ros_comm stack, which provides internal communication in ROS
- Openni_camera stack, which provides communication to the Kinect

The old method of using rosbuild to create user packages has be outdated by the easier to use catkin package whereby a catkin workspace can be set-up to handle multiple packages with the same dependencies. A package typically holds the functionality of a more delimited task than a stack and must contain at least one ROS node. A node typically solves one, or a few, separate tasks such as communicating between the main computer and the Netbooks on board robots. Nodes are started

up using the rosrun or roslaunch command. The first command starts one single node whereas the latter can start multiple nodes at the same time using the specified launch file. The launch file also has the property that values of dynamic parameters may be supplied to the nodes. This results in an accessible way of troubleshooting and trimming without re-compiling the code. ROS scripts which allow the user to create their own packages/stacks can be coded in either C++ or python.

A ROS message is a data structure that can be sent between nodes. Messages can hold any information and ROS comes with useful predefined messages. The two types of messages needed are the Sensor_msgs and Geometry_msgs for the Kinect and iRobotBase respectively. Users can define their own message type such as the result of a calculation.

ROS topics are often referred to as buses over which nodes exchange ROS messages. A topic implements an asynchronous communication pattern as a message is posted from one node to the topic and any nodes which listen to that particular topic will get a callback whereby the received message from the topic can be processed. Nodes in this fashion are referred to as publishers and subscribers to a certain topic. The ROS topic model is a very flexible communication paradigm, but its asynchronicity is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply much the same as a new ROS message type. So how does this network of topics and services be controlled while that is through the master. The master is the computer which runs the roscore command. This command starts the ROS network and enables the master to access all topics etc plus monitor all traffic within the system.

In any robotic system is very useful to be able to record data received from sensors to be processed later or analysed with different software. The ROS framework provides just this tool and it is known as a rosbag. A bag file can record all messages received from any number of topics specified when calling the rosbag record command. The bag file can therefore be used to reply a whole sequence of events such as RGB data from the Kinect to assess different image identification algorithms without needing the Kinect.

3.3 System Design

A design can be systematically produced through understanding how the hardware and software constraints will affect the algorithms needed for leader-follower formation control. The physical set-up of the leader and follower iRobotCreate platforms are explained in *Hardware/Mechanical Design*. The following section, *Software Design*, gives an in depth investigation into how ROS must be utilised to integrate all parts of the system.

3.3.1 Hardware/Mechanical

iRobotCreate Leader

The first order of business is to design a way to for the follower to identify the leader. Not only that but the follower must be able to measure the leader's relative pose through this identification. The most obvious and intuitive way would be to have some very unique markers on the leader which will make it stand out from the background. The following two designs show the progression to the final leader identification design.

Design One:

The most simple method to identify a marker would be through its colour and shape. Since a pure primary colour within a concentric circle would be extremely uncommon in the robots' background, it was chosen to be used as the marker. However this design must also be able to read the leader's relative pose and so through understanding the law of cosines one knows the angles of any triangle can be calculated if all sides are known, Appendix A. Armed with this knowledge using two red circles to both identify the leader and measure the relative pose are proposed. The circles are set 0.25m apart and are parallel with the horizon. This set-up is best shown below in figure 3.1:

There are a number of issues with the above designed set-up, namely:

- See figure 3.2. When the leader is rotates the 2D circles will not appear to be perfectly round but rather ellipses which would alter measurements.
- See figure 3.3. There will be bends in the paper, since it is paper after all, and so the printed circles will not be appear to be perfectly round which would also alter measurements

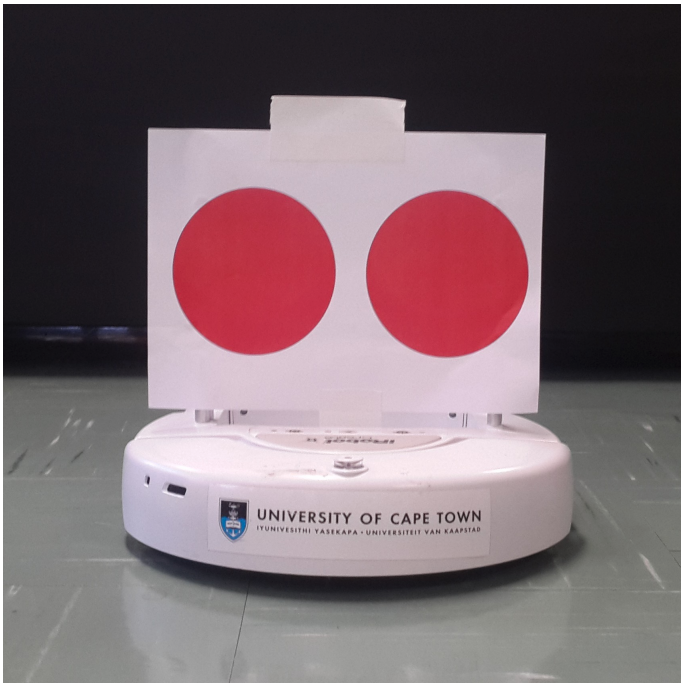

Figure 3.1: Design One Setup

Figure 3.2: Issue One

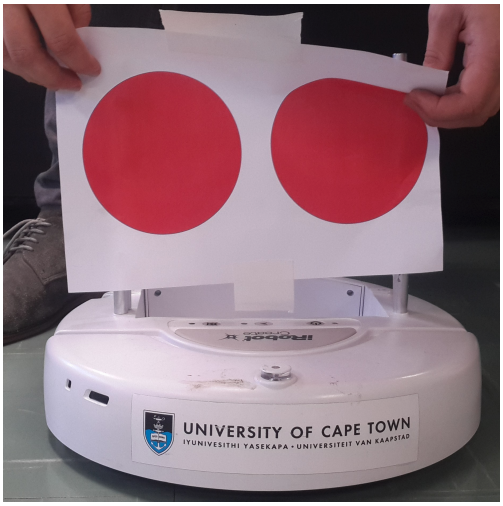

Figure 3.3: Issue Two

Final Design:

In order to deal with the distortions associated with the 2D circles on a piece of paper the following adjustment is proposed. The solution is obvious after the above design, rather use a 3D shape as after all the robots will be navigating a 3D world. The colour will remain the same as this was not the issue but the shape will be chosen to be a sphere. This improvement allows the measurements of the leader to not be distorted by its relative rotation to the follower. The validation is shown in figure 3.4.

Figure 3.4: Improved and Final Design

Landmarks

As will be seen in the *Control Design* section it is necessary for the robot to localise itself with its environment. In order to simplify the problem known landmarks will be used to achieve that. These known landmarks must be easily identified to acquire their relative range and bearing measurement to the follower robot. The leader identification design steps illuminate a way to achieve this. Each landmark will have a different colour to easily identify it and will be a sphere so the follower robot can acquire measurements regardless of its relative position to the landmark. The landmarks are shown in figure 3.5.

iRobotCreate Follower

This section outlines the steps take for the set-up up of the iRobotCreate base and the Kinect. Since no Netbook could be provided for the follower robot the author's

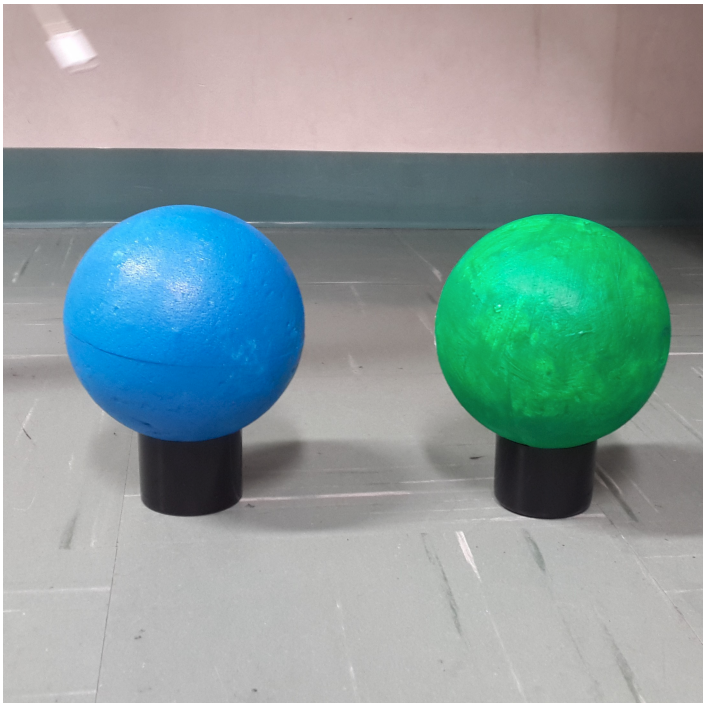

Figure 3.5: Landmark Design

laptop was used instead. The Kinect was not allowed to be dismantled so as to retrofit it to iRobotCreate base as it was borrowed from a source outside the university. The borrowed Kinect's power and USB cable were over 4m long so the Kinect did not have to be powered off the iRobotCreate's battery. It was deemed reasonable to not keep the laptop on top of the base but rather use a 2m long serial to usb cable so as not to interfere with robot's dynamics. The robot's dynamics would be effected due to the weight of the author's laptop, Asus G74SX, which mentioned in the *System Overview* weighs 4.4 kg.

Even though it was deemed acceptable to not have the laptop on board with the Kinect powered from the base due to the unavailability of the Netbooks, I will still explain how to set-up up a power supply for the Kinect for future research. The Kinect fuses the USB and the power source cable into one PoweredUSB cable at the splitter, see figure 3.6. So in order to provide power not from the wall outlet this splitter will have to be permanently damaged hence the reason for using a long cable instead. As mentioned earlier the iRobotCreate base can only output an unregulated supply from the battery which could potentially damage the Kinect. So instead of installing a separate regulated battery on the base a voltage regulator should be employed. The Kinect states it needs 2A of current to operate but this

is actually only if the tilt motor is needed, the cameras work fine with just 1A [61]. Any standard 12V regulator can be used such as common texas instraments LM series[64] but the Kinect is very sensitive to current fluctuations so the coupling capacitors etc. must be set-up properly. The voltage regulators V_{in} and GND should be connected to pin 10 and 14 on the DB25 socket respectively.

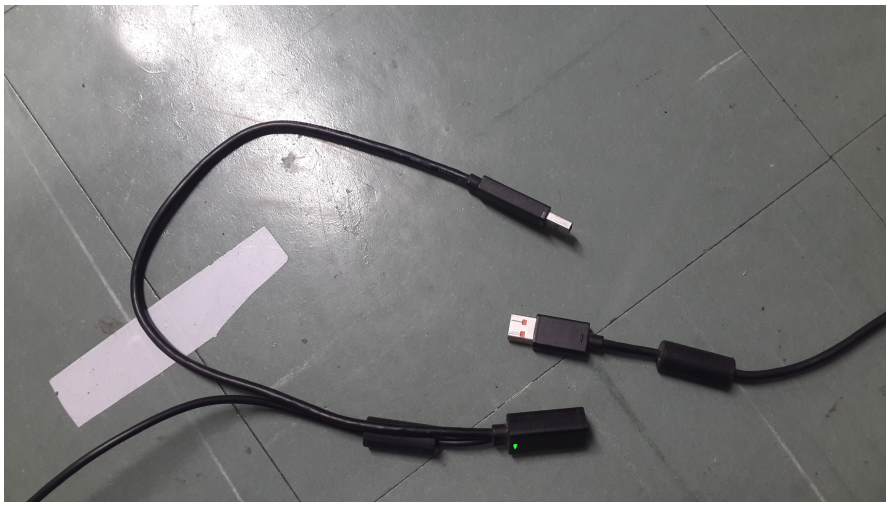

Figure 3.6: Kinect Splitter

As will be seen in the following section on the software design the center of the RGB camera must be aligned with the center of the iRobotCreate base. With this the final follower mechanical set-up is shown below in figure 3.7

3.3.2 Software

This section will outline the specific issues experienced during the coding of the *Image Processing design* and *Control design* algorithms in the following sections. It will not only show how these issues were dealt with but provide the reader with a better understanding of the different ROS packages used rather than trying to figure them out from just the source code.

As mentioned earlier since no official Kinect SDK has been released for linux a community known as the OpenNI project has developed a ROS stack to interface with the Kinect and other PrimeSense devices. The stack contains two Kinect pack-

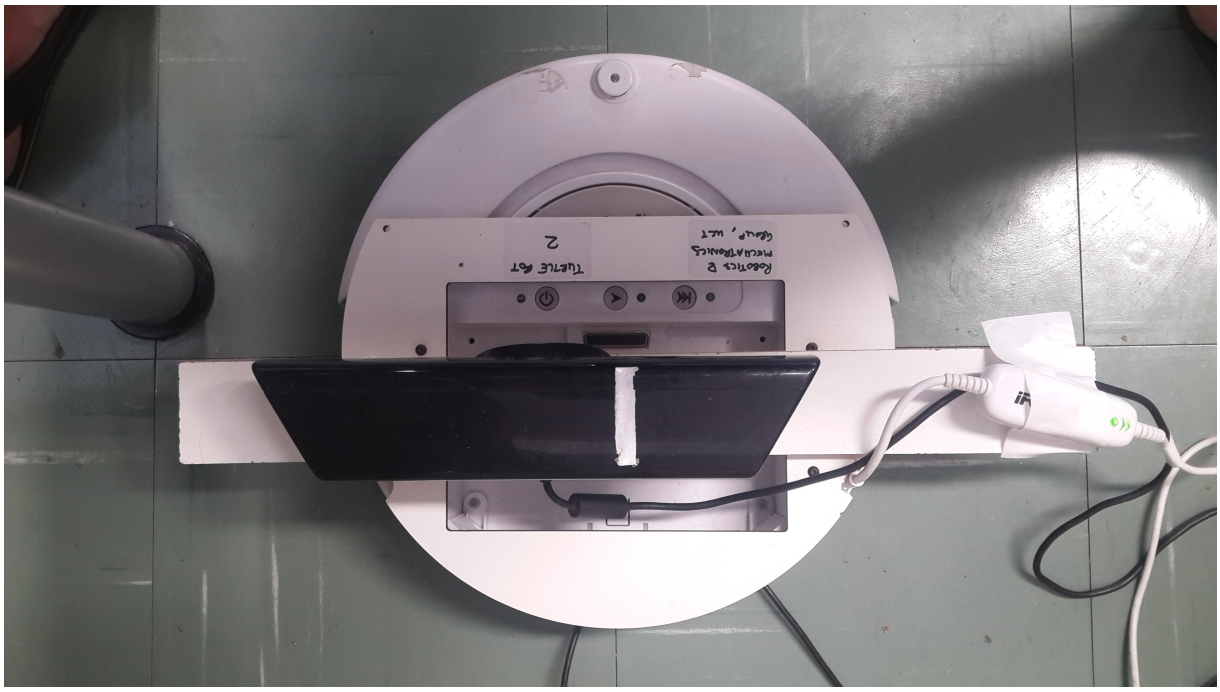

Figure 3.7: Final follower setup

ages one for just the camera and the other for skeleton tracking. The former is the one used for any application without the need to track human movement. The package is launched in ROS with the command roslaunch openni_camera openni.launch. As I mentioned earlier the launch command can run several packages each with different nodes as opposed to just the rosrun command which can only start one node. The nodes from the openni packages publish all topics to do with the Kinect, the ones of interest to us are the following:

- /camera/depth/image_raw
- /camera/rgb/image_color

However the reader will notice if the rostopic list command is run there are several topics which have registered before their name. This is in relation to the fact that all cameras even the Kinect need to be intrinsically and extrinsically calibrated as each device is generally different to the factory calibration settings.

Calibration

Intrinsic refers to the the camera's internal properties such as focal length, principal point etc where as extrinsic denotes the transformation from the 3D world to the 3D camera coordinates. The calibration process is often a long procedure so luckily

ROS has a packages to fully calibrate the Kinect. The camera_calibration package provides a text file containing the camera's intrinsic calibration values. It works by mapping a chequerboard with known dimensions and sizes though space in front of the camera, it can be used for both the RGB and IR camera, see figure 3.8 and 3.9.

Figure 3.8: RGB calibration

Figure 3.9: IR depth calibration

The ROS camera_pose_calibration package can extrinsically calibrate the Kinect's IR camera to an external RGB camera or the built in one. It is a little bit more tricky to use than the intrinsic camera_calibration package. It basically works by recording the RGB data in a rosbag and then by comparing the replayed rosbag to the live depth data topic stream it produces a ROS tf transform relating the chequerboard in both data streams. In short a tf transform is how ROS transforms a frame to another so for example how it updates the iRobotCreate's odometry data with the wheel encoder measurements. The tf transform is saved to disk and by enabling depth registration in the openni rqt_reconfigure package the openni topics will use the extrinsic tf transform and publish registered topics as well. As a side note for both calibration packages the IR camera's emitter must be covered so as to produce a clean(unspeckled) IR image.

Image handling

In order to manipulate and analyse the Kinect's RGB and depth images a computer vision package will be needed. OpenCV V3.0 is chosen as it is released under a BSD license and hence it's free for academic use. OpenCV has both C++ and python interfaces but since the computer vision scope of this project is minimal and the python API is far easier to use, python was chosen as the programming language for all ROS scripts generated. As mentioned earlier openni publishes the Kinect's RGB and depth data in the ROS Sensors_msgs Image format which OpenCV does not

support. The topic's data must be converted into a numpy type array so OpenCV can handle it. The ROS community has distributed a package for just the purpose, known as cv_bridge. In order to use both OpenCV and cv_bridge the package's dependencies must be changed in the CMakeLists file, this basically just tells ROS where to find each package.

Interfacing with the iRobotCreate

The Turtlebot package is used to send control commands to the iRobotCreate base but the package's launch variables needed to be slightly modifed so that it realises it is just an iRobotCreate not a Turtlebot. This is achieved through setting the environmental variables the package uses to set what type of Turtlebot it is, as there are two mobile base options (the iRobotCreate or the default Kobuki). The following must be incorporated into the setup.bash file in the required package:

- export TURTLEBOT_BASE=create
- export TURTLEBOT_STACKS=circles
- export TURTLEBOT_SERIAL_PORT=/dev/ttyUSB0 (may appear under another ttyUSBn)

The command roslaunch turtlebot_bringup minimal.launch starts the nodes responsible for communications with the above defined environmental variables. The /odom topic publishes the iRobotCreate's pose based on the wheel encoders' measurements at around 30 Hz and is the nav_msgs/Odometry format. The Odometry format is just the pose and velocities of the robot at the time of publishing. The topic which is responsible for actuating the velocity commands is called the /cmd_vel_mux/input/navi and uses the ROS Geometry_msgs Twist message format. The Twist variable takes in all linear and angular velocities for all axes but the iRobotCreate only uses the heading direction linear velocity and z direction angular velocty parts, as it is a ground vehicle. The latching time of the velocity commands are 0.1 seconds so in order to keep the robot moving we would have to publish at 10Hz. Since the main script will be running at roughly 30Hz (Kinect's publishing and odometry topic publishing frequency) this is not an issue.

ROS Network Timing

Not many people realise that the RGB and depth data from the PS1080A microprocessor is sent at different rates with different variances, seen in figure **??**. As

mentioned earlier ROS works with callback functions which occur, like an interrupts, when data is published to a particular topic the subscriber is listening to. Since the RGB image is used to identify the leader/landmarks and the depth image is used to provide relative measurements, the issue that arises is how to trigger a callback function for two topics which publish at different rates. There is not much information available in the ROS documentation except for a couple of misleading forum posts which deal with this issue. After searching through the latest ROS indigo API a package, named message_filters, was seen to be able to cope with this issue. It was originally created to deal with topics which have the same timestamps (publishing rates) but has been recently extended to deal with approximate timestamps. It has the same set-up and parameters as the original TimeSynchronizer except for an extra slop parameter in the constructor that defines the delay (in seconds) with which messages can be synchronised. This allows the program to enter a callback function which performs the image processing and thus applies control commands when the RGB, depth and odometry data is available. Figures 3.10 and 3.11 show the publishing rates of the RGB and depth data respectively. As can be seen the difference in publishing times is similar to the odometry topic and is in the micro second range which for this project's purpose is adequate.

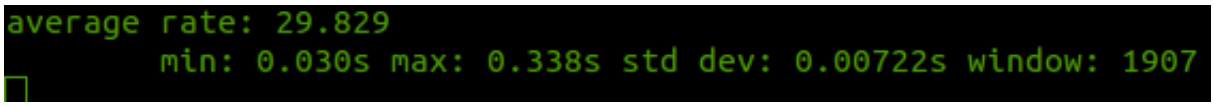

Figure 3.10: RGB publishing rate

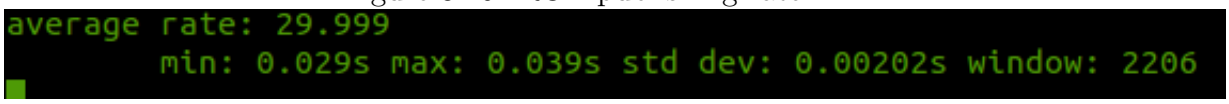

Figure 3.11: Depth publishing rate

Chapter 4

Image Processing Algorithm Design

One of the major challenges in computer vision is determining the shape, location, or quantity of instances of a particular object in an image. A solution to this problem is to provide an algorithm that can be used to find any shape within an image according to parameters needed to describe the shapes. A technique used to achieve this is the Hough Transform (HT) invented by Richard Duda and Peter Hard in 1992. The HT was originally meant to detect arbitrary shapes but was later extended to the more common Circular Hough Transform (CHT). Since all markers in the *Mechanical Design* section are ensured to appear circular in the RGB image this method is ideal thus it is chosen. The method highly depends on converting gray-scale images to binary images through edge detection techniques such as Canny or colour thresholding with morphological operations techniques. I refer the reader to a very good explanation of how the HT/CHT and the edge detection algorithms used in this section work [65].

This section first starts out by evaluating the different preprocessing algorithms used with the CHT. A final preprocessing algorithm will be chosen based on execution time and number of mismatches in identifying the marker with the CHT. Since all the markers are different coloured spheres only a single sphere of the leader's will be chosen as the marker for the comparison. Following the preprocessing investigation the methods used to identify and measure the leader and landmarks are outlined. Finally improvements on the methods are outlined and explained.

4.1 Preprocessing Investigation

A rosbag will be used to store the red sphere moving around in 3D space in front of the Kinect. This stored data will form the basis to judge the number of circle identification mismatches. Two techniques will be investigated namely, the Canny Edge Detection Algorithm and HSV Thresholding with Morphological Operations. When the rosbag file is played back the ROS script Node called PreprocessingEval will output to a csv file if a circle is detected in the frame for each method. The amount of times a detected circle will be counted using excel. All imaging functions used are available in OpenCV.

Canny Edge Dectection Algorithm
The Canny Edge Detection is by far the most popular edge detection algorithm hence it being chosen, it was first developed by John F. Canny in 1986. The algorithm first applies a Sobel filter in the horizontal and vertical direction, from that it can find the gradient magnitude and direction for each pixel. It then follows to suppress edges which are not the local maximum in its neighbourhood in the direction of the gradient, basically produces thin lines showing the edges. A hysteresis thresholding is applied with a pre-set minimum and maximum value, this decides which edges to keep in the binary image.

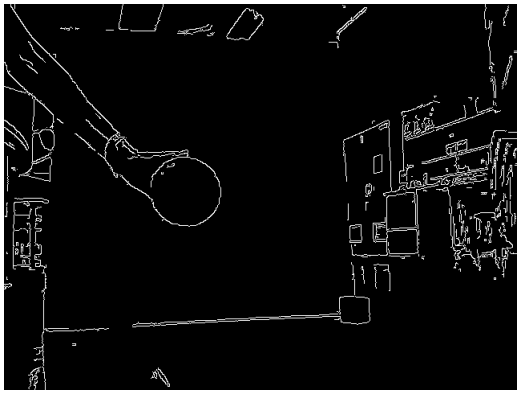

Figure 4.1: Canny Edge Detection

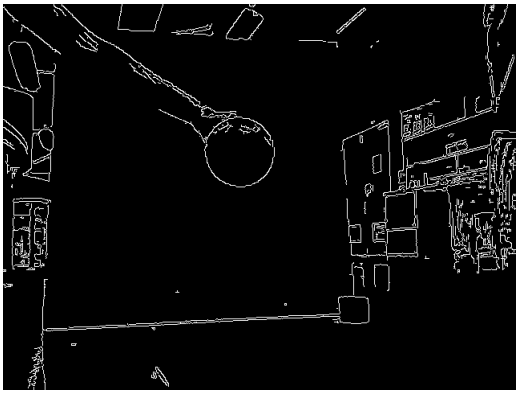

Figure 4.2: Movement of sphere

HSV Thresholding with Morphological Operations
Firstly the RGB image is converted to the HSV space whereby colour identification is made easier. Since the sphere is red it will be between a definitive HSV value space, this thresholding produces a binary image of where the colour red is found. There are two morphological transformations namely, erosion and dilation. The

Table 4.1: Canny Edge Detection Circle Detection

Trial Number	Circle Detection Percentage
1	92.7
2	89.4

type of kernel used will produce either one of the transformations and must only be performed on binary images. Erosion erodes away the boundaries of the foreground object by applying the kernel to each pixel in the image, if all the pixels in the kernel equal 1 (white) then that particular pixel will be considered 1 else it will be eroded i.e made 0. Dilation is the opposite to erosion, it increases the size of the foreground object. The combination of these two morphological transformations will remove noise and fill the inner foreground object's white space, they are known as opening and closing respectively.

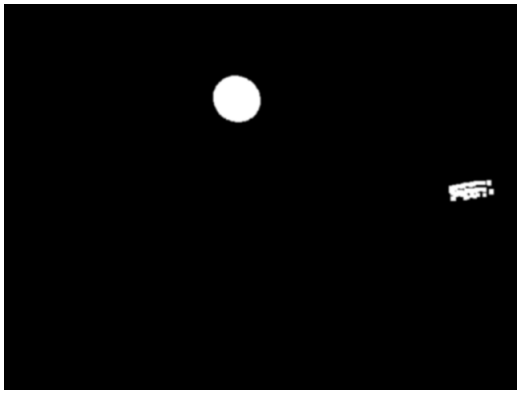

Figure 4.3: HSV Thresholding with Morphological Operations

Figure 4.4: Movement of sphere

Table 4.2: HSV Thresholding with Morphological Operations Circle Detection

Trial Number	Circle Detection Percentage
1	97.2
2	93.9

Investigation Conclusion

Each trial was performed with 200 image frames with a different rosbag file, trial one was performed at 1m away where trial 2 was at 2m. Table 4.2 shows the detec-

tion percentage of HSV Thresholdng is about 5 percent higher on average compared to the Canny Edge Detection, table 4.1. The Canny Edge Detection performed adequately but the robustness the HSV Thresholding allows it to out perform the Canny Edge Detection. The downfall is due to the Canny Edge Detection sometimes picking up circles in the background which are between the set radius interval in the CHT function. For these reasons the Thresholding method was chosen to be the identification algorithm for both the leader and landmarks.

4.2 Leader Measurement

Since the center location of the two leader's spheres are known through the identification algorithm, the z value from the follower to both these spheres can be found from the calibrated Kinect's depth image. From the *System Modelling* section there are three measurements needed to describe the relative pose of the leader from the follower namely, the two polar coordinates and the difference in heading. These values can be calculated from understanding the cosine law and some clever geometry manipulation, the steps are outlined below:

The polar coordinates value row:

By taking advantage of the known distance between the two spheres on the leader plus with the measured depths to either one. The cosine rule is used to find the angle alpha and beta, see Appendix A. Through knowing these angles and the magnitude of the sides the relative coordinates can be calculated as follows:

$$\rho = \sqrt{zright^2 + robotlength^2 - zright * robotlength * \cos(alpha)} \quad (4.1)$$

The bearing angle phi:

The middle pixel coordinate of the two spheres is found. The CHT provides both the circle pixel coordinate and radius, so by knowing the actual size of the sphere a pixel to real distance ratio is formed. This ratio is then used to convert how far of the middle of the spheres are to the follower's heading direction (i.e. how far off the x=320 axis), see figure 4.5. There are two cases in this scenario, leader to either to the right 4.2 or left B.3 of the follower's

heading axis.

$$\phi = \arccos\left(\frac{|xmid * ratio|}{\rho}\right) \tag{4.2}$$

$$\phi = \arcsin\left(\frac{|xmid * ratio|}{\rho}\right) + \frac{\pi}{2} \tag{4.3}$$

The difference in heading angle gamma: We can tell the direction of the leader's relative heading by whether alpha or beta are greater, see figure 4.5. The angle the zright line and the horizontal axis make form the starting point for the calculations, named ϕ_r. It is very much like calculating ϕ with xmid but for the right sphere using xright. Using the same two cases above (right or left of the center x axis) for whether alpha or beta are greater , the following equations calculate gamma:

- alpha>beta:
  $(rightside)$

$$\gamma = -(\alpha - \arcsin\left(\frac{|xright * ratio|}{zright}\right)) \tag{4.4}$$

  $(leftside)$

$$\gamma = -(\alpha - \arccos\left(\frac{|xright * ratio|}{zright}\right)) \tag{4.5}$$

- alpha<beta:
  $(rightside)$

$$\gamma = \alpha - \arcsin\left(\frac{|xright * ratio|}{zright}\right) \tag{4.6}$$

  $(leftside)$

$$\gamma = \alpha - \arccos\left(\frac{|xright * ratio|}{zright}\right) \tag{4.7}$$

4.2.1 Landmark Measurement

The identification algorithm returns all the detected circles' center coordinates and radius of every landmark found in the image. As will be outlined in the *Localisation* section only one landmark measurement is used to update the follower's pose. The method to choose which landmark to measure in the image will be based on the largest returned radius from the CHT, this will cause the depth measurement to be the most accurate, see figure 4.6. Since the landmark measurements only require a relative range and bearing measurement the first two steps outlined in the above *Leader Measurement* section can be slightly modified. The circle coordinates

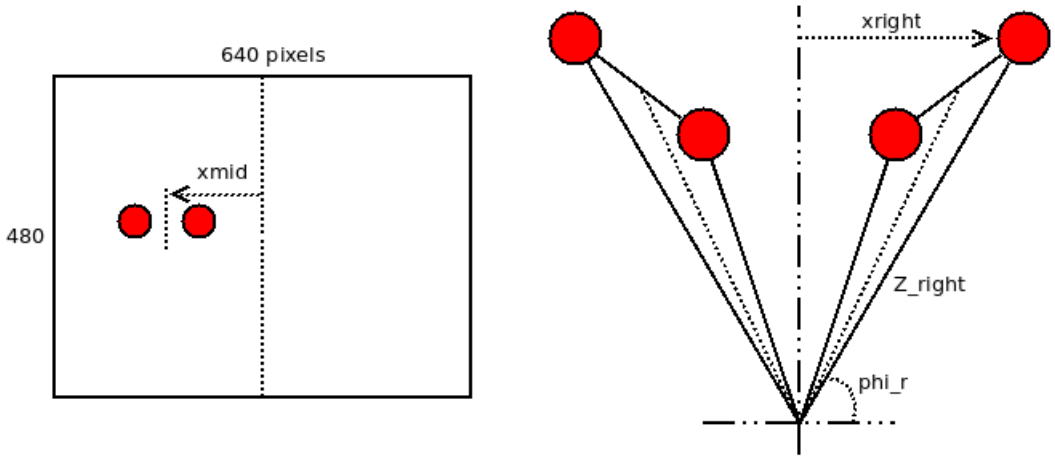

Figure 4.5: Analytical explanation of xmid and xright to calculate phi and gamma

corresponding to the largest radius will be used to find the depth to that sphere through the Kinect's depth map. A landmark assignment algorithm will iterate through each colour of all the landmarks and match the found landmark with the corresponding index. It achieves this by thresholding a small area around the cicles's center coordinates to work out the index.

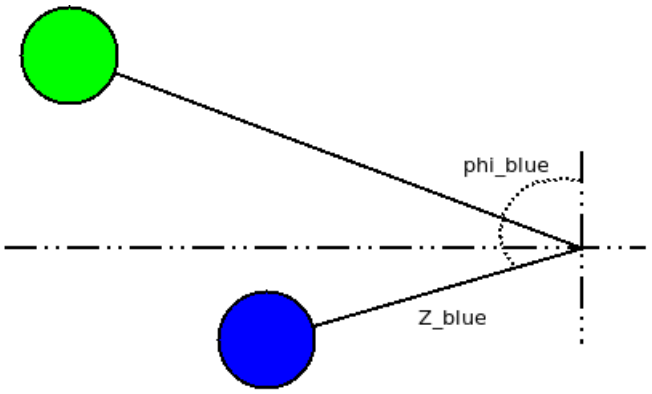

Figure 4.6: Landmark Measurement

4.3 Final Algorithm

The complete image processing algorithm takes in the RGB and depth images from the Kinect and returns the relative measurements from the follower to the leader and landmarks. The functions used in the algorithm are in the Measure python module (See CD) and are defined as follows:

getLeaderCenterPoints(RGB_image): Takes in an OpenCV format image and performs the identification algorithm and returns the two center points and radii of the leader's red spheres.

getLandMarkPoints(RGB_image): Takes in an OpenCV format image and performs the identification algorithm and returns the center points and radii of all the landmarks.

getLeaderMeasurements(leader_center_points,depth_image) : Takes in the leader's center points and returns the relative coordinates namely, $[\rho, \phi, \gamma]$.

getLandMarkMeasurements(leader_center_points,depth_image) : Takes in the landmark's center point and returns the relative coordinates namely, $[\rho, \phi]$.

4.4 Optimisation

The above outlined algorithms are very simple so they are robust but there are a couple of methods to improve their runtime.

- **Crop Image**
  The depth image map is only accurate in a 320x240 resolution which is far less than the RGB 640x320. If the depth map is indexed outside of its resolution it will either return a very large value or zero. It therefore makes sense to crop the RGB image down to 320x240 before the identification algorithm is run as the program would not be able to find any measurements outside of this resolution. This cropping of the resolution will allow the identification algorithm to run faster as less pixels need to be iterated through.

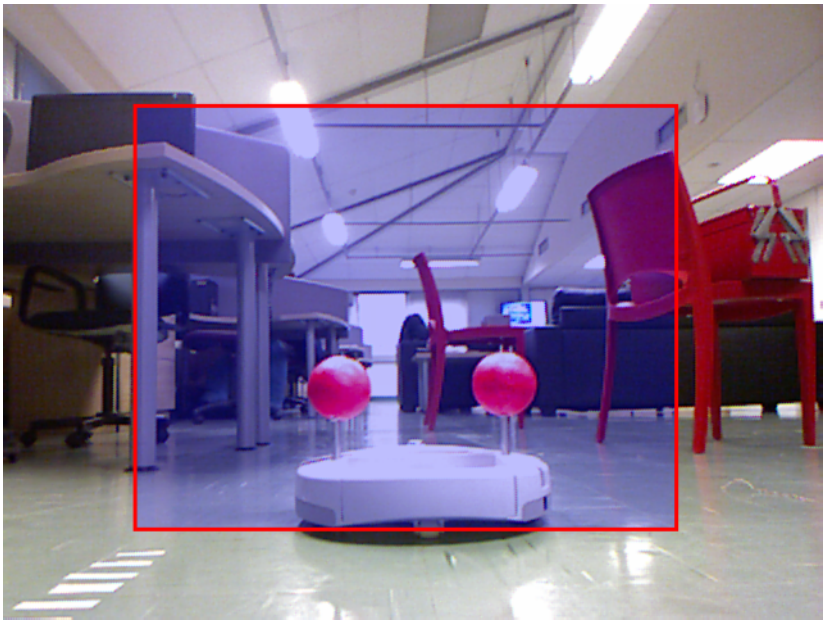

Figure 4.7: Improved Design

- **CHT radius interval parameter**
  The OpenCV CHT function parses two parameters namely, min_radius and max_radius. These parameters govern what circles the CHT will consider in the hough space based on the set pixel radius interval passed into the function. If no future knowledge is available to give an indication of the radius interval the function takes considerably longer to find the circles as it has to iterate through the hough space. This computation time can be decreased by storing the last known radius of both the leader's spheres and the last measured landmark then defining a $+-10$ interval based on those stored radii.

Chapter 5

Control Design

In any control set-up there are certain steps that must be followed to ensure the design produces adequate results. The system to be controlled's dynamics must first be modelled through analysing the set-up. Once sets of equations defining the system have be defined, the sensors' and actuators' physical limitations must be determined through "step test" data so as to design around these constraints. These steps are encompassed in the *System Modelling* section. A controller must be designed/selected and shown to mathematically cause the system to achieve the desired results, *Controller Selection*. The rest of this chapter outlines the other control design steps needed for the controller to achieve formation control.

5.1 System Modelling

5.1.1 iRobotCreate's Dynamics

In general the configuration of a robot can be described by six parameters. Three-dimensional Cartesian coordinates plus three Euler angles pitch,roll and yaw. Since the iRobotCreate is a ground robot its pose is described with a two-dimensional Cartesian coordinate (x,y) and a single Euler angle yaw (heading). In practice you find two types of motion models namely, Odometry based and Velocity-based (dead reckoning). Dead reckoning is a mathematical procedure for determining the present location of a vehicle without sensor feedback information about what actually happened in the real world. It is achieved using the previous pose and command velocities to produce the current pose hence the velocity-based motion model is dead

reckoning. I refer the reader to Appendix B.1, it explains how the two different motion models work and why it is necessary to understand both.

The two types of motion models presented in Appendix B.1 are very much related to do with the same type motion. In essence the velocity-based model is centered around the differential drive kinematics (DDK) of the robot to show ideally how the robot will move through its environment where as the odometry-based model shows what actually happened using the same DDK of the robot. So the velocity-based model is used to predict where the robot will go and the odometry is used to compare what actually happened after the velocity commands have been applied. The relevance of being able to predict the motion and then compare it to a noisy measurement of what actually happened will become apparent in the *Localisation* section.

5.1.2 Leader-follower Formation Dynamics

This short section outlines the relative coordinate model between the leader and follower robot, it is best analytically understood, see figure 5.1.

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \sqrt{(x_L - x_F)^2 + (y_L - y_F)^2} \\ \arctan\left(\frac{y_L - y_F}{x_L - x_F}\right) \\ \theta_L - \theta_F \end{bmatrix} \tag{5.1}$$

Using the decribed motion model above and differentiating gives:

$$\begin{bmatrix} \Delta q_1 \\ \Delta q_2 \\ \Delta q_3 \end{bmatrix} = \begin{bmatrix} -c_2 & 0 \\ \frac{s_2}{q_1} & -1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_F \\ \omega_F \end{bmatrix} + \begin{bmatrix} c_{32} & 0 \\ \frac{s_{32}}{q_1} & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_L \\ \omega_L \end{bmatrix} \tag{5.2}$$

where $c_i = \cos q_i$, $c_i = \sin q_i$ and $c_{ij} = \cos(q_i - q_j)$, $c_{ij} = \sin(q_i - q_j)$.

5.2 Controller Selection

The literature review outlines the different types of controllers used to achieve the leader-follower formation control. Complex controllers such as neural/sliding mode were out of the author's undergraduate control understanding and so were not considered. Linearising controllers about an operating point[35] and state feedback lin-

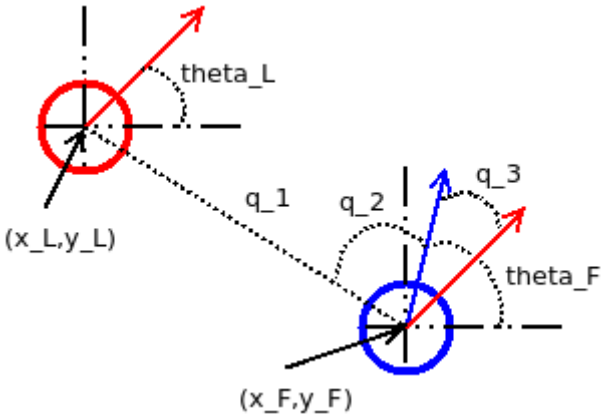

Figure 5.1: Relative Coordinates

earising controllers [40] had already been experienced during undergraduate studies, so relatively nothing new would be understood by exploring them further. Receding Horizon (RH) controllers [34] appeared to be the most common type in formation control because if some future knowledge is known, which it generally is, then an optimised trajectory can be planned out where the magnitude in error and control action is taken into account. However RH studies often include far more design issues, such as actuator velocity constraints [42], into the formation controller which considerably complicates the understanding of how the controller works. Intuitively error based tracking controllers [48] are the easiest to understand and since the suggested paper by the author's supervisor is an error base tracking control design, this type of controller was chosen for this study. Following the *System Modelling* section the selected controller [47] is mathematically shown and then simulated, in Matlab, for further validation. The controller assumes there is no sensor or actuator noise, true pose values are always known and the system is continuous. The following constants, control laws and conditions are derived and proven in Appendix A.4

Due to the non-holonomic nature the controller can only control a combination of all three states. The controller can prioritises the control of q_2 over q_3 or vice versa. The error between the desired and actual relative formation coordinates is:

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} = \begin{bmatrix} q_1 - q_{1d} \\ q_2 - q_{2d} \\ q_3 - q_{3d} \end{bmatrix} \tag{5.3}$$

It assumes $|q_i|\langle \frac{\pi}{2}$ for $i = 1, 2$, $|q_1|\rangle 0$ and the desired $|q_{2d1}|\langle \frac{\pi}{2}$. The desired value of q_3 is assumed to be zero.

$$v = -k_1(\Lambda_1\epsilon_1 + \Lambda_2\epsilon_2\epsilon_3) \tag{5.4}$$

$$\omega = k_2(p_{22}\frac{t_2}{q_2}\epsilon_1 + p_{22}\epsilon_2 + p_{33}\epsilon_3 + \frac{p_{22}}{q_1c_2^2}\epsilon_1\epsilon_2) \tag{5.5}$$

where k_1 and k_2 are the postive gains. The following conditions must be satisfied:

$$p_{11} > p_{22}(\frac{t_2}{q_1})^2 + \frac{p_{22}}{q_1^2}(|\frac{s_2}{c_2^3} + \frac{s_2}{c_2}|)|\epsilon_2| \tag{5.6}$$

and either 5.7 or 5.8 must be satisfied:

$$p_{22} < p_{33} \tag{5.7}$$

$$p_{22} > p_{33}\ and\ \frac{|\omega_L|}{v_L} < \frac{1}{q_1} \tag{5.8}$$

Final Controller Remarks

Although through the manipulation of the gains k_1 and k_2 it can be shown that the error states will become uniformly bounded, there is no way to mathematically show what the gains should be in order for certain bounds to be achieved and at what rate they will converge. This means that the gains must be experimentally changed for different types of formation trajectories in order to realise what combination of gains will result in what error bounds and what convergence rates. It should first be determined through simulations and then finally on the actually physical system, as the robot's velocity constraints might prevent certain gains from performing as in the simulations.

5.3 Localisation

The selected controller assumes the true pose value for each robot is known in order to output the required velocity commands to keep the formation. However, as seen in the *System Modelling* section the iRobotCreate like any robotic platform has noise in both its odometry and sensors. So when the controller sends out the required velocity commands the robot's actuators do not perform ideally due to many different types of errors. The feedback provided through the odometry data and visual both provide noisy measurements of what really happened in the robot's environment.

So to acquire a more accurate pose estimation the two feedback sensors' measurements are fused through an algorithm rather than just using the odometry data where errors will accumulate over time. These algorithms take a probabilistic approach through modelling the system as Bayesian network and then take advantage of knowing the sensors' probabilty distributions to produce a far more accurate posterior pose estimation.

The most common type of recursive estimation for a non-linear system is the Extended Kalman Filter (EKF). The Kalman Filter proposed was by R.E. Kalman in 1960 but it could only handle linear systems and so it was later extended to handle non-linear systems hence EKF. Another filter known as the Unscented Kalman Filter (UKF) can also handle non-linear systems but it takes advantage of the unscented transform rather than through the use of jacobians. An in-depth explanation and derivation of the EKF and UKF is in Appendix A.5. Even though both an EKF and UKF are designed and simulated in the *Simulation* section, only the EKF will be implemented on the physical set-up. The noise covariance matrices R_t and Q_t used in the state estimators below are derived in Appendix A.3. The following sections derive the state transition and state measurement functions for the state estimators.

5.3.1 Follower pose state

The motion noise model R_t for the position reading error is larger than what it should be as it does not take into account how the heading direction of the robot changes the error in reading the x and y states. So it is the supremum of the largest error for both, meaning the odometry reading is actually slightly more accurate than what the EKF and UKF will think due to R_t. However due to the relatively low measurement error from the Kinect the state estimators will most likely regard the measurement as more true. For the EKF in simulations, the prediction of the state uses the motion model with some additive noise to simulate the odometry noisy measurement but in practice the predicted state will be the current change in odometry reading. The following are the non-linear state transition g and measurement model h for estimating the follower's pose:

Function g

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta T) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta T) \\ \omega\Delta T \end{bmatrix} \tag{5.9}$$

Jacobian of g

$$G = \begin{bmatrix} 1 & 0 & -\frac{v}{\omega}\cos\theta + \frac{v}{\omega}\cos(\theta + \omega\Delta T) \\ 0 & 1 & -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta T) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.10}$$

Function h

$$\delta = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \begin{bmatrix} x_{landmark} - \overline{x_{follower}} \\ y_{landmark} - \overline{y_{follower}} \end{bmatrix}$$

$$q = \delta_T \delta$$

$$\overline{z_t} = \begin{bmatrix} \sqrt{q} \\ \arctan\frac{\delta_y}{\delta_x} - \overline{\theta_{follower}} \end{bmatrix} \tag{5.11}$$

Jacobian of h

$$H = \frac{1}{q}\begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 \\ \delta_y & -\delta_x & -q \end{bmatrix} \tag{5.12}$$

5.3.2 Leader-follower relative state

The state of the follower is not the only state that needs to be estimated but the leader-follower relative state vector must also be estimated so the controller can generate commands to accurately keep the formation. The function g and the necessary constants are set out in the *Leader-follower Formation Dynamics* section where the differential is just added to the current vector. The jacabian is as follows:

Jacobian of g

$$G = \begin{bmatrix} 1 & v_f s_2 + v_l s_{32} & -v_l s_{32} \\ -v_f\frac{s_2}{q[0]^2} - v_l\frac{s_{32}}{q[0]^2} & 1 + v_f\frac{c_2}{q[0]} - v_l\frac{c_{32}}{q[0]} & v_l\frac{c_{32}}{q[0]} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.13}$$

The sensor model h is assumed to measure the leader's relative coordinates directly since it does not have to transform the measurements to be compared to values in the global frame like the landmarks, hence relative. So the sensor model h is just a 3x3 identity matrix. As a side note there are two ways to achieve estimating the leader, one through estimating the relative coordinates and then using them to

update the leader's pose or the other to first use the measured relative coordinates to determine the measured leader's pose and then to use a motion model, like the follower pose EKF, to gain a better estimate of the leader. The former was chosen as the function g had already been determined and was more intuitive.

Chapter 6

Final Design Overview and Experimental Methodology

6.1 Final Design Overview

The final designed algorithm can be shown in a ROS type node structured flow diagram, see figure 6.1.

6.2 Experimental Methodology

The final design flow diagram illustrates the many different aspects that need to be combined in order for the entire system to operate properly. To insure the final design works as planned a bottom up approach for testing each different subsystem will be used. The main essential systems such as the controller and state estimators will be simulated to validate their design so as to allow easier debugging of the implemented system as problems should not be associated with the design but rather the hardware etc. The final implemented system will be rigorously tested to ascertain whether the system performs as specified by the scope of this project. The steps for collecting the experimental data in the *Simulation Results* section and *Implementation Results* section are described below:

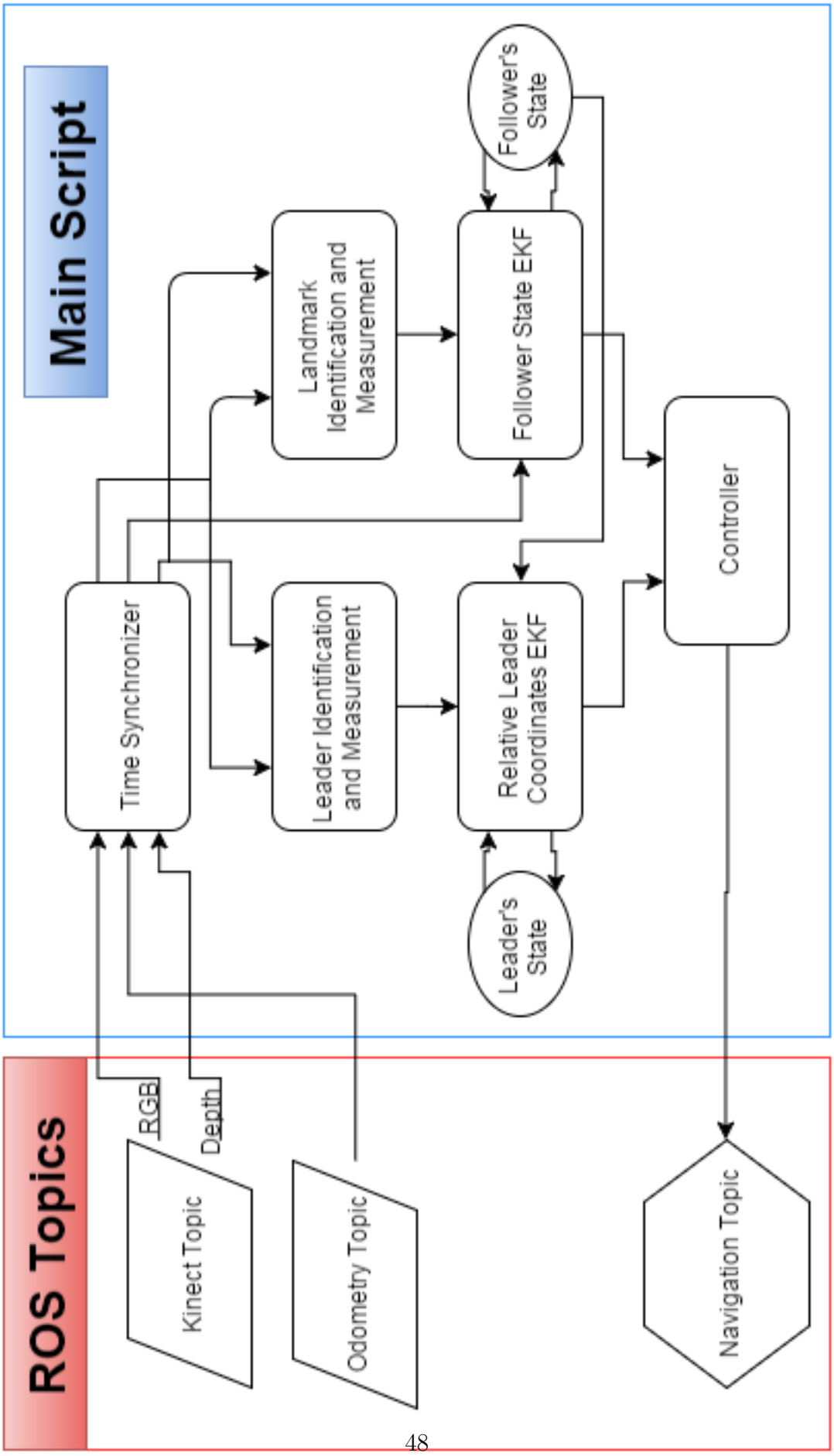

Figure 6.1: Final Design Algorithm

6.2.1 Simulations

The selected controller was mathematically proven, in Appendix A.3, to drive the errors in the leader-follower formation set-up towards a bounded region in the state space through the manipulation of the controller's gains. For further validation just the controller will be simulated in Matlab's Simulink environment to show how the relative coordinate errors tend towards zero. The controller and the two different state estimators, EKF and UKF, will be incorporated in a Matlab script and the combined systems will be shown to still maintain the formation in the presence of simulated environmental noise. Since the time between iterations does not matter in simulations, as all the systems are digital, so convergence speed etc. can be though of in number of iterations but for simplicity the time difference will be one second.

Three simulations will be performed on each state estimator where by the noise transition and measurement matrix will be varied. The following three noise matrices are used to test both the follower's pose state estimator and the relative coordinate state estimator. Where the measurement covariance matrix Q_t shown below, becomes a 3x3 matrix for the relative coordinate state estimator where the third diagonal entry is just the same as the second. The noise matrices R_t and Q_t do not correspond to the modelled matrices for the iRobot and Kinect as these were too low noise to test the state estimators and so values were chosen to show off the two state estimators' efficacy in simulations.

High Noise

$$R_t = \begin{bmatrix} 0.9^2 & 0 & 0 \\ 0 & 0.9^2 & 0 \\ 0 & 0 & 0.08^2 \end{bmatrix}$$

$$Q_t = \begin{bmatrix} 0.1^2 & 0 \\ 0 & 0.08^2 \end{bmatrix}$$

Medium Noise

$$R_t = \begin{bmatrix} 0.09^2 & 0 & 0 \\ 0 & 0.09^2 & 0 \\ 0 & 0 & 0.04^2 \end{bmatrix}$$

$$Q_t = \begin{bmatrix} 0.08^2 & 0 \\ 0 & 0.04^2 \end{bmatrix}$$

Low Noise

$$R_t = \begin{bmatrix} 0.009^2 & 0 & 0 \\ 0 & 0.009^2 & 0 \\ 0 & 0 & 0.01^2 \end{bmatrix}$$

$$Q_t = \begin{bmatrix} 0.04^2 & 0 \\ 0 & 0.01^2 \end{bmatrix}$$

6.2.2 Implementation

The final designed ROS script will be edited so that all internal data in the script will be written to multiple .CSV text files every iteration of the main loop so that the results can be presented in a readable format. The data to be stored for the evaluation of the implementation is as follows:

Leader and Landmark identification misses

The type of dataset used to evaluate the identification algorithm is the same as in the *Image Processing* but the data from the implemented system involves the iRobotCreate moving and so this motion might cause different results which must be investigated. The ability of the leader and landmark state measurement algorithm will not be tested as the equations derived are adequate to show if depth data is available and identification algorithms have produced matches the required measured states are produced with accuracy determined only by the noise of the sensors.

As stated in the scope of the project since there is no aerial webcam available to track the positions of the iRobots in real time so as to validate the experimental results. So instead the final poses of the iRobots will be measured and compared to the estimated states by the system, this constraint is deemed satisfactory as it still validates the state estimators' ability.

Estimated Leader/Follower pose states and Relative coordinate state

Multiple starting scenarios for when the leader is stationary or moving will be investigated with different gain sets for the controller. The final states of each data set will be compared to the true state value for the data set from measuring using the tiled grid and tape measures/protractors. The resulting mean and standard deviation of the error for all the data sets from each particular

scenario will be stated in the *Implementation Results* section. The raw data for each data set is in the CD and the all the plots for one of each scenario data set will be shown in Appendix D to provide a better insight into what happened during the scenario. As stated in the scope of this project since no Netbooks for the iRobotCreates were available the leader will be either stationary or pulled along by hand at a relatively constant velocity.

Chapter 7

Simulation Experiments

7.1 Controller

The study [47] simulates their proposed controller with a circular and straight line leader trajectories with different desired relative coordinates. The simulink model of the controller is shown in C.1, the simulation steps will be similar to what was done in the study so as to confirm the study's results. Only the plot of the robots' positions will be shown in this section, figures 7.1 and 7.2, refer to Appendix C for the error and velocity plots for each leader trajectory.

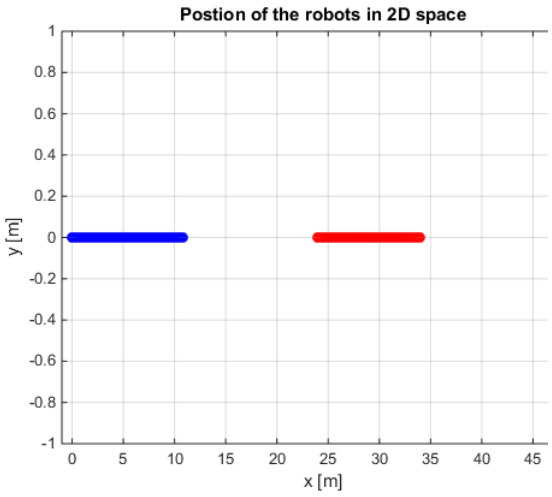

Figure 7.1: Linear leader velocity

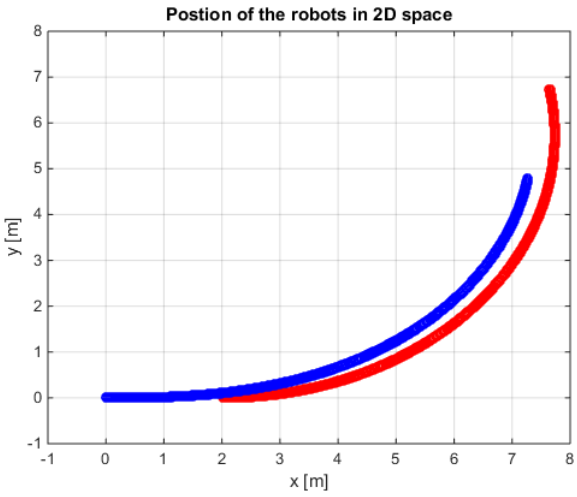

Figure 7.2: Circular leader trajectory

7.2 State Estimators

7.2.1 Follower Pose Estimators

Figures C.6 to C.11 show the results of the EKF and UKF for just estimating the follower's pose with the surrounding landmarks.

7.2.2 Leader Pose Estimator

Figures C.12 to C.14 show the results of the EKF estimating both follower's pose using surrounding landmarks and the leader's pose through estimating the relative coordinates.

7.3 Final Control System

Figures 7.3 and 7.4 show the final simulated system where the EKF is estimating the follower's pose and the relative coordinate state with low and high noise respectively. Since the controller only simulations showed the ability of the controller to follow a straight line and circular leader trajectory, the final simulated design shows a much more complex motion. As can be seen in the figures 7.3 and 7.4 the leader's trajectory changes at around x = 20 and y = 35 step mark to a circular arc.

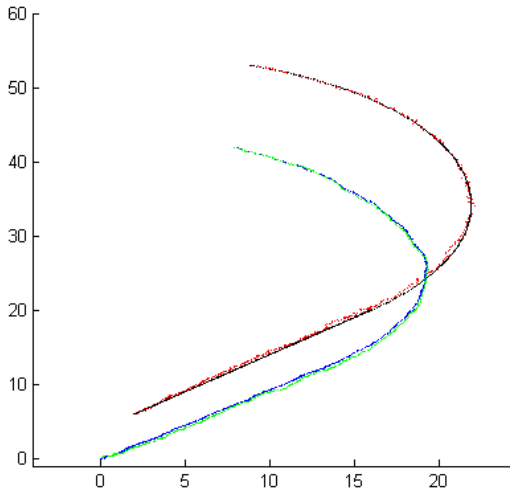

Figure 7.3: Both state estimators for leader follower control(low noise)

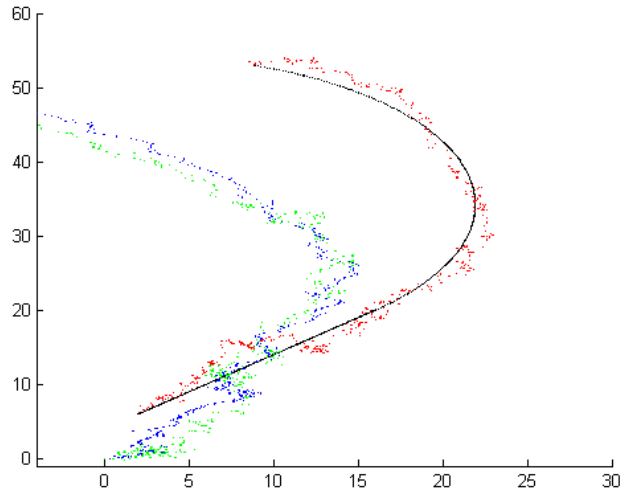

Figure 7.4: Both state estimators for leader follower control(high noise)

Chapter 8

Implementation Results

Note: All position values and angular values in the following tables are absoluted and in [$\times 10^{-3}m$] and [$\times 10^{-3}rad$] respectively. The gain sets are laid out in D.1.

8.1 Zero desired polar angle with stationary leader

8.1.1 In line start

Five data sets with gain set 1 were used the average is shown in table 8.1. The xy positions of the robot during one of the data sets is shown in D.1.

Table 8.1: In line start results

Missdetection	mean	std dev
Leader[%]	93.2	2
Landmark[%]	89.4	2.4

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	8.2	0.2	1.3	0.02	1.8	0.01
Follower Estimate Pose	3.9	0.1	0.12	0.006	0.22	0.01
Leader Estimate Pose	28.1	2	18	1.2	61.2	0.9

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	56.8	0.04	15.3	0.06	75.3	0.06

8.1.2 Angle offset start

Five data sets with gain set 2 were used the average of all the results is shown in table D.6. The xy positions of the robot during one of the data sets is shown in D.6.

Table 8.2: Angle offset start results

Missdetection	mean	std dev
Leader[%]	94	1.9
Landmark[%]	88.2	2.3

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	7.8	0.18	23.2	0.06	21.1	0.08
Follower Estimate Pose	3.1	0.17	0.18	0.004	0.16	0.012
Leader Estimate Pose	38.5	3.4	2102.6	28.5	115	4.5

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	76.8	0.06	10.3	0.09	89.3	0.11

8.2 Zero desired polar angle with moving leader

8.2.1 In line start

Five data sets with gain set 1 were used the average of all the results is shown in table 8.3. The xy positions of the robot during one of the data sets is shown in D.11.

8.2.2 Angle offset start

Five data sets were used the average of all the results is shown in table 8.4. The xy positions of the robot during one of the data sets is shown in D.16.

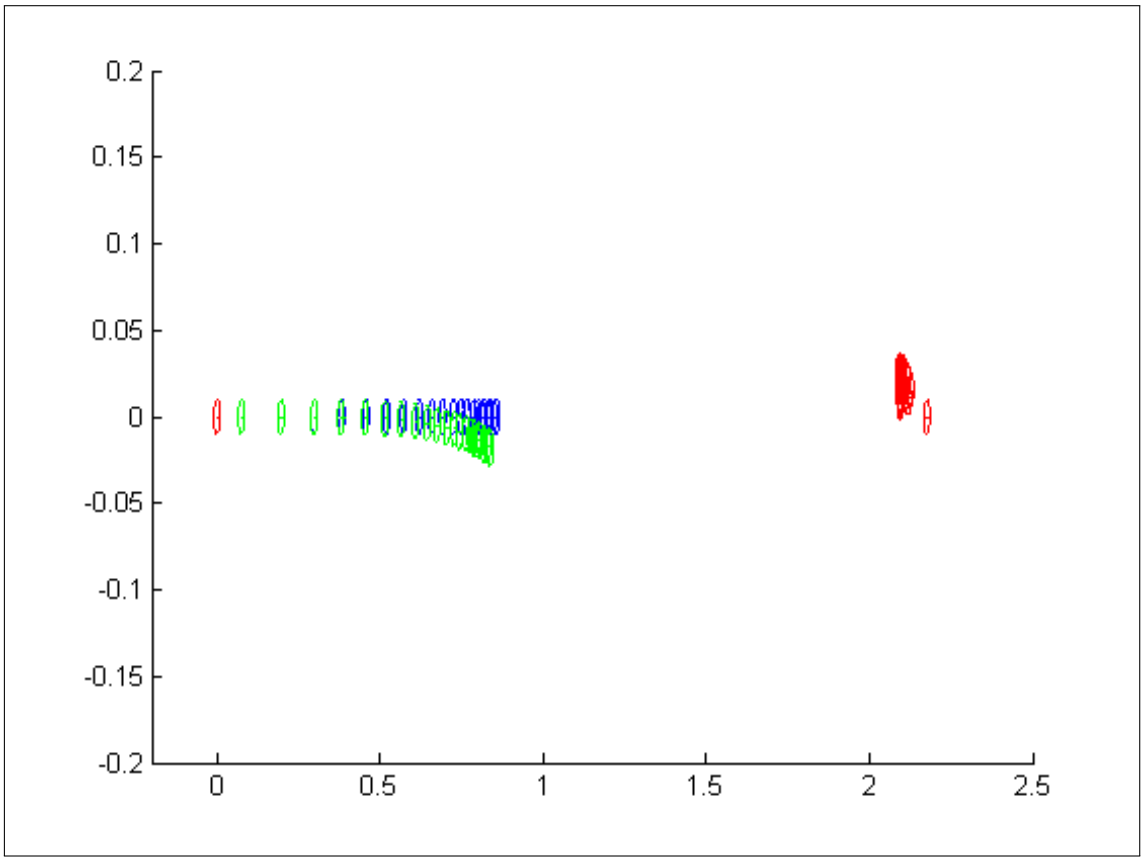

Figure 8.1: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

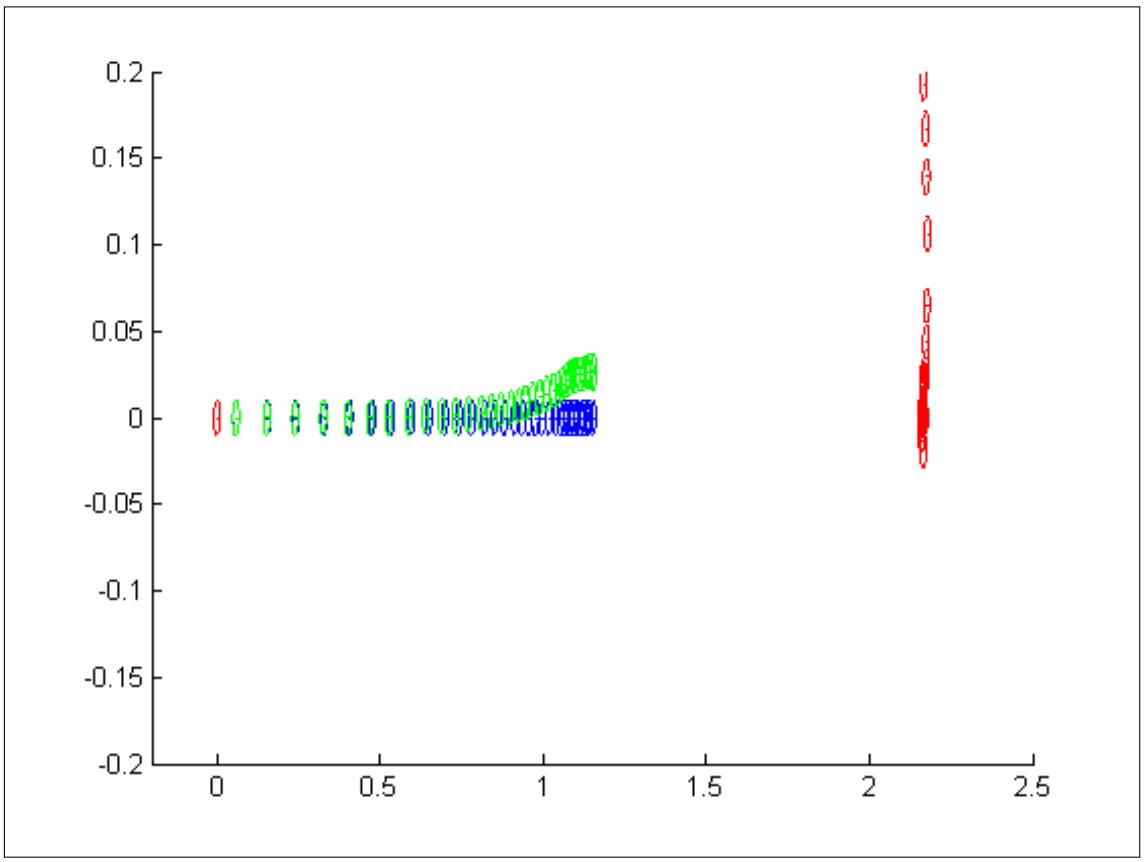

Figure 8.2: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

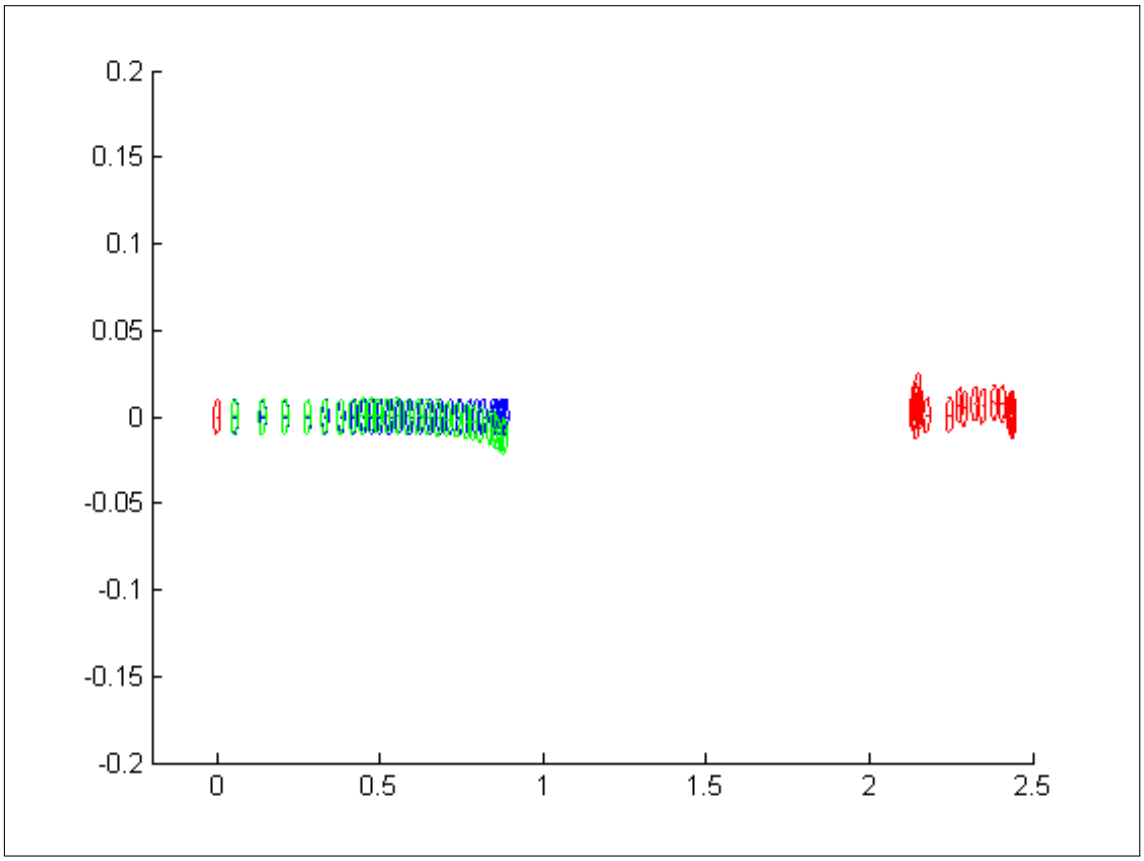

Figure 8.3: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

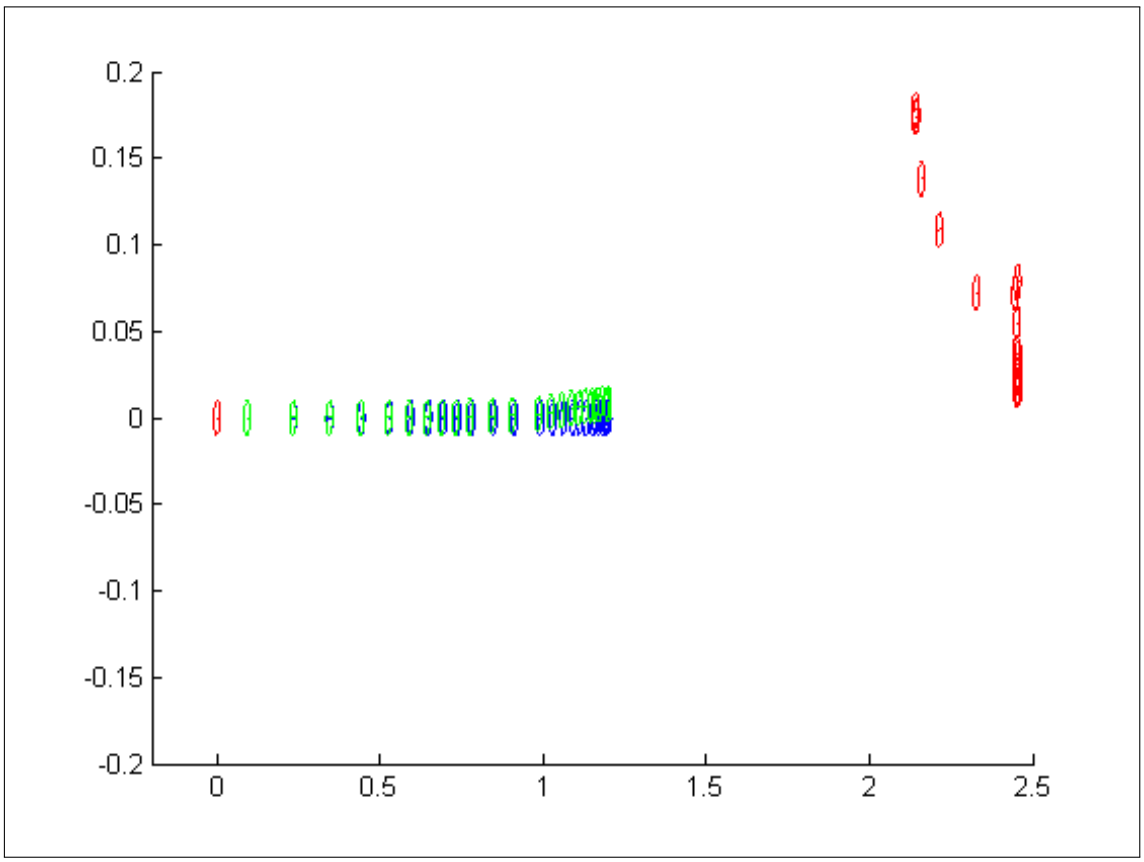

Figure 8.4: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

Table 8.3: In line start results

Missdetection	mean	std dev
Leader[%]	94.	1.8
Landmark[%]	91.1	2.2

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	8.3	0.19	1.4	0.01	2	0.04
Follower Estimate Pose	3.5	0.09	0.14	0.007	0.24	0.02
Leader Estimate Pose	16.1	1.4	13.2	1.4	65.2	0.8

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	57.8	0.05	44.5	1	10.3	4

Table 8.4: Angle offset start results

Missdetection	mean	std dev
Leader[%]	91.2	1.5
Landmark[%]	88.2	2.1

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	9.3	0.9	20.4	0.8	10.9	0.4
Follower Estimate Pose	4.2	0.7	0.8	0.8	0.44	0.5
Leader Estimate Pose	20.1	1.5	1907.4	22.4	112	7.8

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	64.7	0.6	10.3	0.9	15.3	1.1

8.3 Non Zero desired polar angle

8.3.1 $\phi = 10^\circ$ with stationary leader

The desired polar angle is set to 10 degrees and starts out in a zero error start. Three data sets were used the average of all the results is shown in table 8.5. The xy positions of the robot during one of the data sets is shown in D.21.

Table 8.5: Zero Angle offset start results

Missdetection	mean	std dev
Leader[%]	94.6	1.7
Landmark[%]	91.6	1.1

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	11.8	1.1	6.3	0.8	6.1	0.7
Follower Estimate Pose	4.8	0.9	0.9	0.6	0.54	0.5
Leader Estimate Pose	146	2.2	99	2.8	145.3	3.1

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	66.7	0.6	32.6	1.4	91.2	1.7

8.3.2 $\phi = 10°$ with moving leader

The desired polar angle is set to 10 degrees and starts out in a zero error start. Three data sets were used the average of all the results is shown in table 8.6. The xy positions of the robot during one of the data sets is shown in D.26.

Table 8.6: Zero Angle offset start results

Missdetection	mean	std dev
Leader[%]	92.4	1.3
Landmark[%]	87.6	1.7

Error in final state	Postion x		Postion y		Heading θ	
Value	mean	std dev	mean	std dev	mean	std dev
Follower Odometry Pose	45.1	2.1	6.2	0.9	7.1	0.5
Follower Estimate Pose	3.8	0.7	0.4	0.5	0.38	0.6
Leader Estimate Pose	19.2	0.9	45	3.4	100.3	2.8

Error in final state	ρ		ϕ		γ	
Value	mean	std dev	mean	std dev	mean	std dev
Relative Coord State	63.7	1.5	27.2	1.9	97.5	2.1

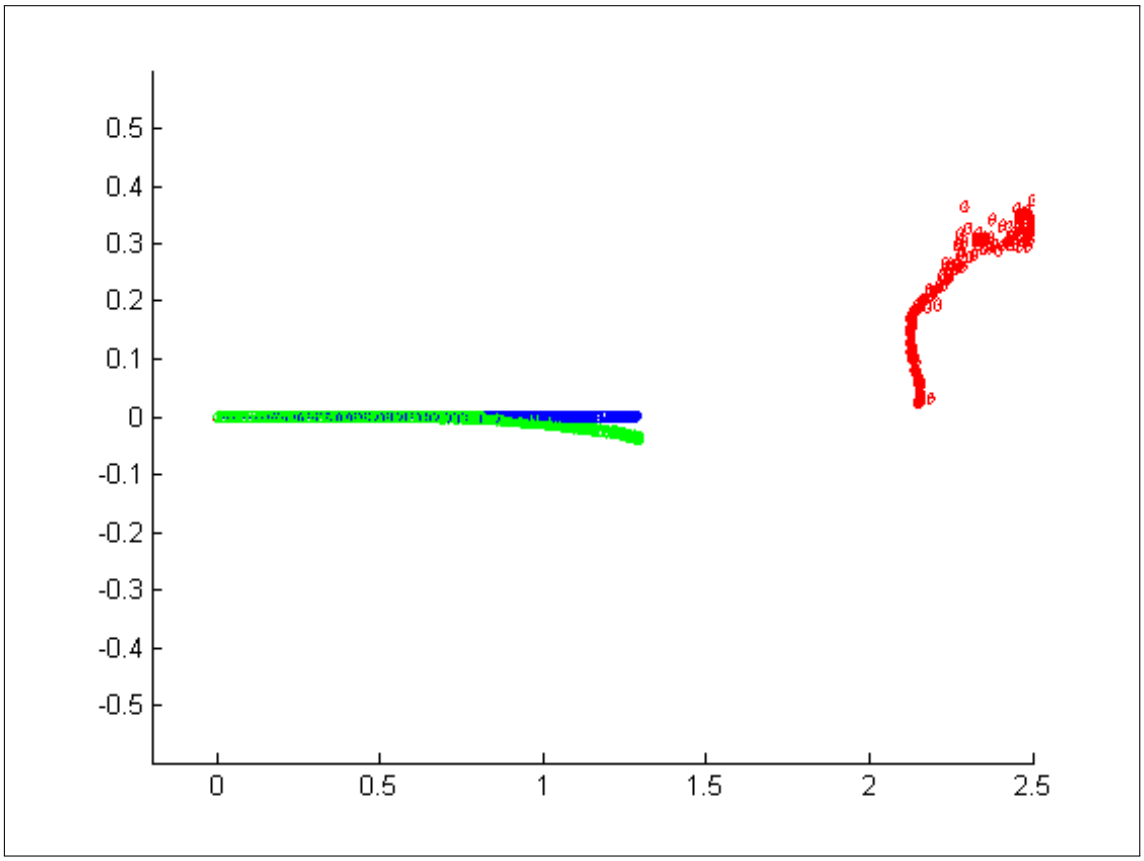

Figure 8.5: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

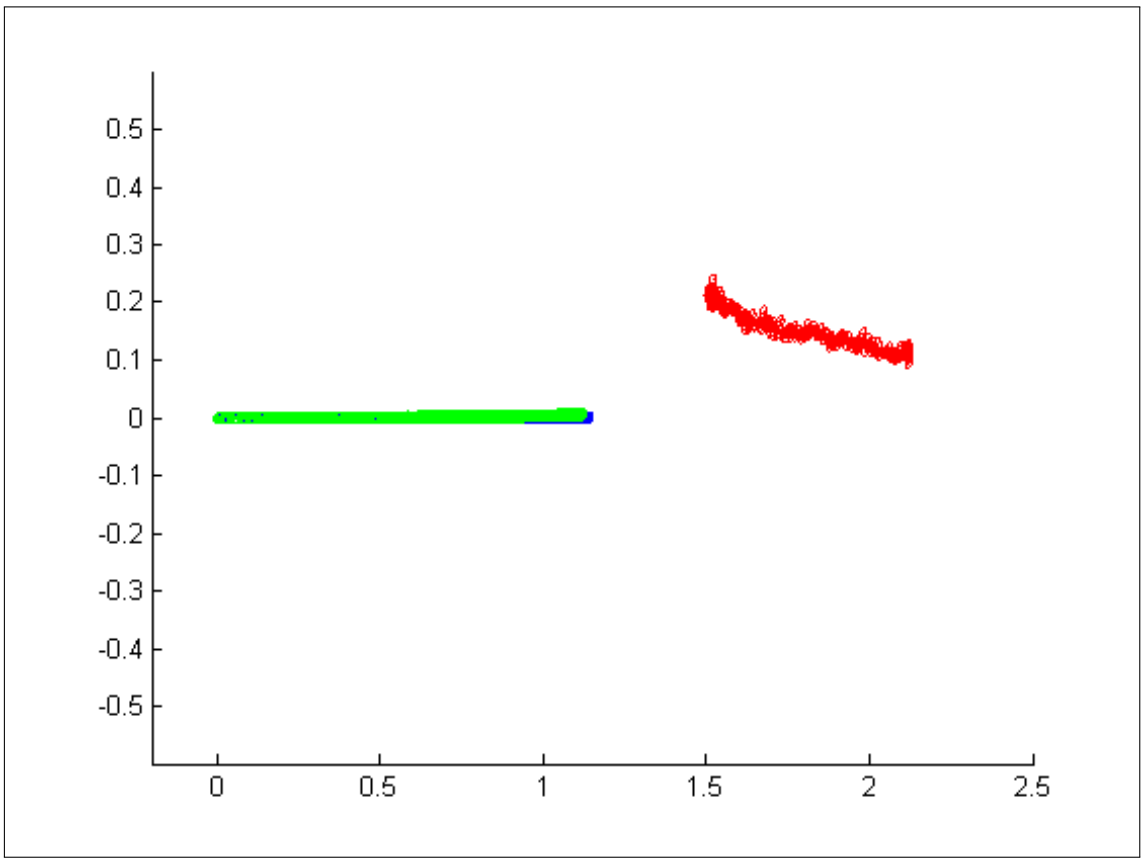

Figure 8.6: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

Chapter 9

Discussion

9.1 Image Processing Algorithm

The designed markers provided the image identification algorithm with good data sets to process. The real time identification algorithm performed with the same accuracy, within a 5% bound, as the buffered preprocessing investigation's results even with the shaking of the Kinect due to the robot's motion. It is therefore reasonable to state the other investigated preprocessing method, using the Canny Edge Detector, would perform with the same accuracy and so would still not be the optimal choice. The Canny Edge Detector performed with less identification accuracy due to the binary image containing all the background's edges which will contain some circle like objects. These objects would be mistaken as the markers and thus the algorithm would return invalid values thus decreasing the accuracy. The HSV thresholding algorithm removes the entire background except for a certain colour before applying the CHT, this drastically reduces the number of mismatches as the chances of a same coloured circle like object being in the background is relatively low. The optimisation methods improved the computation performance of the final program but since the processor, i7 @ 2.2Ghz, being used was extremely powerful this computation decrease was negligible.

The identification accuracy, and thus measurement data, mainly effected the estimated relative coordinate state vector as the state could only be estimated through a prediction using the dynamic model. So when measurement data is not available for a while, the state will become inaccurate and even when measurement data becomes available again it will take time to reconverge to the true state value. As the leader's

pose state is updated through the estimated relative coordinate state, the belief of the leader's pose will also be effected. The consequences for the estimated follower's pose state is not as severe due to the odometry measurements still providing some feedback on what happened in the environment.

9.2 State Estimators

9.2.1 Follower' Pose Estimate

Two state estimators were investigated and designed to estimate the follower's pose amidst noisy measurements namely, the EKF and UKF. Both showed extremely good results with different levels of noise covariances in the simulations, figures C.6 to C.11. The EKF and UKF produced very similar results with the low and medium noise levels but the UKF far surpass the EKF with very high levels of noise. The reason is due to the different ways the two estimators handle non-linearities. The EKF linearises the functions into Jacobians and then uses them to parse the covariance matrices through, so if the covariance matrix's values are large the result of the linearisation will be very inaccurate. The UKF instead parses chosen sigma points through the non-linear functions and reconstructs the covariance matrix on the other side through weights so large variance values passed through are not so inaccurate. The simulation results of the two estimators are expected as the literature [66] points to the UKF outperforming the EKF in the presence of high noise levels.

As most formation control studies [43][59], which consider state estimators, use the EKF and since the system's variances were lower than the low noise level in the simulations, the EKF was chosen as the state estimator in the final implemented system. The state transition and state measurement noise model for the follower's pose was shown not to be explicitly Gaussian distributed but with more samples it could tend towards being Gaussian. Even if the noise is not Gaussian given just the variance the EKF could still be used to estimate the state. The EKF estimate of the follower's pose was extremely accurate to within the cm range in all states. The results show using just the odometry pose data would of caused errors to grow in all states over the course of the robots movements. This is expected due to the nature of of using only one noisy measurement to estimate the state. The comparison between the estimated pose and odometry pose differed significantly more when the

robot travelled in the y direction with inaccuracies in the odometry pose of around 0.2m. This result was more than expected but the outcome was still expected due to how errors propagate during motion, if the robot ideally moves along one axis the error in the other axis will grow more. The accuracy of the estimated state provides a backwards proof to show the noise models used must resemble a Gaussian distribution.

9.2.2 Relative Coordinate State's Estimate

The estimate of the relative coordinate state was accurate when the follower started out with a zero polar angle error, see figures D.1 and D.11. However, when the follower started with a non zero polar angle error the estimate of the relative coordinate state was extremely in accurate, see figures D.6 and D.16. This result was not at all expected as the simulations showed the EKF able to estimate the relative coordinate state and thus the leader's pose accurately up to the medium noise level. Since the medium noise model is far more than the state transition and state measurement noise model derived in the *System Modelling* section, it cannot be the variance value of the noise. A likely explanation for this result, is as the state transition model is a function of the follower's and leader's velocities, the accuracy of the predicted state in the EKF is dependent on the estimation of the follower's and leaders velocities. Since the follower's pose estimation was accurate so will be its estimated velocity, meaning since the leader's pose is measured through visual data only any difference between measurements will result in a estimated velocity for the leader. So even if the leader is stationary, the estimator will produce changes in the leader's pose and over time the error will grow.

9.3 Formation Control

The chosen controller from study [47] was shown to drive the errors towards zero through a Lyapunov function by manipulating the gain values. The simulations of just the controller showed the formations are kept when the leader's trajectory is linear or circular. These simulations showed how the manipulation of the gain constants effected the final error bound and the error convergence rate, figures C.2 and C.4, as well as the velocity command inputs to the iRobot, figures C.3 and C.5. The gain effects on the final error bound and convergence rate was expected from the proof but the initial velocities show too much increase in gain would surpass the

velocity magnitude constraints of the iRobot's wheels.

The simulated controller and state estimators, figures 7.3 and 7.4, finally show how the entire system can perform complex formation control in the presence of low and high levels of noise. Even if the estimate of the relative coordinate state is inaccurate the controller will still drive the states towards set desired relative coordinates. The controller performed as expected in this aspect as all the data set's relative coordinates are driven towards the desired values within some arbitrary bound determined by the gain sets used. The Kinect's FOV prevented the testing of desired polar angles greater than roughly 30 degrees as the leader would be seen in the RGB image but not in the depth image.

9.4 Comparison to literature results

Since this study uses a specific type of controller to achieve formation control, the results of the controller can therefore be compared to other types in literature. Due to the limitations imposed on this study only some of the implementation results can be compared the rest will be from the simulation section. The final error bound in the relative states from study [35] is lower than that of this study's. However, since they linearised the system the ability to switch between different leader trajectories was extremely poor compared to simulated results of this study. The ability of study [42] to switch formations was extremely impressive compared to the simulations of this study, as the use of MPC allowed a better switching follower trajectory where the error will not exceed the maximum as it sometimes would in this study.

9.5 Final System

9.5.1 Fulfilment of primary objectives

The selected controller was shown to achieve complex formation control in simulations where the states are estimated with the modelled noise from the complete modelling of the system. The feature identification and measurement algorithms could locate and measure the leader and landmarks with accuracy determined only by the sensor's constraints. The localisation algorithms were able to accurately localise the follower and the leader up to a certain point determined by the initial

starting conditions of the system. The final designed system when implemented is able to maintain a formation from a both a zero starting error as well as starting error in the polar coordinates of the relative coordinate state. The ability to test a desired relative polar angle greater than 30° was constrained by the Kinect's FOV but as it worked for an angle of 10° it should be able to achieve greater desired angles with a different sensor set-up.

9.5.2 Fulfilment of secondary objectives

The design of the simple yet robust markers for the identification algorithms allowed the algorithms to not be hindered by the point of view of the follower. The methods of optimisation allowed the computation burden of the algorithms to slightly decrease.

Chapter 10

Conclusions

This study systematically approached the leader-follower formation control of two iRobotCreates through the use of visual feedback to produce a design which allows the desired specifications to be achieve. The image processing side of the study was not the main aspect so instead of a complicated feature detecting algorithm, a simple yet robust marker based algorithm was designed. The study had several quite severe limitations imposed from the start but these were overcome by first simulating the entire system to show extremely adequate results and then following up with the physical implementation. Even with the imposed physical limitations the final implemented design performed adequately well in relation to the scope of the project's objectives.

10.1 Primary Objectives

The investigation and thus design of the feature identification adequately provided a identification rate of around 90% for both the leader and landmarks. The measurement equations produced state measurements whose accuracy is only determined by the accuracy of the sensors.

The iRobot's type of motion was sufficiently analysed so as to better model the system and thus improve the control design. The proposed noise models for the follower's pose state and the relative coordinate state estimation were adequately modelled and the follower's pose noise models were validated by the final implemented results.

The simulated system performed perfectly with the modelled system noise and was even shown to still operate in amidst higher levels of noise than that of the physical

system.
The localisation algorithm produced accurate state estimates of the follower's pose in all testing scenarios. The localisation algorithm for the relative coordinates produced adequate state estimates of the leader's pose only in certain testing scenarios. The selected controller performed as expected from the design in both the simulations and implementation and so was an adequate controller to meet the specifications of the system.

10.2 Secondary Objectives

The markers designed and used provided the necessary robustness for accuracy of their identification to only depend on the image processing algorithms. The optimisation of the algorithms illustrated ways of how to implement the algorithms on systems constrained by processing speeds.

Appendix A

Software Design

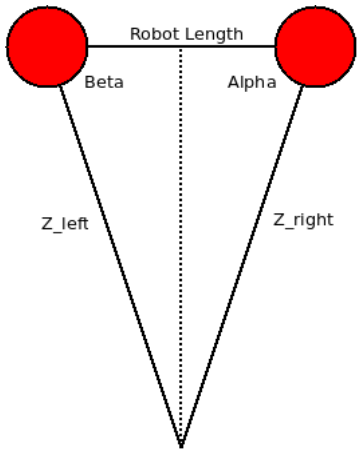

Figure A.1: Cosine Rule

$$\alpha = \arccos \frac{Robot_{length}^2 + Z_{right}^2 - Z_{left}^2}{2(Robot_{length})(Z_{right})}$$
$$\beta = \arccos \frac{Robot_{length}^2 + Z_{left}^2 - Z_{right}^2}{2(Robot_{length})(Z_{left})}$$

Appendix B

Control

B.1 Motion Model Theory

The following explanations are based on the *Computational Principles of Mobile Robotics* by Dudek and Jenkin [67] and the author's (UCT)MEC2023F Dynamics I notes.

Velocity-based
The iRobotCreate robotic platform is part of a common wheeled robot class which ideally moves through 2D space with what is known as Differential Drive Kinematics (DDK). The velocity-based motion model is centered around these ideal kinematics. Most ground vehicles use a drive mechanism known as differential drive, where two drive wheels are mounted on a common axis and each wheel can be rotated independently. When the common axis is rotated the wheels will be rotated about a point which lies along the axis, this point is known as the Instantaneous Center of Curvature (ICC), see figure B.1.

So therefore by varying the velocities of the two wheels we can vary the trajectory the robot takes. Since the rate of rotation ω about the ICC is the same for both wheels, we can write:

$$\omega(R + \frac{l}{2}) = V_r \tag{B.1}$$

$$\omega(R - \frac{l}{2}) = V_l \tag{B.2}$$

where l is the distance between the wheel centers, V_r and V_l are wheels velocities along the ground and R is the radius from the ICC to the center of the robot. So

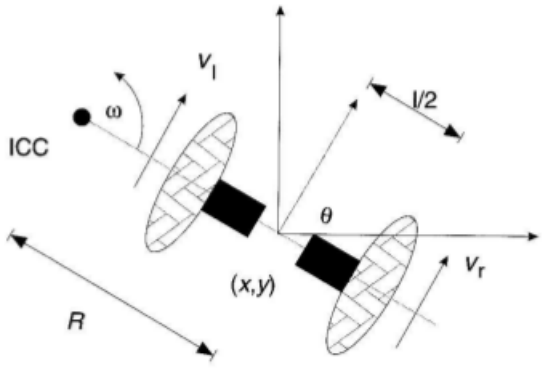

Figure B.1: Instantaneous Center of Curvature [67]

therefore we can at any time write:

$$R = \frac{l}{2}\frac{V_r + V_l}{V_r - V_l} \tag{B.3}$$

$$\omega = \frac{V_r - V_l}{l} \tag{B.4}$$

If V_r=V_l the robot will move with a constant velocity in a straight line. R becomes infinite and ω is zero. If V_r=$-V_l$ then $R = 0$ and the robot rotates around its center with ω. If either $V_r = 0$ or $V_l = 0$ then we will have a rotation around the right or left wheel respectively with $R = \frac{l}{2}$.

The forward kinematics of a differential drive is relatively straight forward. If the current pose (x,y,heading θ) and each wheel's ground velocity is known we can define the following using equation B.3 and B.4:

$$ICC@t = [x - R\sin(\theta), y + R\cos(\theta)] \tag{B.5}$$

Therefor at time $t + \Delta t$ the robot's pose will be:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \cos(\omega\Delta t) & -\sin(\omega\Delta t) & 0 \\ \sin(\omega\Delta t) & \cos(\omega\Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\Delta t \end{bmatrix} \tag{B.6}$$

This equation simply describes the motion of the robot rotating a distance R about is ICC with an angular velocity ω.

The inverse is a lot more tricky as it deals with what commands we should send

in order to move the robot to a certain location. The differential drive imposes non-holonomic constraints on establishing position, meaning it cannot traverse perpendicular to its heading direction. The simplest method to perform only navigation to certain goal pose would be through a series of rotations about its center and linear traversing. However this formation control application will not require it to perform rapid manoeuvres which the non-holonomic constraints would prevent but rather follow the reference trajectory i.e. the leader. The iRobotCreate uses serial communications at a predefined baud rate via operational codes. The op code, 137, to actuate the wheels takes two 16-bit signed values using two's complement. The first two bytes specify the average velocity of the drive wheels in mm/s, the next two bytes specify the radius in millimeters at which Create will turn. This command data will allow the iRobotCreate to perform its differential drive kinematics or velocity-based motion model. The longer the radii the more straight the Create will drive, while shorter radii make the Create turn more. The radius is measured from the center of the turning circle (ICC) to the center of Create. A Drive command with a positive velocity and a positive radius makes the Create drive forward while turning toward the left and vice versa. Special cases for the radius make the Create turn in place or drive straight [68]. ROS abstracts this communication, it converts the user specified ROS Geometry_msg Twist velocity command into the right format for op code 137. The average velocity op code parameter is the specified $Twist.linear.x$ where as the radius is described by:

$$R = \frac{Twist.linear.x}{Twist.angular.z} \qquad \frac{V_r + V_l}{2} = Twist.linear.x \tag{B.7}$$

However to simplify the steps from the user specified commands through ROS to the 137 op code and finally to the DDK motion, the motion model in terms of the applied ROS Geometry_msg Twist velocity command using the sets of equations above becomes:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} -R\sin\theta + R\sin\left(\theta + Twist.angular.z\Delta T\right) \\ R\cos\theta - R\cos\left(\theta + Twist.angular.z\Delta T\right) \\ Twist.angular.z\Delta T \end{bmatrix} \tag{B.8}$$

Odometry-based

There are a number of reasons for motion errors such as bumps, different wheel diameters etc. so this is where odometry comes in. An Odometry-based motion model calculates the present pose through the use of wheel encoders. A wheel encoder sensor outputs a high when the sensor sees the white strip and low when it

sees the black strip. There are two popular types of encoders namely, incremental B.2 and absolute B.3. The difference being with incremental encoders, you can measure only changes in position (from which you can determine velocity and acceleration), but it is not possible to determine the absolute position of an object. The absolute encoder is capable of determining the absolute position of an object due to each ring of the absolute encoder having double the number of segments of the prior ring, the values form numbers for a binary counting system. Encoders allow the robot to keep track of how much each wheel has rotated over a certain time interval.

Figure B.2: Wheel Encoder [67]

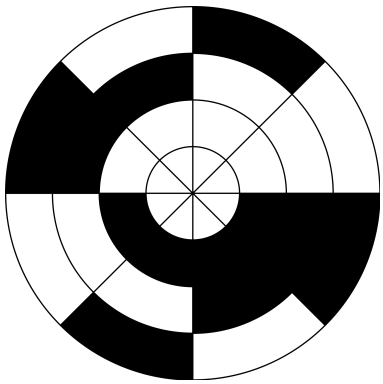

Figure B.3: Absolute Wheel Encoder [67]

The iRobotCreate sensors can be queried through the right op codes to access all the different sensor readings. Through packet ID 19 and 20 the distance and angle change from the last query can be accessed. The iRobotCreate produces these values through measuring the distance travelled by each wheel through the encoder and using the DDK motion equations produces the change in pose. ROS abstracts this information and publishes it in the /odom topic where the current pose is updated. There are a couple of odometry models such as [69] where each iteration of extracted odometry data is a 3 step transformation between x_t and x_{t-1} where by δ_{rot1} is the initial turn, δ_{trans} is the linear translation and δ_{rot1} is the final turn in the time interval. With this different noise models can be used but since ROS abstracts the odometry data it is sufficient to just use the change in pose from the /odom topic for the odometry data.

B.2 Sensor Model Theory

The type of sensor is classified as a range bearing type whereby the measurements are in the form of the radius and angle to the landmark from the point of the follower's frame, shown below:

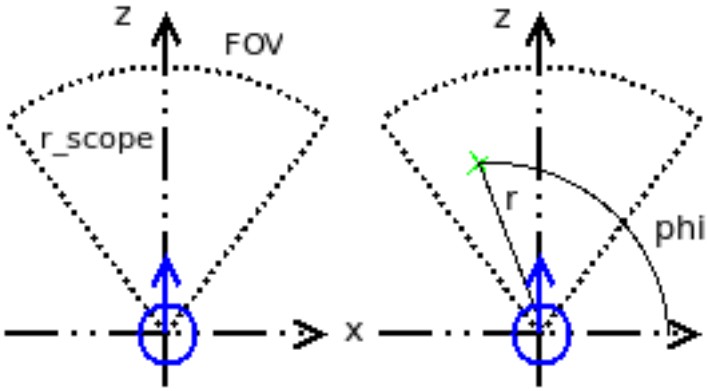

Figure B.4: Sensor Model

$$z = \begin{bmatrix} r \\ \phi \end{bmatrix}$$

The general sensor limitations would be the maximum and minimum range plus what its field of view is. The robot only measures the landmark in its reference fram and so to compare it to the stored position of the landmark in the global reference frame, it must undergo a transformation. The transformation is simply a rotation matrix and is defined as follows:

$$z = \begin{bmatrix} x_{global} \\ y_{global} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix}$$

B.3 Actuator and Sensor Noise Model

There are two different states to be estimated with two different state transition models namely, follower's motion model and the change in the relative coordinates as well as two different state measurement models for the follower's pose and relative coordinates. These covariance matrices are determined and outlined in the following sections.

B.3.1 State Transition Measurement

Follower's Pose (Motion Model)

The iRobotCreate will be "step tested" to find out the probability distribution of the robot measuring the change in its odometry from the ROS /odom topic compared to what the true change in odometry data is. The true odometry data will be measured

with ruler/protractor over each iteration of the robot moving. Comparing the true measurement and the odometry measurement will result in an error which will be plotted for each measured pose state to determine the properties of its probability distribution function. (p.d.f)

Due to how the x and y positions of the pose change depending on the heading direction at the time, the error in the odometry measurement for the change in x and y will depend on which way the robot is heading. The proposed noise model assumes that there is no relation between the two errors meaning the error measurement of both is the same regardless of the heading direction, thus the error in measuring the change in positions x and y will be the same. Since the error in measuring the change in angle is just related to the angular velocity it will have its own p.d.f.

To determine the error in measuring state x and thus y the iRobotCreate will receive velocity commands $[V = 0.1, \omega = 0]$ for 0.1 seconds by publishing to the /navi topic. After each step the measured value will be recorded against the /odom topic's value and the error will be stored. The same will be done for determining the angular measuring error except the velocity command will $[V = 0, \omega = 1]$ again for 0.1 seconds. The experimental set-up is shown in figures B.5 to B.7, the resulting histograms with the means and standard deviations for the position and angular errors are shown in figures B.8 and B.9 respectively.
The covariance matrix is thus defined as follows:

$$R_t = \begin{bmatrix} 0.0064^2 & 0 & 0 \\ 0 & 0.0064^2 & 0 \\ 0 & 0 & 0.035^2 \end{bmatrix}$$

Figure B.5: Start position of the step tests

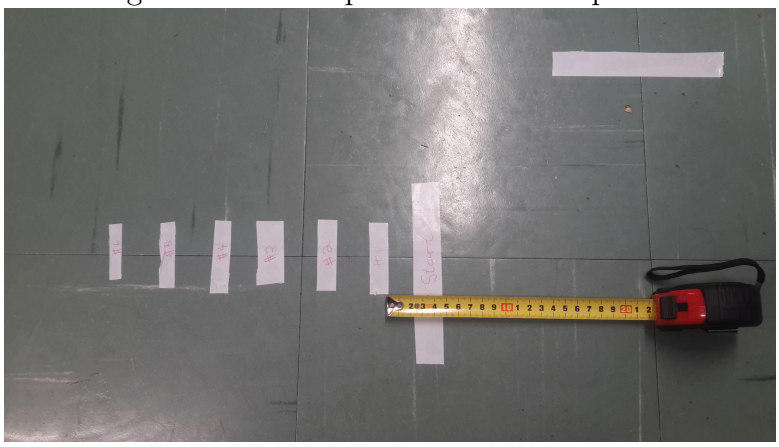

Figure B.6: Measuring the data

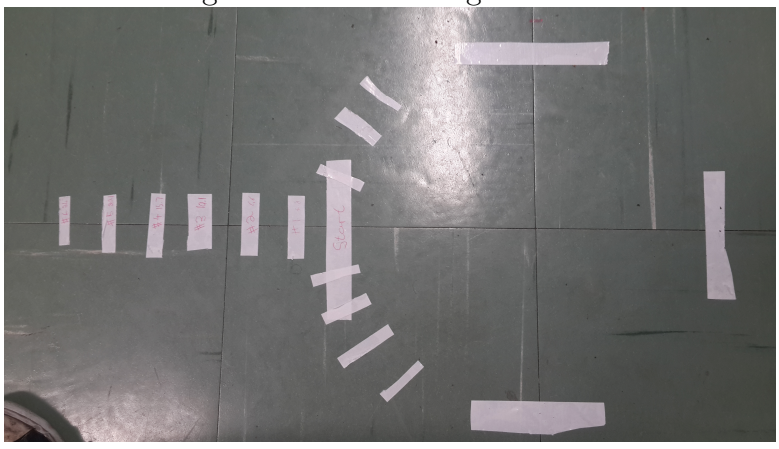

Figure B.7: Markers after step tests

Bin	Frequency
-0.004	4
-0.003	3
-0.002	5
-0.001	3
0	0
0.001	3
0.002	5
0.003	1
0.004	2
More	2
Sigma	0.0064 m
Mean	-0.00157 m

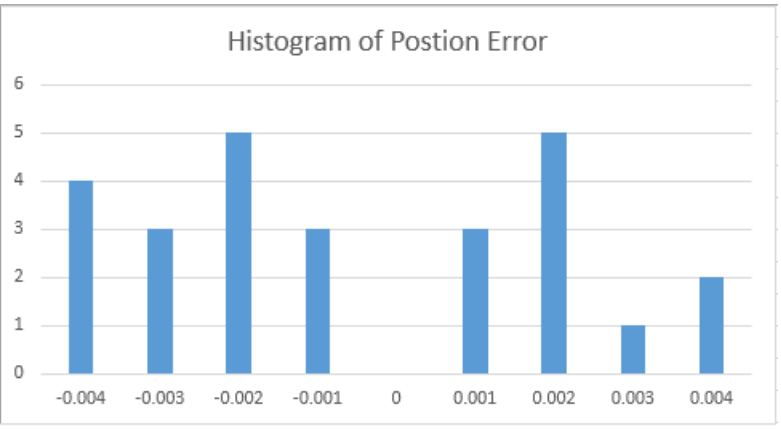

Figure B.8: Error for change in position state

Bin	*Frequency*
-0.05	0
-0.04	3
-0.03	2
-0.02	1
-0.01	2
0	3
0.01	0
0.02	1
0.03	3
0.04	0
0.05	2
More	3
Sigma	0.035
Mean	0.01

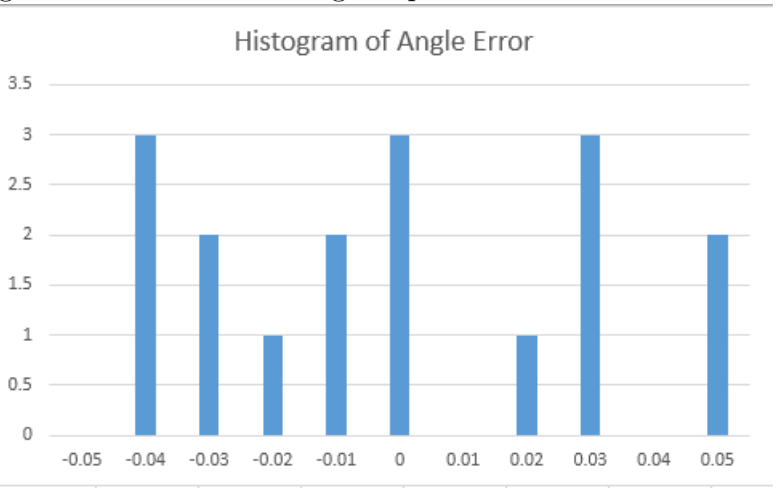

Figure B.9: Error for change in angular state

Relative Coordinates' Noise Model

Since the change in the relative coordinates, see *System Modelling* section, is a function of the velocities of the follower and leader which are estimated, the transition of the relative coordinate state's error will also be a function of the error in the estimated velocities. The limitation of not having an aerial webcam to monitor the robots poses and thus the relative coordinates makes it very difficult to determine the relative coordinate state's transition covariance matrix. Reasonable values were chosen at first and then tweaked with the final design's estimated covariance matrix from the EKF after the program had run, the final "optimal" covariance matrix is as follows:

$$R_t = \begin{bmatrix} 0.0078^2 & 0 & 0 \\ 0 & 0.0054^2 & 0 \\ 0 & 0 & 0.048^2 \end{bmatrix}$$

B.3.2 Vision Measurement of State

Follower's Pose Vision Measurement Noise Model

The noise model used in this thesis for the Kinect comes from study [70]. They determined the covariance matrix for the measurement error in 3D data provided by the Kinect i.e. pixel positions [x,y] and depth to that particular pixel. Since the sensor model is a range/bearing type where the range is measured directly from the depth image and the bearing is calculated by the offset in the x direction pixels with a real world proportional ratio, see *Image Processing* section. Only the standard deviations for the depth and pixel position x error will be used for the sensor's covariance matrix. These values would be different to if step like tests were performed on this particular Kinect but due to the limit on time and the fact that both the study's Kinect and this thesis's Kinect were calibrated the results should be relatively similar. I note that the bearing angle is only a function of the x pixel position but it should provide a reasonable covariance matrix and is as follows:

$$Q_t = \begin{bmatrix} 0.033^2 & 0 \\ 0 & 0.028^2 \end{bmatrix}$$

Relative Coordinates' Vision Measurement Noise Model

Since the relative coordinates is just the range and bearing measured value plus the difference in heading. The covariance values used for measuring the landmarks to update the follower's pose can be used for the range and bearing measurement for the relative coordinates. The third state γ is calculated in a similar way and so the same variance value used for the bearing */phi* will be used. The noise covariance matrix is thus:

$$Q_t = \begin{bmatrix} 0.033^2 & 0 & 0 \\ 0 & 0.028^2 & 0 \\ 0 & 0 & 0.028^2 \end{bmatrix}$$

B.4 Controller Proof

The study [47] only provides a short proof to validate the chosen control laws as it is a IEEE paper with a page limit. This appendix provides a much more indepth explanation and derivation for the control laws used so as to gain a better understanding of the control system. I note that there is an error in the study in the proof section where they miss print the error value $|\epsilon_1|$ as $|q_1|$ where $|q_1|$ is the relative coordinate ρ. However, this the controller is still valid with the following proof.

A quadratic Lyapunov function candidate is proposed:

$$V = \frac{1}{2}\epsilon^T P \epsilon \tag{B.9}$$

where P is the 3x3 matrix representation of a symmetric positive definite transformation with entries are zero except for the diagonal and p_{12} and p_{21}. Where $p_{12} = p_{22}\frac{t_2}{q_1}$, $t_2 = \tan q_2$.

$$\begin{bmatrix} p_{11} & p_{22}\frac{t_2}{q_1} & 0 \\ p_{22}\frac{t_2}{q_1} & p_{22} & 0 \\ 0 & 0 & p_{33} \end{bmatrix} \tag{B.10}$$

P is only positive definite if and only if its diagonal entries are positive and condition:

$$p_{11} - p_{22}(\frac{t_2}{q_1})^2 > 0 \tag{B.11}$$

which is derived from Sylvester's positive definite matrix proof. Since it is a 3x3 hermitian matrix there will be 3 steps (i.e 3 sub matrix determinant conditions)
First: $p_{11} > 0$
Second: $p_{11}p_{22} - p_{22}^2(\frac{t_2}{q_1})^2 > 0$ simplifies to $p_{11} - p_{22}(\frac{t_2}{q_1})^2 > 0$ which is B.11
Third: $p_{22}p_{33}[p_{11} - p_{22}(\frac{t_2}{q_1})^2] > 0$ which just restates p_{22} and p_{33} must be > 0
Equation B.11 is guaranteed to hold by the 5.6 in the *Controller Selection* section. Through differentiating B.9 along the trajectories of the ϵ vector using equation 5.2 from the *System Modelling* section. Then collecting the v_L, ω_L, v_F and ω_F terms produces the differential potential energy function B.13. The beginning of this derivation is shown in B.12 but it is left out as it is just a matter of standard differentiating and grouping like terms.

$$\delta V = \frac{1}{2}[\delta\epsilon^T P\epsilon + \epsilon^T \delta P \epsilon + \epsilon^T P \delta\epsilon] \tag{B.12}$$

$$[1] \qquad \delta V = (\Lambda_1\epsilon_1 + \Lambda_2\epsilon_2\epsilon_3)v_f$$

$$
\begin{aligned}
[2] \qquad & -(p_{22}\frac{t_2}{q_1}\epsilon_1 + p_{22}\epsilon_2 + p_{33}\epsilon_3 + \frac{p_{22}}{q_1 c_2^2}\epsilon_1\epsilon_2)\omega_F \\
[3] \qquad & +(\Theta_1\epsilon_1 + \Theta_2\epsilon_2 + \Theta_3\epsilon_1\epsilon_2)v_L \\
[4] \qquad & +p_{33}\epsilon_3\omega_L
\end{aligned}
\tag{B.13}
$$

where:

$$
\begin{aligned}
\Lambda_1 &= (-p_{11}c_2 + p_{22}\frac{s_2^2}{q_1^2 c_2}) \\
\Lambda_2 &= \frac{p_{22}}{q_1^2}(\frac{s_2}{c_2^2} + s_2) \\
\Theta_1 &= (p_{11}c_2 + p_{22}\frac{t_2 s_{32}}{q_1^2}) \\
\Theta_2 &= \frac{p_{22}}{q_1}\frac{\sin q_3}{cosq_2} \\
\Theta_3 &= \frac{p_{22}}{q_1^2}(\frac{s_{32}}{c_2^2} - t_2 c_{32})
\end{aligned}
\tag{B.14}
$$

With the control laws selected 5.4 and 5.5 the first two terms become negative semi-definite. The condition 5.6 is necessary so that the only null space of v is $\epsilon = 0$ this will ensure $\epsilon_1 \to 0$ as $t \to \infty$. The condition is derived as follows:

$$From\ [1]\ \Lambda_1 + \Lambda_2\epsilon_2 \neq 0$$

This implies the LHS must be either > 0 or < 0. This condition can be shown just by substituting in the relevant constants from B.14 and simplifying, shown below:

$$-p_{11}c_2 + p_{22}\frac{s_2^2}{q_1^2 c_2} + \frac{p_{22}}{q_1^2}(\frac{s_2}{c_2^2} + s_2)\epsilon_2$$

They chose that the above equation is > 0 so as to satisfy the Sylvester's condition refSly as well hence the 5.6 condition satisfies both.

- Discussion One

  The [1] term in B.13 can dominate the forcing terms [3] and [4] with a large enough k_1 if $|\epsilon_1| \geq \varepsilon_1$ for all $\varepsilon > 0$ so that $\delta V < 0$, remember ϵ is the error state used in the Lyapunov function. Given $k_1 > 0$ there will be an $\varepsilon > 0$ such that if $\epsilon < \varepsilon$, the forcing terms cannot be dominated by [1]. In this case, the second term [2] can be used unless by the same logic above there exists a $k_2 > 0$ such that $p_{22}\epsilon_2 + p_{33}\epsilon_3 < \varepsilon_2$ for some ε_2 such that $-k_2\varepsilon_2^2$, like ε_1, is also very close to zero.

  This discussion implies the states ϵ tend to a neighbourhood set N where

$\epsilon_1 = 0$ and $p_{22}\epsilon_2 + p_{33}\epsilon_3 = 0$, which when thinking back to the non-holonomic constraints makes sense.

- Discussion Two
  If the leader is travelling in a straight line $\omega_L = 0$ then since $\epsilon_1 < \varepsilon_1$ is the only term in [3] that could be large is $\Theta_2\epsilon_2$. Θ_2 is an odd function (sine in numerator) of ϵ_3 with $\Theta_2 < 0$ if $\epsilon_3 < 0$. This means $\epsilon_3\Theta_2 > 0$ for all ϵ_3. Now in the set N we have $\epsilon_2 = -\frac{p_{33}}{p_{22}}\epsilon_3$ since p_{22} and p_{33} are positive constants, we have $\Theta_2\epsilon_2 < 0$ thus $\delta V < 0$ (negative definite) when $\omega_L = 0$.

- Discussion Three
  If however, $\omega_L \neq= 0$ the fourth term [4] can be positive. This is over come through the negative definite function $\Theta_2\epsilon_2$, remember this relation $\epsilon_2 = -\frac{p_{33}}{p_{22}}\epsilon_3$. So two conditions can overcome this positive term [4], (1) $\frac{p_{33}}{p_{22}} > 1$ or (2) $\frac{p_{33}}{p_{22}} < 1$ and $\frac{|\omega_L|}{v_L} < \frac{1}{q_1}$ then there exists q_3 s.t

$$|\frac{\sin q_3}{\cos q_2}| > q_1\frac{\omega_L}{v_L} = \kappa q_1$$

  This shows the need for conditions 5.7 or 5.8. Condition (1) is used for when the leader is performing basic manoeuvres however when the leader is performing more complex motion where it is not just straight or in a circle. The bounds on the states can be tightened by going through the same above procedure and using the supremum of the complex curvature motion.

B.5 Kalman Filtering

So what is a Kalman filter and what can it do. The actual Kalman filter is an optimal recursive estimator i.e it infers parameters of interest from indirect, inaccurate and uncertain observations. It is optimal due to it minimising the mean square error of the estimated parameters if all noise is Gaussian and is recursive in the sense it only uses the previous estimate to update with the current measurements. Even if the noise is not Gaussian, given only the mean and standard deviation of the noise the Kalman filter is still the best linear estimator however other estimators such as particle filters might be better.

Probabilistic Foundation

In any autonomous system decisions are often based probabilities. So if the probability of the outcome of a decision or in our case motion is known there are certain facts that the system can make about what happened after the motion, this type of approach is known as a Bayesian network. The goal of state estimation is to estimate the state x of a system given observations z and control u. The following equations show how a belief of a state can be found through probabilities, the following is the recursive Bayes filter:

$$bel(x_t) = p(x_t|z_t, u_t) \qquad This\ is\ the\ definition\ of\ the\ belief$$

$$bel(x_t) = \eta p(z_t|x_t, z_{t-1}, u_t)p(x_t|z_{t-1}, u_t) \qquad Baye's\ Rule/Theorem$$

$$bel(x_t) = \eta p(z_t|x_t)p(x_t|z_{t-1}, u_t) \qquad Markov\ assumption$$

The Markov assumption is simply that we define that the transition from one state to the next is not dependant on the previous states. This property is assumes the system is "memoryless", meaning the p.d.f of the next state only relies on the current state not the previous sequence of events leading to that transition.

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, z_{t-1}, u_t)p(x_{t-1}|z_{t-1}, u_t)\delta x_{t-1} \qquad Law\ of\ total\ probability$$

The law of total probability is the fundamental rule for relating marginal probabilities to conditional probabilities where the total probability is just the sum of the individual events.

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|z_{t-1}, u_t)p(x_{t-1}|z_{t-1}, u_{t-1})\delta x_{t-1} \qquad Markov\ assumption$$

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|z_{t-1}, u_t)bel(x^{t-1})\delta x_{t-1} \qquad recursive\ nature$$

This means the Bayes filter can be shown to have two steps namely, a prediction B.15 and then a correction B.16.

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1}) \tag{B.15}$$

$$bel(x_t = \eta p(z_t|x_t)\overline{bel}(x_{t-1}) \tag{B.16}$$

Clearly the motion model is used to predict the state, $p(x_t|u_t, x_{t-1})$, and the measurement model is used to correct the prediction $p(z_t|x_t)$. So the motivation behind

showing the Bayes Filter is to illustrate that the Kalman Filters are just special cases of the Bayes Filter where the p.d.f is a Gaussian distribution.

Gaussian

A Gaussian is just a normally distributed p.d.f with a mean and variance, shown in equation B.17 and figure B.10.

$$p(x)\ N(\mu, \sigma^2) \qquad p(x) = \frac{1}{\sqrt{2\pi\sigma}e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}} \tag{B.17}$$

If there is more than one state it is multivariate function. Where the variance for the univariate case σ^2 becomes a covariance Σ where the dependency for each state on each other state is take into account. For the two state case the co-variance matrix can be modelled as an ellipse where the eigen values of the matrix determine the shape of the ellipse, it would be a sphere for a three state case etc. The bivariate case is shown in figure B.11.

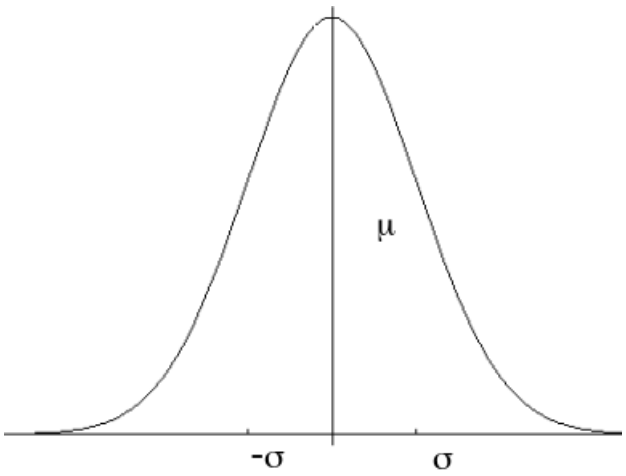

Figure B.10: Wheel Encoder [67]

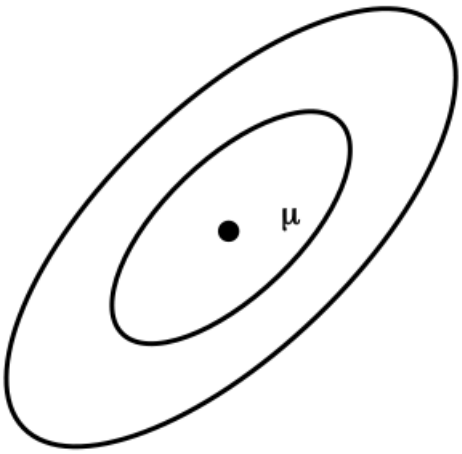

Figure B.11: Absolute Wheel Encoder [67]

When a Gaussian p.d.f is passed through a linear function it retains the Gaussian distribution. When it parses through a non-linear function is becomes distorted according to how hard the non-linearity is and is no longer Gaussian distributed. This is the motivation to linearise a system so as to be able to carry on using the standard linear Kalman Filter.

Kalman Filters

Through understanding the probabilistic foundation the Kalman Filter is simply just a Bayes Filter for Gaussian distributed systems. So the Kalman Filer steps

follow the very same logic where there is a prediction of the next state through the state transition model, our case the motion model, and an update/correction using a measurement to correct the predict state. The Extended Kalman Filter (EKF) is exactly the same as the original Kalman Filter just with a linearised system. Following the Bayes Filter steps the EKF just becomes a matter of laying out the equations, shown below:

$$\boxed{\begin{aligned}
\overline{x_t} &= g(u_t, x_{t-1}) && \text{(B.18a)}\\
\overline{\Sigma_t} &= G_t \Sigma_{t-1} G_t^T + R_t && \text{(B.18b)}\\
K_t &= \overline{\Sigma_t} H_t^T (H_t \overline{\Sigma_t} H_t^T + Q_t)^{-1} && \text{(B.18c)}\\
x_t &= \overline{x_t} + K_t(z_t - h(\overline{x_t})) && \text{(B.18d)}\\
\Sigma_t &= (I - K_t H_t)\overline{\Sigma_t} && \text{(B.18e)}
\end{aligned}}$$

The functions g and h are the motion model and observational model respectively, the functions G and H are the Jacobians of each of the respective models. The matrices R and Q are the noise, modelled from the *Actuator and Sensor Noise Model* section, added for a state transition and measurement respectively.

Another type of Kalman Filter is called the Unscented Kalman Filter (UKF) which attempts to obtain a more accurate linearisation by using multiple samples to calculate the covariance. It uses the unscented transform (UT) which is a method for calculating the statistics of a random variable which undergoes a non-linear transformation. So instead of passing a single point through the action transition function and shaping the covariance, it passes multiple points through the action transition function and calculates the covariance from scratch using the transformed points. A primitive version of this filter might use a large number of samples and calculate the covariance and mean of the posterior distribution by brute force. When all of the distributions involved Gaussian, it is possible to construct the posterior Gaussian using a few specifically selected samples. These deterministically selected samples are referred to as sigma points. These sigma points can completely capture the true mean and covariance of the Gaussian variables, and when propagated through the non-linear system, captures the posterior mean and covariance accurately up to the third order Taylor series expansion.
The sigma point selection is influenced by three parameters:

- α (alpha) determines the spread of the distance of the sample points from the mean of the prior.

- β (beta) determines how much weight is given to the mean when calculating the new Gaussian.
- κ (kappa) is a secondary scaling parameter normally associated with the controlling the spread of the sigma points, usually set to zero

The following equations governs how the sigma points are chosen through the above parameters and how the posterior covariance matrix is reconstructed. The steps for the UT are shown below where μ and Σ are the mean and covariance of the state:

$$
\begin{aligned}
X_0 &= \mu && && \text{(B.19a)}\\
X_i &= \mu + \sqrt{(L+\lambda)\Sigma} && i = 1, ..., L && \text{(B.19b)}\\
X_i &= \mu - \sqrt{(L+\lambda)\Sigma} && i = L+1, ..., 2L && \text{(B.19c)}\\
\lambda &= \alpha^2(L+\kappa) - L && && \text{(B.19d)}\\
w_m^{[0]} &= \frac{\lambda}{L9\lambda} && && \text{(B.19e)}\\
w_c^{[0]} &= w_m^{[0]} + (1 - \alpha^2 + \beta) && && \text{(B.19f)}\\
w_m^{[i]} &= w_c^{[i]} = \frac{1}{2(L+\lambda)} && i = 1, ..., 2L && \text{(B.19g)}
\end{aligned}
$$

The UT parameters are generally set as:

- $\kappa \geq 0$
- $\alpha \in (0, 1]$
- $\beta = 2$ optimal choice for Gaussians

Where X are the chosen sigma points, w_m and w_c are the mean and covariance weights respectively. Steps 2 and 4 of B.19 require the square root of the covariance matrix. There are a couple of ways to accomplish that, the most common way is through the Cholesky decomposition. Where a diagonalising matrix V with the eigen vectors is used to calculate the square root. The chosen sigma points and weights from the UT are then used in the UKF algorithm which is very similar to the EKF steps as it is still based on probability foundation and are as follows:

Prediction (B.20a)

$$\overline{X}_t = g(X_{t-1}, u) \tag{B.20b}$$

$$\overline{\mu_t} = \sum_{i=0}^{2L} w_m^{[i]} \overline{X}_t^{[i]} \tag{B.20c}$$

$$\overline{\Sigma_t} = \sum_{i=0}^{2L} w_c^{[i]} (\overline{X_t^{[i]}} - \overline{\mu_t})(\overline{X_t^{[i]}} - \overline{\mu_t})^T + R_t \tag{B.20d}$$

Correction (B.20e)

$$\overline{Z_t} = h(\overline{X}_t) \tag{B.20f}$$

$$\overline{z_t} = \sum_{i=0}^{2L} w_m^{[i]} \overline{Z}_t^{[i]} \tag{B.20g}$$

$$\overline{S_t} = \sum_{i=0}^{2L} w_c^{[i]} (\overline{Z_t^{[i]}} - \overline{z_t})(\overline{Z_t^{[i]}} - \overline{z_t})^T + Q_t \tag{B.20h}$$

$$\overline{\Sigma_t^{x,z}} = \sum_{i=0}^{2L} w_c^{[i]} (\overline{X_t^{[i]}} - \overline{\mu_t})(\overline{Z_t^{[i]}} - \overline{z_t})^T \tag{B.20i}$$

$$K_t = \overline{\Sigma_t^{x,z}} S_t^{-1} \tag{B.20j}$$

$$\mu_t = \overline{\mu_t} + K_t(z_t - \overline{z_t}) \tag{B.20k}$$

$$\Sigma_t = \overline{\Sigma_t} - K_t S_t K_t^T \tag{B.20l}$$

Appendix C

Simulation Results

Appendix Navigation

Figure C.1 is the Simulink Model

- [1]: Pose updater block
- [2]: Calculate relative coordinates
- [3]: Calculate velocity commands

Figures C.2 to C.3 is the controller only simulations

- red: Leader
- blue: Follower

Figures C.6 to C.11 is the EKF and UKF simulations for the follower's pose only

- blue: Actual Follower's path taken
- green: Estimated Follower's path taken
- black: Actual Follower's path taken
- red: Estimated Follower's path taken

Figures C.12 to C.14 is the EKF simulations for the leader's pose

- black: Actual Follower's path taken
- red: Estimated Follower's path taken

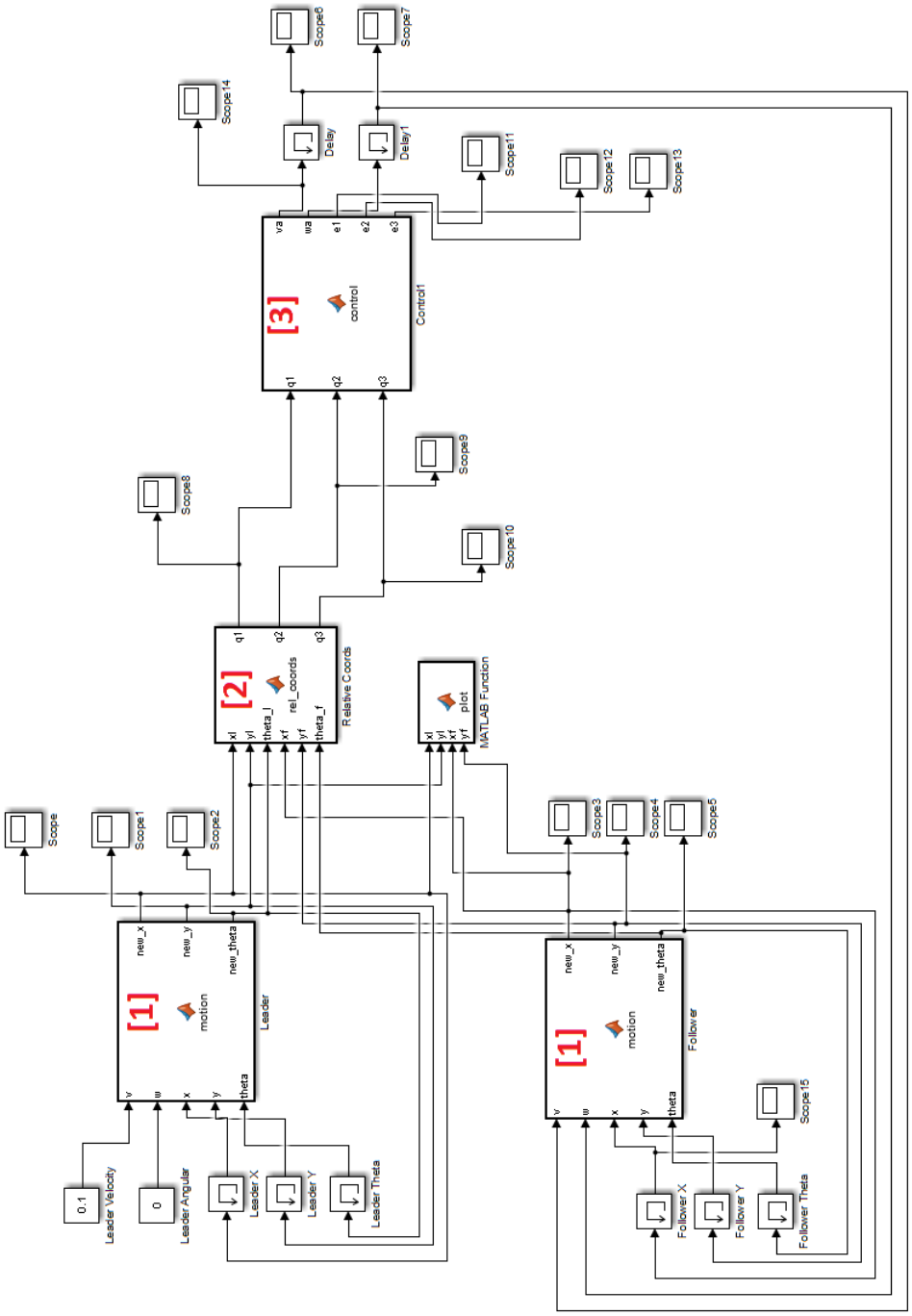

Figure C.1: Controller's Simulink Model

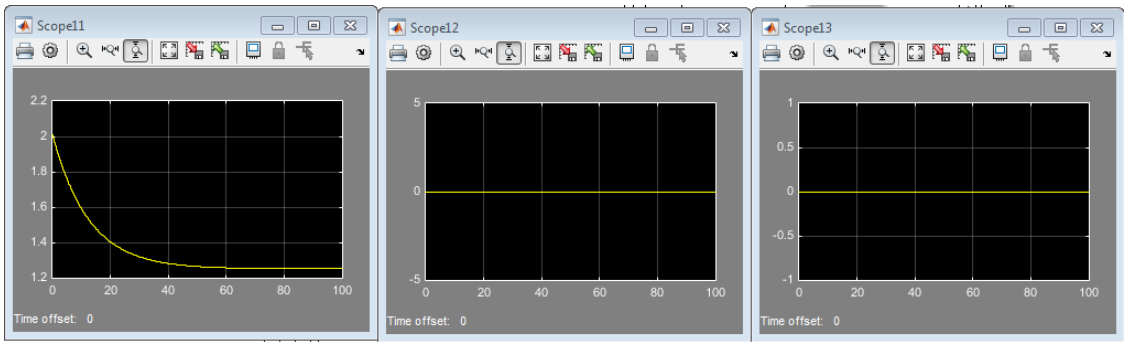

Figure C.2: Linear Leader Velocity:Errors

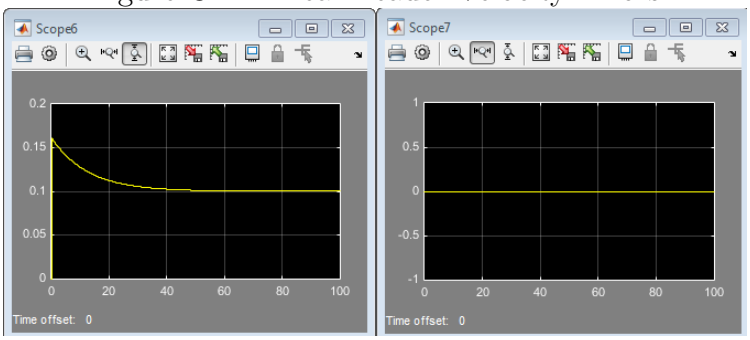

Figure C.3: Linear Leader Velocity:Follower velocities

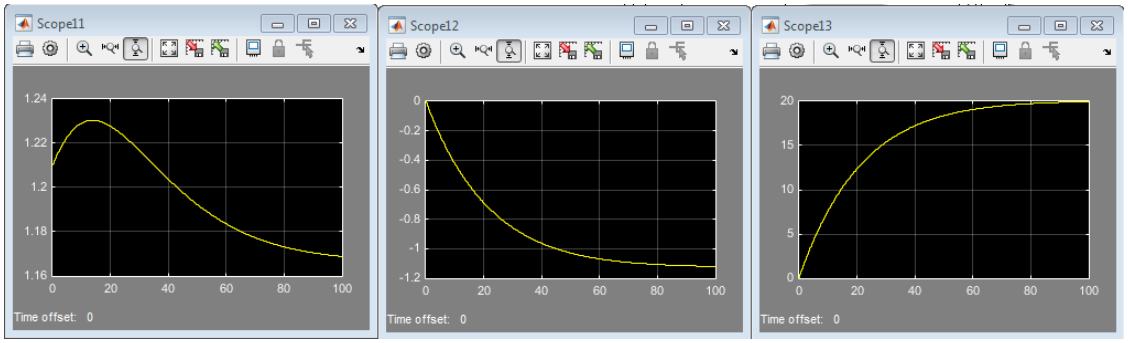

Figure C.4: Circular Leader Trajectory:Errors

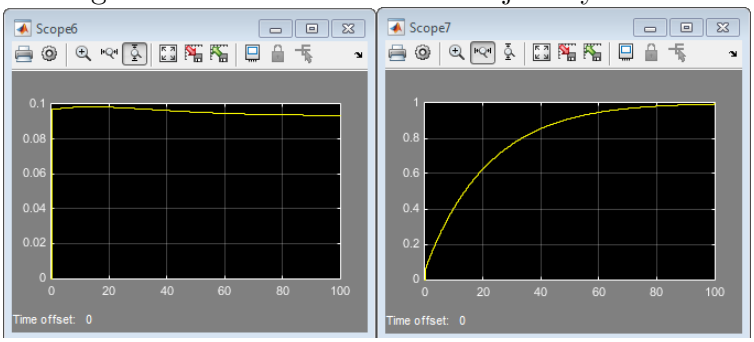

Figure C.5: Circular Leader Trajectory:Follower velocities

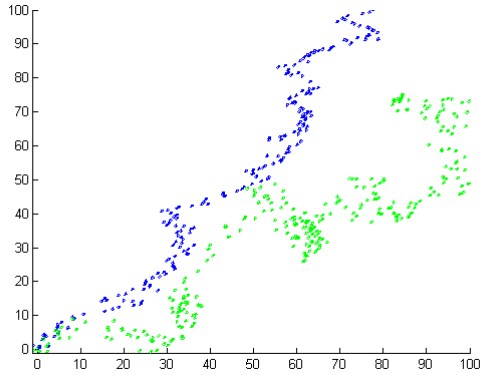

Figure C.6: EKF high noise

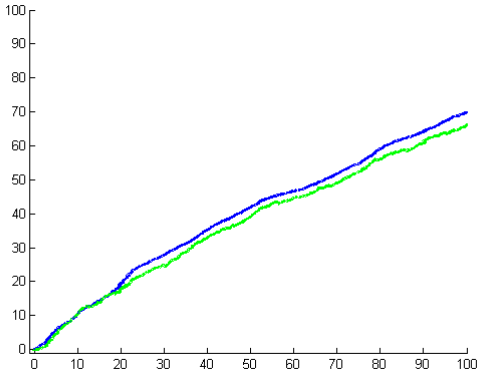

Figure C.7: EKF medium noise

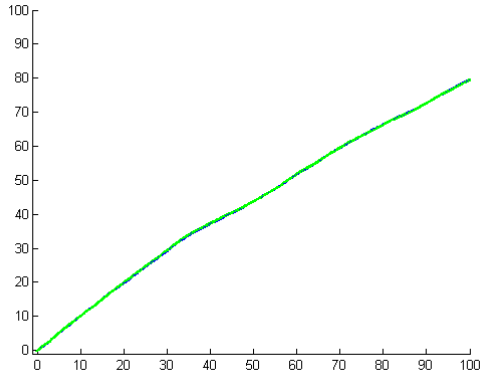

Figure C.8: EKF low noise

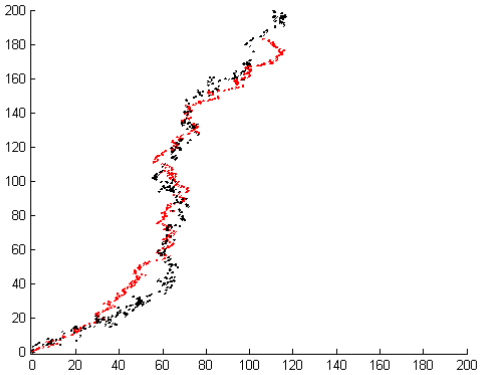

Figure C.9: UKF high noise

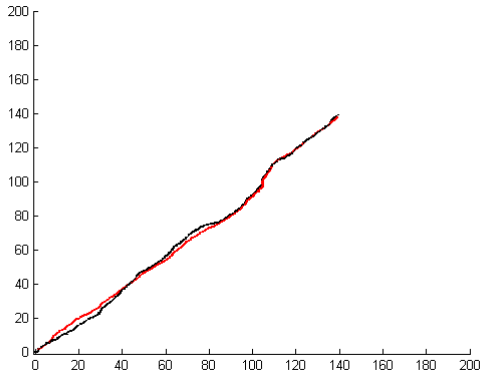

Figure C.10: UKF medium noise

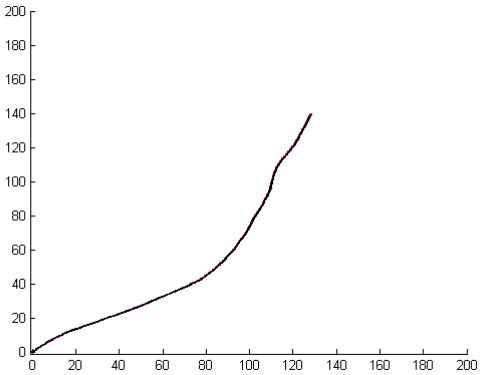

Figure C.11: UKF low noise

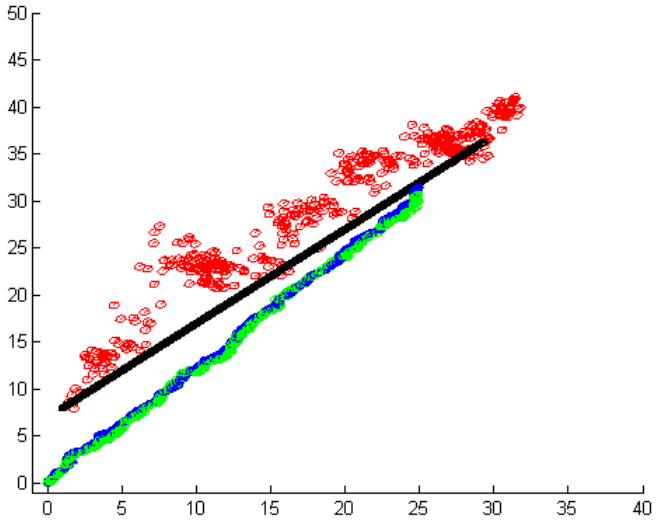

Figure C.12: EKF high noise

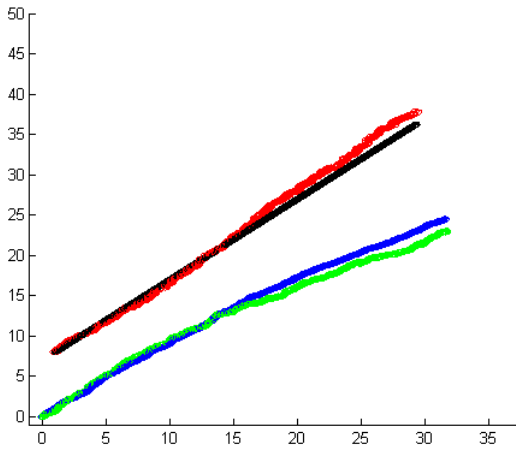

Figure C.13: EKF medium noise

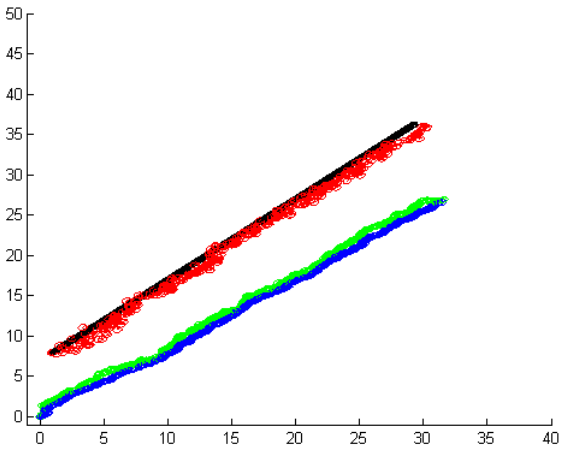

Figure C.14: EKF low noise

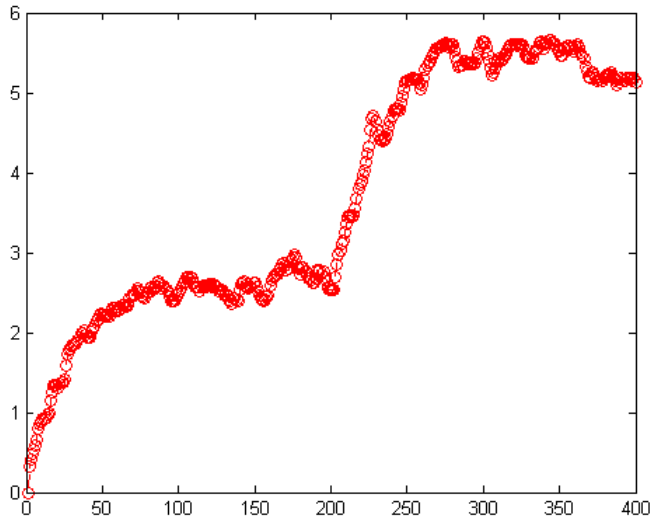

Figure C.15: Error in relative coordinate ρ

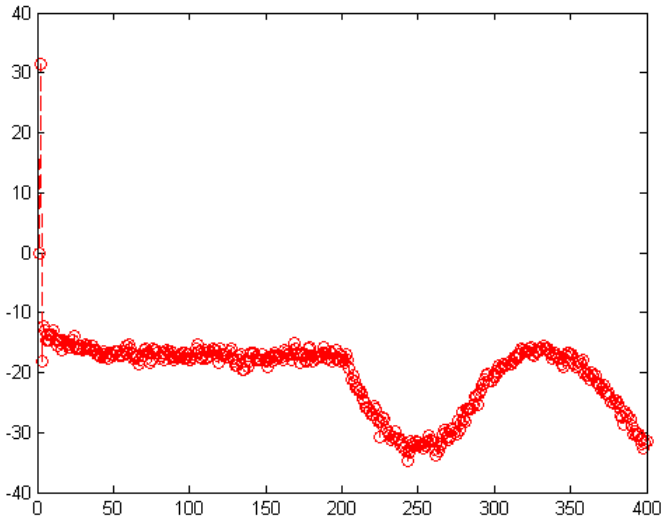

Figure C.16: Error in relative coordinate ϕ

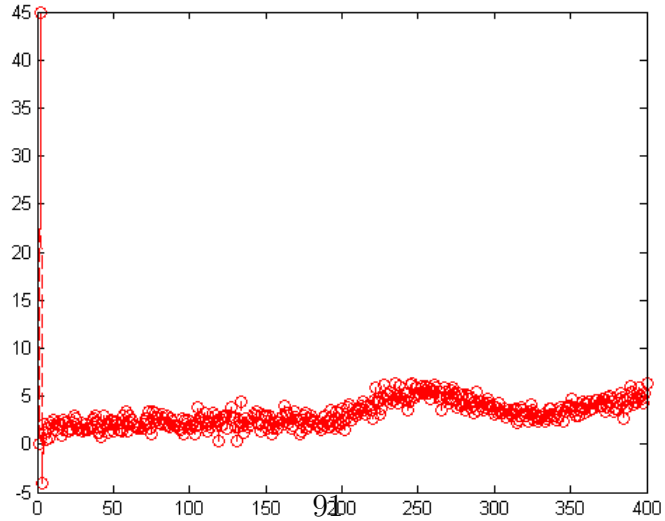

Figure C.17: Error in relative coordinate γ

Appendix D

Implementation Results

Note: All position values and angular values in the following tables are absoluted and in [$\times 10^{-3}m$] and [$\times 10^{-3}rad$] respectively. **Gain Sets**

Table D.1: Gain Sets

Gain Set Number	k_1	k_2
1	0.14	0.01
2	0.07	0.2

D.1 Zero desired polar angle with stationary leader

D.1.1 In line start

Plots of one of the datasets are shown in figures D.1 to D.5.

D.1.2 Angle offset start

Plots of one of the datasets are shown in figures D.6 to D.10.

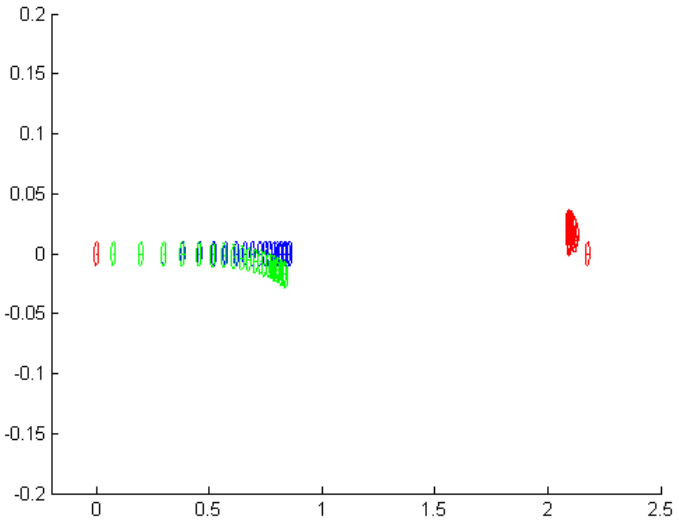

Figure D.1: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

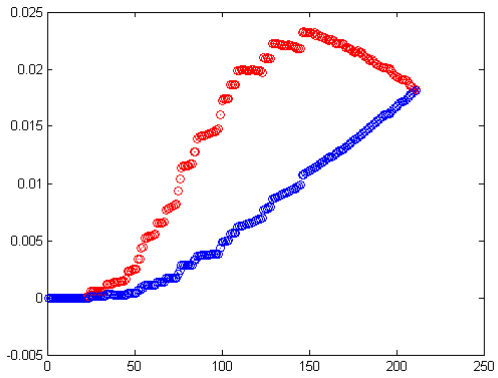

Figure D.2: Difference in Odom and Est Pose(Blue=x,Red=y)

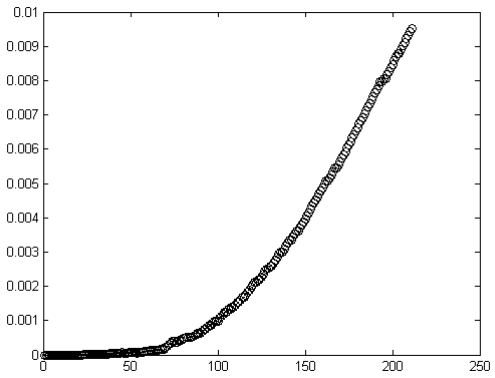

Figure D.3: Difference in Odom and Est Pose(Black=theta)

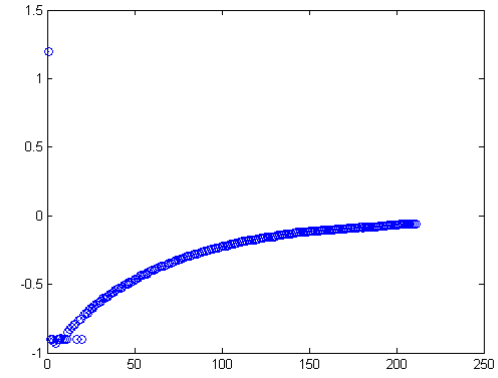

Figure D.4: Relative error from desired(Black=ρ)

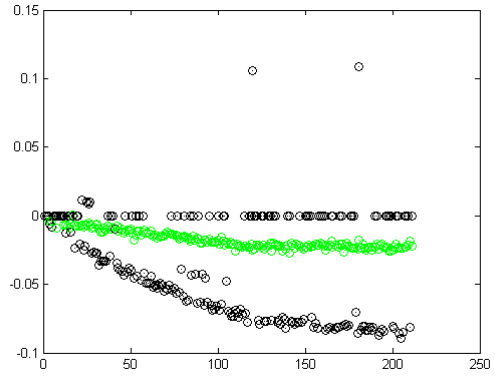

Figure D.5: Relative error from desired(Green=ϕ,Blue=γ)

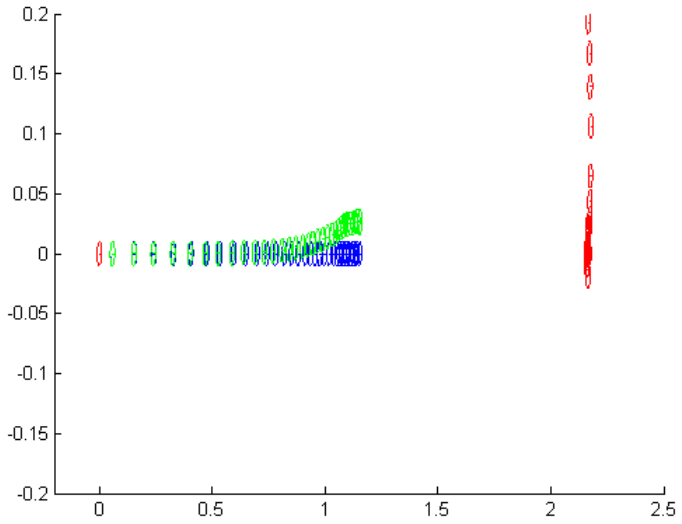

Figure D.6: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

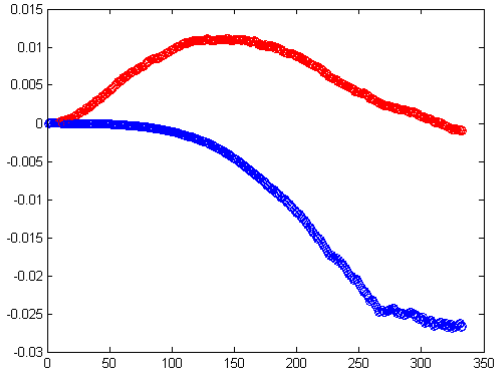

Figure D.7: Difference in Odom and Est Pose(Blue=x,Red=y)

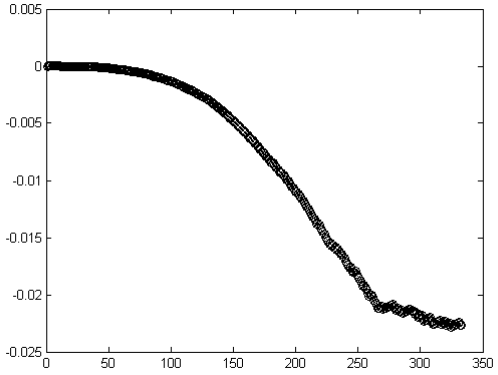

Figure D.8: Difference in Odom and Est Pose(Black=theta)

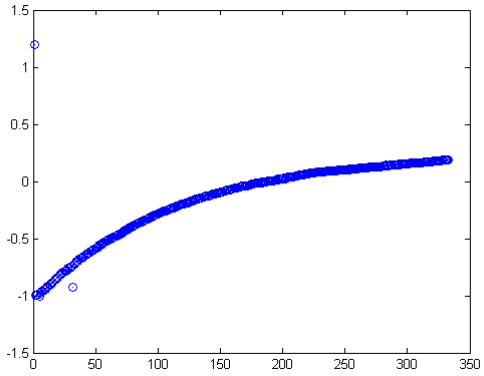

Figure D.9: Relative error from desired(Black=ρ)

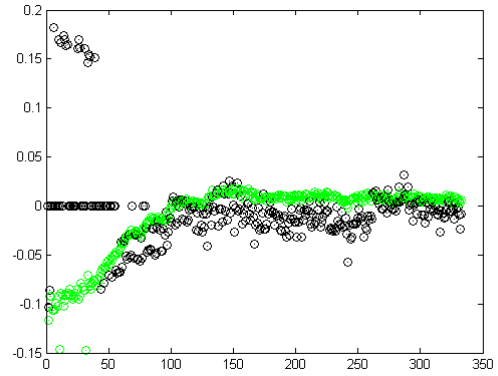

Figure D.10: Relative error from desired(Green=ϕ,Blue=γ)

D.2 Zero desired polar angle with moving leader

D.2.1 In line start

Plots of one of the datasets are shown in figures D.11 to D.15.

D.2.2 Angle offset start

Plots of one of the datasets are shown in figure D.16 to D.20.

D.3 Desired polar angle = 10

D.3.1 Stationary leader

Plots of one of the datasets are shown in figure D.21 to D.25.

D.3.2 Moving leader

Plots of one of the datasets are shown in figure D.26 to D.30.

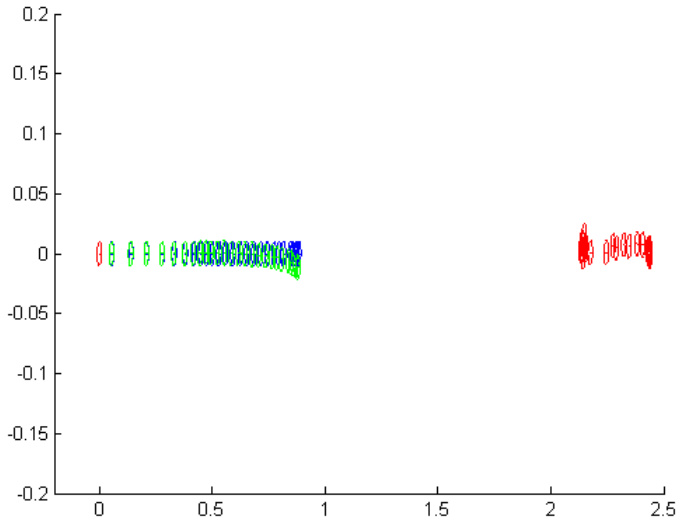

Figure D.11: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

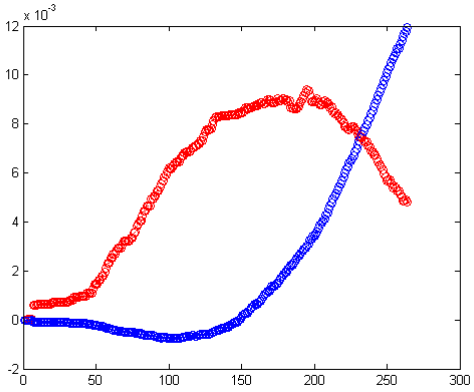

Figure D.12: Difference in Odom and Est Pose(Blue=x,Red=y)

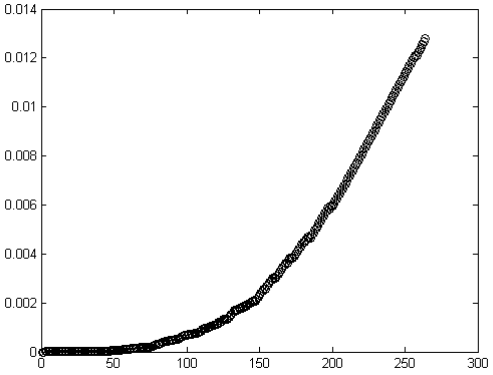

Figure D.13: Difference in Odom and Est Pose(Black=theta)

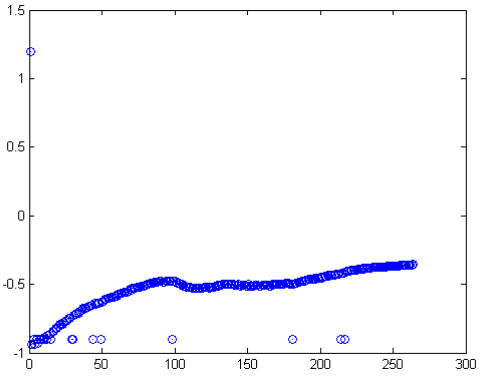

Figure D.14: Relative error from desired(Black=ρ)

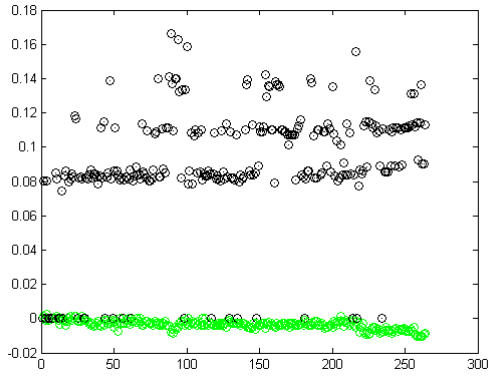

Figure D.15: Relative error from desired(Green=ϕ,Blue=γ)

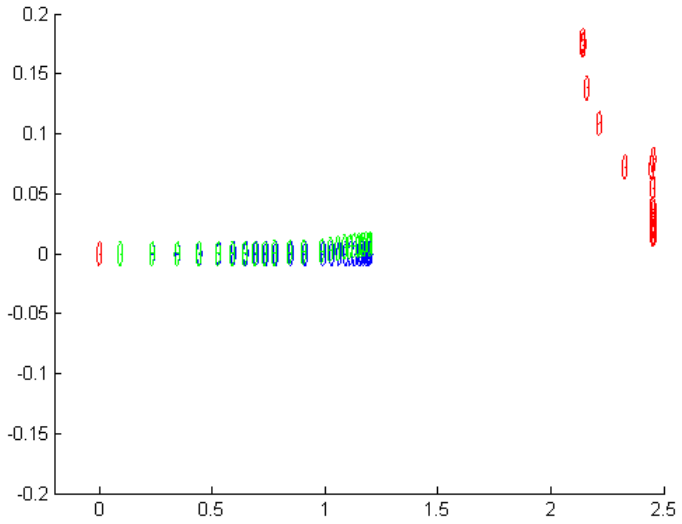

Figure D.16: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

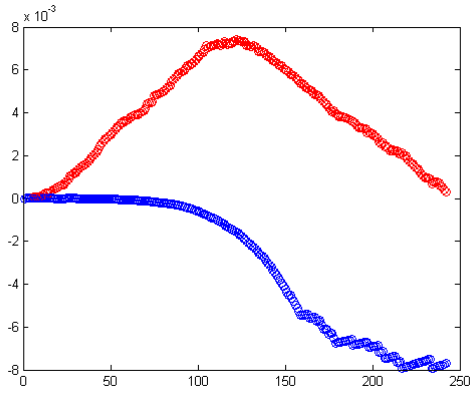

Figure D.17: Difference in Odom and Est Pose(Blue=x,Red=y)

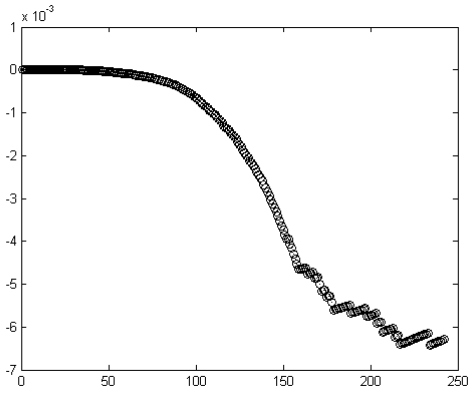

Figure D.18: Difference in Odom and Est Pose(Black=theta)

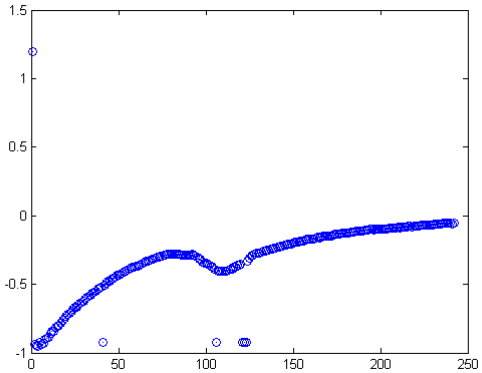

Figure D.19: Relative error from desired(Black=ρ)

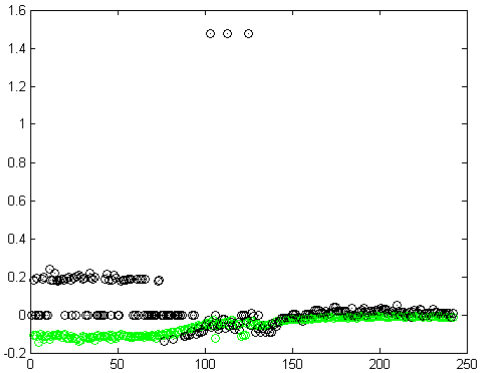

Figure D.20: Relative error from desired(Green=ϕ,Blue=γ)

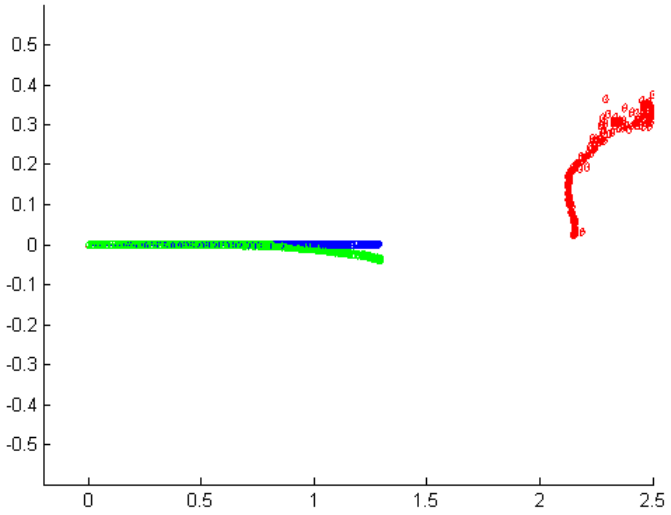

Figure D.21: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

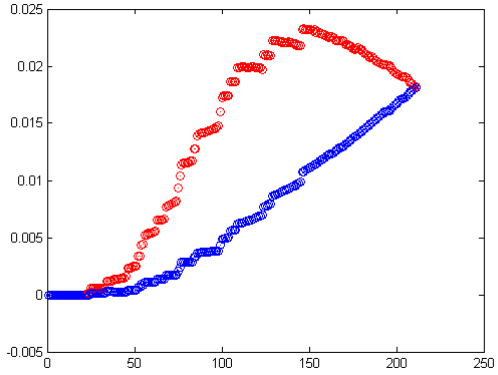

Figure D.22: Difference in Odom and Est Pose(Blue=x,Red=y)

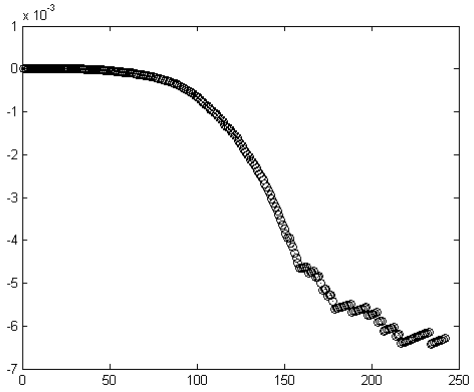

Figure D.23: Difference in Odom and Est Pose(Black=theta)

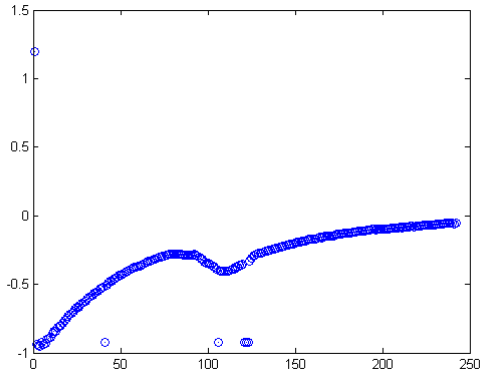

Figure D.24: Relative error from desired(Black=ρ)

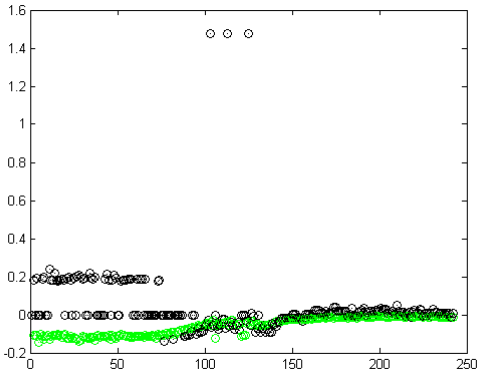

Figure D.25: Relative error from desired(Green=ϕ,Blue=γ)

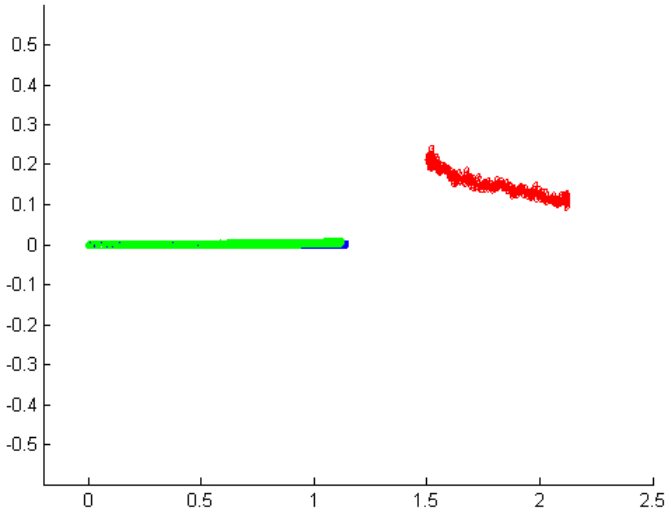

Figure D.26: Topview (Red=leader,Blue=Odom Pose,Green=Est Pose)

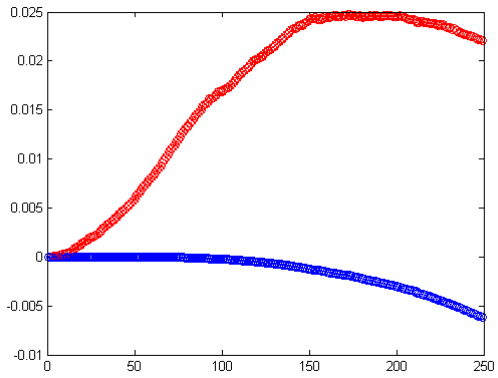

Figure D.27: Difference in Odom and Est Pose(Blue=x,Red=y)

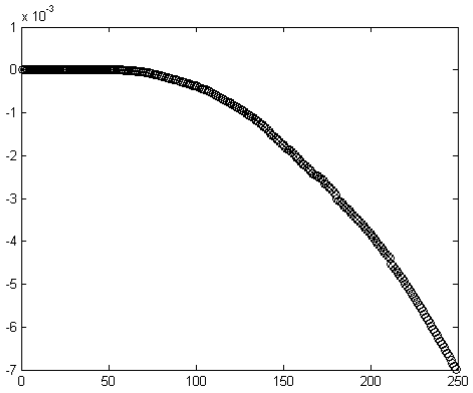

Figure D.28: Difference in Odom and Est Pose(Black=theta)

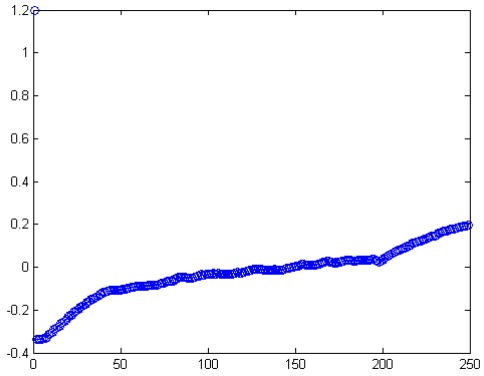

Figure D.29: Relative error from desired(Black=ρ)

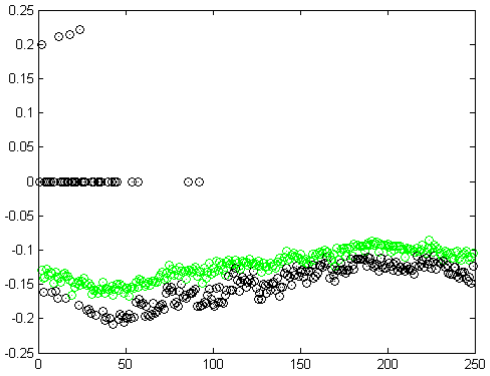

Figure D.30: Relative error from desired(Green=ϕ,Blue=γ)

Bibliography

[1] J. Green, "Mine rescue robots requirements outcomes from an industry workshop," in *Robotics and Mechatronics Conference (RobMech), 2013 6th*, pp. 111–116, IEEE, 2013.

[2] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, *et al.*, "Gamma-ray irradiation test of electric components of rescue mobile robot quince," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pp. 56–60, IEEE, 2011.

[3] M. Martin, P. Klupar, S. Kilberg, and J. Winter, "Techsat 21 and revolutionizing space missions using microsatellites," 2001.

[4] Z. Chen, T. I. Um, and H. Bart-Smith, "Modeling and control of artificial bladder enabled by ionic polymer-metal composite," in *American Control Conference (ACC), 2012*, pp. 1925–1930, IEEE, 2012.

[5] D. Richert and J. Cortés, "Optimal leader allocation in uav formation pairs ensuring cooperation," *Automatica*, vol. 49, no. 11, pp. 3189–3198, 2013.

[6] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[7] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM Siggraph Computer Graphics*, vol. 21, pp. 25–34, ACM, 1987.

[8] A. Khamis, "Minesweepers uses robotics awesomeness to raise awareness about landmines & explosive remnants of war," 2015.

[9] J. J. Wray, "Gale crater: the mars science laboratory/curiosity rover landing site," *International Journal of Astrobiology*, vol. 12, no. 01, pp. 25–38, 2013.

[10] E. Mueggler, M. Faessler, F. Fontana, and D. Scaramuzza, "Aerial-guided navigation of a ground robot among movable obstacles," in *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pp. 1–8, IEEE, 2014.

[11] S. Latscha, M. Kofron, A. Stroffolino, L. Davis, G. Merritt, M. Piccoli, and M. Yim, "Design of a hybrid exploration robot for air and land deployment (herald) for urban search and rescue applications," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 1868–1873, IEEE, 2014.

[12] F. Azis, M. Aras, M. Rashid, M. Othman, and S. Abdullah, "Problem identification for underwater remotely operated vehicle (rov): A case study," *Procedia Engineering*, vol. 41, pp. 554–560, 2012.

[13] S. J. Thomalla, M.-F. Racault, S. Swart, and P. M. S. Monteiro, "High-resolution view of the spring bloom initiation and net community production in the subantarctic southern ocean using glider data," *ICES Journal of Marine Science: Journal du Conseil*, 2015.

[14] "Fastwave becomes australian distributor of kongsberg seaglider system." http://subseaworldnews.com/2014/08/20/fastwave-becomes-australian-distributor-of-kongsberg-seaglider-system/. Accessed: 2015:10:01.

[15] D. P. M. Dr. Seb Swart, "Robot to test health of oceann 'lungs'." http://socco.org.za/news/robot-to-test-health-of-ocean-lungs/. Accessed: 2015:10:01.

[16] Y. Chen and Z. Wang, "Formation control: a review and a new consideration," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3181–3186, IEEE, 2005.

[17] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 6, pp. 926–939, 1998.

[18] X. Tu and D. Terzopoulos, "Artificial fishes: Physics, locomotion, perception, behavior," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 43–50, ACM, 1994.

[19] L. L. He and X. C. Lou, "Study on the formation control methods for multi-agent based on geometric characteristics," in *Advanced Materials Research*, vol. 765, pp. 1928–1931, Trans Tech Publ, 2013.

[20] W. Li and W. Shen, "Swarm behavior control of mobile multi-robots with wireless sensor networks," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1398–1407, 2011.

[21] J. D. Jeon and B. H. Lee, "Multi-robot formation shape control using convex optimization and bottleneck assignment," *Journal of Industrial and Intelligent Information Vol*, vol. 2, no. 1, 2014.

[22] M. Sisto and D. Gu, "A fuzzy leader-follower approach to formation control of multiple mobile robots," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2515–2520, IEEE, 2006.

[23] Y.-H. Chang, C.-Y. Yang, W.-S. Chan, H.-W. Lin, and C.-W. Chang, "Adaptive fuzzy sliding-mode formation controller design for multi-robot dynamic systems," *International Journal of Fuzzy Systems*, vol. 16, no. 1, p. 121, 2014.

[24] A.-M. Zou and K. D. Kumar, "Neural network-based adaptive output feedback formation control for multi-agent systems," *Nonlinear Dynamics*, vol. 70, no. 2, pp. 1283–1296, 2012.

[25] Y.-H. Chang, W.-S. Chan, C.-Y. Yang, C.-W. Chang, and T.-C. Chung, "Design of adaptive neural fuzzy formation controller for multi-robot systems," in *American Control Conference (ACC), 2012*, pp. 3161–3166, IEEE, 2012.

[26] M. A. Lewis and K.-H. Tan, "High precision formation control of mobile robots using virtual structures," *Autonomous Robots*, vol. 4, no. 4, pp. 387–403, 1997.

[27] N. E. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials and coordinated control of groups," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, vol. 3, pp. 2968–2973, IEEE, 2001.

[28] A. Kahn, J. Marzat, and H. Piet-Lahanier, "Formation flying control via elliptical virtual structure," in *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on*, pp. 158–163, IEEE, 2013.

[29] Y. Liu and Y. Jia, "An iterative learning approach to formation control of multi-agent systems," *Systems & Control Letters*, vol. 61, no. 1, pp. 148–154, 2012.

[30] K. L. Moore, M. Dahleh, and S. Bhattacharyya, "Iterative learning control: a survey and new results," *Journal of Robotic Systems*, vol. 9, no. 5, pp. 563–594, 1992.

[31] A. Krontiris, S. Louis, and K. E. Bekris, "General dynamic formations for non-holonomic systems along planar curvilinear coordinates," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4903–4908, IEEE, 2011.

[32] C. B. Low, "A flexible virtual structure formation keeping control design for nonholonomic mobile robots with low-level control systems, with experiments," in *Intelligent Control (ISIC), 2014 IEEE International Symposium on*, pp. 1576–1582, IEEE, 2014.

[33] K. H. Kowdiki, R. K. Barai, and S. Bhattacharya, "Leader-follower formation control using artificial potential functions: A kinematic approach," in *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pp. 500–505, IEEE, 2012.

[34] M. A. Kamel and Y. Zhang, "Decentralized leader-follower formation control with obstacle avoidance of multiple unicycle mobile robots," in *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pp. 406–411, IEEE, 2015.

[35] D. Zermas, "Control of a leader-follower mobile robotic swarm based on the nxt educational lego platform," in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, pp. 1381–1386, IEEE, 2011.

[36] G. Klančar, D. Matko, and S. Blažič, "Mobile robot control on a reference path," in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pp. 1343–1348, IEEE, 2005.

[37] L. Shi-Cai, T. Da-Long, and L. Guang-Jun, "Robust leader-follower formation control of mobile robots based on a second order kinematics model," *Acta Automatica Sinica*, vol. 33, no. 9, pp. 947–955, 2007.

[38] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor, "A vision-based formation control framework," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 813–825, 2002.

[39] J. P. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 4, pp. 2864–2869, IEEE, 1998.

[40] G. L. Mariottini, G. Pappas, D. Prattichizzo, and K. Daniilidis, "Vision-based localization of leader-follower formations," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pp. 635–640, IEEE, 2005.

[41] J. Yuan, "A feedback linearization based leader-follower optimal formation control for autonomous underwater vehicles," *Advances in Computer Science and its Applications*, vol. 1, no. 1, pp. 45–48, 2012.

[42] Y. Dai and S.-G. Lee, "The leader-follower formation control of nonholonomic mobile robots," *International Journal of Control, Automation and Systems*, vol. 10, no. 2, pp. 350–361, 2012.

[43] M. Farrokhsiar and H. Najjaran, "An unscented model predictive control approach to the formation control of nonholonomic mobile robots," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1576–1582, IEEE, 2012.

[44] D. Zhao, T. Zou, S. Li, and Q. Zhu, "Adaptive backstepping sliding mode control for leader–follower multi-agent systems," *IET control theory & applications*, vol. 6, no. 8, pp. 1109–1117, 2012.

[45] O. Mohareri, R. Dhaouadi, and A. B. Rad, "Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks," *Neurocomputing*, vol. 88, pp. 54–66, 2012.

[46] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 384–389, IEEE, 1990.

[47] H. A. Poonawala, A. C. Satici, N. Gans, and M. W. Spong, "Formation control of wheeled robots with vision-based position measurement," in *American Control Conference (ACC), 2012*, pp. 3173–3178, IEEE, 2012.

[48] Z. Peng, G. Wen, and A. Rahmani, "Leader-follower formation control of multiple nonholonomic robots based on backstepping," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 211–216, ACM, 2013.

[49] X. Li and J. Xiao, "Robot formation control in leader-follower motion using direct lyapunov method," *International Journal of Intelligent Control and Systems*, vol. 10, no. 3, pp. 244–250, 2005.

[50] Z.-P. JIANGdagger and H. Nijmeijer, "Tracking control of mobile robots: a case study in backstepping," *Automatica*, vol. 33, no. 7, pp. 1393–1399, 1997.

[51] J. D. Tardós, J. Neira, P. M. Newman, and J. J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 311–330, 2002.

[52] H. Surmann, A. Nüchter, and J. Hertzberg, "An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, no. 3, pp. 181–198, 2003.

[53] C. Holzmann and M. Hochgatterer, "Measuring distance with mobile phones using single-camera stereo vision," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pp. 88–93, IEEE, 2012.

[54] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1691–1696, IEEE, 2012.

[55] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[56] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[57] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 427–438, 2013.

[58] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[59] S. Chen, "Kalman filter for robot vision: a survey," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 11, pp. 4409–4420, 2012.

[60] Q. Gan and C. J. Harris, "Comparison of two measurement fusion methods for kalman-filter-based multisensor data fusion," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 37, no. 1, pp. 273–279, 2001.

[61] "Openkinect documents." http://openkinect.org/wiki/Documentation. Accessed: 2015:10:01.

[62] "Primesense 3d sensors." http://www.i3du.gr/pdf/primesense.pdf. Accessed: 2015:10:01.

[63] "Kinect 1 vs. kinect 2." https://channel9.msdn.com/coding4fun/kinect/Kinect-1-vs-Kinect-2-a-side-by-side-reference. Accessed: 2015:10:01.

[64] "Texas instraments." http://www.ti.com/lit/ds/symlink/lm2940c.pdf. Accessed: 2015:10:01.

[65] M. Nosrati, R. Karimi, and M. Hariri, "Detecting circular shapes from areal images using median filter and cht," *Global Journal of Computer Science and Technology*, vol. 12, no. 2, 2012.

[66] E. Wan, R. Van Der Merwe, *et al.*, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158, IEEE, 2000.

[67] G. Dudek and M. Jenkin, *Computational principles of mobile robotics*. Cambridge university press, 2010.

[68] "irobot." http://www.irobot.com/filelibrary/create/CreateManualFinal.pdf. Accessed: 2015:10:01.

[69] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[70] J.-H. Park, Y.-D. Shin, J.-H. Bae, and M.-H. Baeg, "Spatial uncertainty model for visual features using a kinect sensor," *Sensors*, vol. 12, no. 7, pp. 8640–8662, 2012.