

A NAVIGATION SYSTEM FOR INDOOR/OUTDOOR ENVIRONMENTS

WITH AN UNMANNED GROUND VEHICLE (UGV)

by

Milad Roigari

A Thesis Presented to the Faculty of the
American University of Sharjah
College of Engineering
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Mechatronics Engineering

Sharjah, United Arab Emirates

May 2015

Approval Signatures

We, the undersigned, approve the Master's Thesis of Milad Roigari.

Thesis Title: A Navigation System for Indoor/Outdoor Environments with an Unmanned Ground Vehicle (UGV)

Signature

Date of Signature

(dd/mm/yyyy)

Dr. Mohammad Abdel Kareem Rasheed Jaradat
Associate Professor, Department of Mechanical Engineering
Thesis Advisor

Dr. Mamoun Abdel-Hafez
Associate Professor, Department of Mechanical Engineering
Thesis Co-Advisor

Dr. Hasan Saeed Mir
Associate Professor, Department of Electrical Engineering
Thesis Committee Member

Dr. Shayok Mukhopadhyay
Assistant Professor, Department of Electrical Engineering
Thesis Committee Member

Dr. Mamoun Abdel-Hafez
Director, Mechatronics Engineering Graduate Program

Dr. Mohamed El Tarhuni
Associate Dean, College of Engineering

Dr. Leland Blank
Dean, College of Engineering

Dr. Khaled Assaleh
Director of Graduate Studies

Acknowledgements

I would like to express my gratitude to my supervisors, Dr. Mohammad Abdel Kareem Rasheed Jaradat, Associate Professor and Dr. Mamoun Fahed Saleh Abdel-Hafez, Associate Professor of department of mechanical engineering, American University of Sharjah, for their support and guidance, and for providing me with the opportunity to work on a thesis related to my own area of interest. I would also like to thank Dr. Ali Jhemi, my previous supervisor, for his patience and support. My deepest thanks go to my parents and my sister for their kind and continuous support all the way throughout my life. I would also like to thank my fellow students Alexander Adveev, Hesam Afzali, Muhannad Al-Omari, Rabiya Ahmed, Bara Emran, Syed Ali Rizvi and Amina Amoor for their friendship and for the good time that we spent together, as Mechatronics students at AUS. I am also grateful for Ehab Al-khatib and Wasim Al-Masri for their immense assistance during the testing phase of the thesis. Furthermore, I would like to thank Mr. Kent Bernales Roferos and Mr. John Mempin for their help and for valuable suggestions in the Mechatronics lab of AUS. Finally, I gratefully acknowledge the graduate assistantship provided to me by AUS, without which the present study could not have been accomplished.

Abstract

This thesis presents an approach for solving the global navigation problem of wheeled mobile robots. The presented solution for outdoor navigation uses Extended Kalman Filter (EKF) to estimate the robot location based on the measurements from Global Positioning System (GPS), inertial measurement unit (IMU) and wheel encoders. For indoor navigation (where GPS signals are blocked) another probabilistic approach, based on Monte Carlo Localization (MCL), is used for localization. This algorithm utilizes the map of the environment to estimate the posterior of the robot using the depth measurements from a Kinect sensor. The output from the Kinect sensor is processed to imitate the output of a 2D laser scanner by projecting the points from a thin horizontal strip of pixels in the image plane to the corresponding real world 3D coordinates using the pin-hole camera model. Two different controllers based on Dynamic Feedback Linearization (DFL) and Input-Output State Feedback Linearization (I-O SFL) have been analyzed, simulated and compared. Based on the thesis objective and the simulated results, the I-O SFL method was chosen for solving the trajectory tracking problem. A set of test experiments was conducted to evaluate the performance of the proposed system in outdoor, indoor and a combination of both environments. The results show that the robot can successfully navigate through the way-points with a great accuracy in indoor environments, while the accuracy in outdoor environments is within the 3m position accuracy of the GPS.

Search Terms: *Navigation, Extended Kalman Filter, Monte Carlo Localization, Input Output State Feedback Linearization, Dynamic Feedback Linearization, Kinect, Depth Camera*

Table of Contents

Abstract	5
List of Figures	8
List of Tables	12
List of Abbreviations	13
1. Introduction	15
1.1 Background	15
1.2 The Navigation Problem	16
1.3 Applications	17
1.4 Literature review	18
1.5 Software	26
1.5.1 Programming language	26
1.5.2 Used Libraries	26
1.6 Hardware	27
1.7 Thesis Overview	28
2. System Setup	29
2.1 Communication	31
3. Robot Motion	36
3.1 Introduction	36
3.2 Motion Model	36
3.2.1 Unicycle	36
3.2.2 Exact Motion	37
4. Trajectory Following	40
4.1 Motion Control	40
4.2 Dynamic Feedback Linearization	41
4.3 Input-Output State Feedback Linearization	48
5. Outdoor Navigation	55
5.1 Extended Kalman Filter	56
5.1.1 Filter Design	57
5.1.2 Simulation	60

5.2	Navigation Software	64
5.2.1	GUI	64
5.2.2	Geodetic Transformation	64
5.3	Practical Results	69
6.	Indoor Navigation	75
6.1	Monte Carlo Localization (MCL)	75
6.2	Kinect	79
6.2.1	Calibration	79
6.2.2	Measurement	82
6.3	Simulation	87
6.4	Practical Results	89
7.	Hybrid System (Indoor/Outdoor)	92
7.1	Practical Result	94
8.	Conclusion and Future Work	95
8.1	Summary	95
8.2	Future Work	96
	References	97
	Vita	103

List of Figures

Figure 1:	Why multi-sensor data fusion?	16
Figure 2:	A comparison of the posture stabilization controllers implemented on SUPERMARIO [35].	21
Figure 3:	Particle filters have been used successfully for on-board localization of soccer-playing Aibo robots with as few as 50 particles [47].	25
Figure 4:	General Hardware Setup	29
Figure 5:	Block Diagram of the system	30
Figure 6:	mBin Packet Frame	31
Figure 7:	Fletcher Algorithm	31
Figure 8:	First Loop	32
Figure 9:	Second Loop: Initialize	33
Figure 10:	Second Loop: Get byte	33
Figure 11:	Second Loop: Check Sync	34
Figure 12:	Second Loop: Check Checksum	34
Figure 13:	Second Loop: Calculate Checksum	34
Figure 14:	Second Loop: Packet Received	35
Figure 15:	Third Loop	35
Figure 16:	Robot pose, in global configuration space	37
Figure 17:	Motion carried out by a noise-free robot moving with constant velocities (v) and (ω) , starting at $(x \ y \ \theta)^T$	38
Figure 18:	Trajectory Following	40
Figure 19:	Steering Block(LabVIEW Robotics)	41
Figure 20:	Reference Trajectory of The Robot	43
Figure 21:	DFL: Reference and Robot Trajectories	44
Figure 22:	DFL: Robot motion: $x(m), y(m), \theta(rad)$	44
Figure 23:	DFL: Linear velocity	45
Figure 24:	DFL: Angular velocity	45
Figure 25:	DFL: Cartesian error (x,y)	46

Figure 26:	DFL: Norm of the error	46
Figure 27:	DFL: Robot following a square shaped trajectory.(brighter the color, the closer the robot is to the final position)	47
Figure 28:	DFL: Angular velocity of the robot following a square shaped trajectory.	48
Figure 29:	Point B outside the wheel axle with distance $b \neq 0$	49
Figure 30:	Pulling a toy car with a piece of rope	49
Figure 31:	I-O SFL: Reference and Robot Trajectories	51
Figure 32:	I-O SFL: Robot motion: $x(m),y(m),\theta(rad)$	51
Figure 33:	I-O SFL: Linear velocity	52
Figure 34:	I-O SFL: Angular velocity	52
Figure 35:	I-O SFL: Cartesian error (x,y)	53
Figure 36:	I-O SFL: Norm of the error	53
Figure 37:	I-O SFL: point B (yellow) is used by the robot to track a square shaped path	54
Figure 39:	The Extended Kalman filter algorithm (for full mathematical derivation of the EKF refer to [20])	57
Figure 40:	Flowchart: The basic outdoor navigation process	60
Figure 41:	EKF with encoder and heading measurement	61
Figure 42:	EKF with encoder, heading and noisy GPS measurements	62
Figure 43:	EKF is used with the I-O SFL algorithm to follow the square shaped trajectory	63
Figure 44:	Overall robot pose along the path(robot's color becomes brighter as it gets closer to the end of the path)	63
Figure 45:	Main features of the GUI	64
Figure 46:	Map features include a list of map providers, local caching of the map, zooming and panning)	65
Figure 47:	WGS84 Ellipsoid Parameters [57]	66
Figure 48:	ECEF and Reference Ellipsoid [57]	67
Figure 49:	A screen shot of a portion of the GUI.	68

Figure 50:	Conversion result from the software compared with the same conversion from google map engine.	68
Figure 51:	A simple algorithm that checks if the robot has reached the target way-point	69
Figure 52:	Outdoor navigation using a set of 17 way-points	70
Figure 53:	EKF output versus GPS output.	71
Figure 54:	Norm of the Cartesian error: the dotted lines shows the error in time when the robot reaches the current way-point and changes its destination to the next one.	71
Figure 55:	Outdoor Navigation with a circular path	73
Figure 56:	Norm of the Cartesian error: the dotted lines shows the error in time when the robot reaches the current way-point and changes its destination to the next one.	73
Figure 57:	Monte Carlo Localization Algorithm [20]	76
Figure 58:	The probability densities and particle sets for one iteration of the algorithm [60].	76
Figure 59:	Kidnapped robot problem	77
Figure 60:	Modified Monte Carlo Localization Algorithm [20]	78
Figure 61:	KLD-Sampling algorithm [61]	79
Figure 62:	Kinect Sensor Components	79
Figure 63:	Depth camera imaging geometry based upon the pinhole camera model.	80
Figure 64:	Kinect calibration using a checker board	81
Figure 65:	Depth image to range finder.	82
Figure 66:	Depth stream values [65]	83
Figure 67:	Depth Range [65]	83
Figure 68:	A frame from the depth stream and its 3D projection that shows a door at the end of a corridor	84
Figure 69:	(a) Infrared image of the pattern of speckles projected on a sample scene. (b) The resulting depth image [48].	84

Figure 70: Algorithm for computing the likelihood of a depth measurement z_t , assuming conditional independence between the individual depth measurements in the image.	86
Figure 71: Monte Carlo Localization with 1000 particles	87
Figure 72: Monte Carlo Localization with 5000 particles	88
Figure 73: Monte Carlo Localization with 200 particles	89
Figure 74: Indoor Navigation with a set of 800 particles	90
Figure 75: Robot trajectory shown in blue, has been recorded during the practical test in AUS	91
Figure 76: Ray Casting Algorithm	93
Figure 77: Ray Casting Flowchart	93
Figure 78: The hybrid system has been tested by placing it on the bridge between the two engineering buildings in AUS	94
Figure 79: Robot trajectory during the hybrid test, plotted using the recorded data.	94

List of Tables

Table 1:	KF Vs EKF [20]	57
Table 2:	Outdoor navigation error: way-points 0 to 7	72
Table 3:	Outdoor navigation error: way-points 8 to 16	72
Table 4:	Outdoor navigation(circular path) error: way-points 0 to 7	74
Table 5:	Outdoor navigation(circular path) error: way-points 8 to 15	74
Table 6:	Calibration Parameters	81

List of Abbreviations

API Application Programming Interface.

AUS American University of Sharjah.

CUDA Compute Unified Device Architecture.

DFL Dynamic Feedback Linearization.

ECEF Earth-Centered, Earth-Fixed.

EKF Extended Kalman Filter.

FOV Field Of View.

FPGA Field-Programmable Gate Array.

FPS Frames Per Second.

GPS Global Positioning System.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

ICP Iterative Closest Point.

IMU inertial measurement unit.

I-O SFL Input-Output State Feedback Linearization.

IR Infrared.

KF Kalman Filter.

LLA Latitude-Longitude-Altitude.

LTP Local Tangent Plane.

MAV Micro Aerial Vehicles.

mBin Microbotics binary protocol.

MCL Monte Carlo Localization.

NI National instruments.

OpenGL Open Graphics Library.

PIP point-in-polygon.

RGB Red-Green-Blue.

RMS Root Mean Square.

SDK Software Development Kit.

SGM Semi-Global Matching.

SLAM Simultaneous Localization And Mapping.

SV Satellite Vehicle.

UKF Unscented Kalman Filter.

WGS84 World Geodetic System 1984.

WMR Wheeled Mobile Robots.

WSN Wireless Sensor Network.

Chapter 1: Introduction

The aim of this thesis is to provide a comprehensive solution to the navigation problem of Mobile robots, by developing robust software that enables robots to withstand the numerous challenges arising in indoor and outdoor environments. This research focuses mainly on analyzing localization techniques using commercially available sensors, integrating them with a trajectory tracking algorithm and implementing the integrated system on a Wheeled Mobile Robots (WMR) platform.

1.1. Background

Mobile robots navigation covers a broad spectrum of Mechatronic systems. However, the most challenging problem is obtaining exact knowledge of the position of the robot. In search for a solution, researchers and engineers have developed a variety of systems, sensors, and techniques for mobile robot positioning [1].

Based on the measurement techniques used, positioning systems can roughly be classified into two categories:

Absolute position measurements (reference based systems)

- Global positioning systems [2, 3]
- Landmark navigation [4–6]
- Active beacons [7–9]
- Model matching [10–12]

Relative position measurements (dead-reckoning)

- wheel odometry [13]
- Visual odometry [14]
- Inertial navigation [15]

Each of these methods comes with a number of advantages and disadvantages. For instance, GPS signal is not always available or in case of relative measurement, the error will accumulate over time. Because of the lack of a single good method, developers of mobile robots usually combine the two methods to provide a third solution, called Sensor Fusion [16–18].

Figure 1 provides a brief comparison between sensor fusion and non sensor fusion systems.

Disadvantages of a non sensor fusion system	Advantages with sensor fusion
<ul style="list-style-type: none"> • Sensor loss • Limited spatial coverage • Limited temporal coverage • Imprecision • Uncertainty 	<ul style="list-style-type: none"> • Robustness and reliability • Extended spatial and temporal coverage • Increased confidence • Reduced ambiguity and uncertainty

Figure 1: Why multi-sensor data fusion?

The basic problem of multi-sensor data fusion is determining the best procedure for combining the multi-sensor data inputs [19]. Another significant issue with navigational systems that needs to be dealt with, is the unpredictability that exists in the operating environment, whether it is the unpredictability of the physical world, or the corruption in the sensors output due to noise. The level of uncertainty arises if the robot lacks critical information for carrying out its task. To address such issues scientists and researchers often come up with mathematical techniques and algorithms that can improve the overall robustness of the system. Thus robotics, is increasingly becoming a software science. The goal is to develop robust software that enables robots to withstand the numerous challenges arising in unstructured and dynamic environments [20]. The main advantage of a statistical approach is that explicit probabilistic models are employed to describe the various relationships between sensors and sources of information taking into account the underlying uncertainties [19].

1.2. The Navigation Problem

To overcome the navigation problem, we need to answer three key questions:

- Where am I?

The robot's ability to determine its own position in the robots reference environment, which is referred to as localization.

- Where am I going?

The robot's ability to plan a path towards the goal location. This process usually

consists of pre-planning a path that is optimized for the shortest distance by taking point obstacles and dangerous areas into consideration.

- How should I get there?

The robot's ability to accurately follow the planned path.

1.3. Applications

While fixed robots will always have a place in manufacturing, mobile robots equipped with reliable navigation systems, promises additional operation flexibility without requiring alterations in existing infrastructure in new applications. These applications include medical and surgical uses, personal assistance, security, warehouse and distribution applications, as well as ocean and space exploration. The list below provides a brief explanation on how self-navigating mobile robot can be of benefit to different applications:

Inspection and maintenance

A self-navigating mobile robot can be used as a tool to efficiently perform the maintenance and inspection work in challenging areas such as tunnels or other industrial installations.

Security and defense

A mobile robot can access dangerous environments and can provide a range of assistance from rescue missions to detection and sampling of toxic industrial agents (nuclear, radiological, biological and chemical) in the environment [21].

Logistic system

In the logistics sector, cost optimization and reduction of task execution times are two determining factors. Autonomous robots are able to carry heavy loads between different areas, without causing any risk.

Urban transport

The use of autonomous vehicles can revolutionize urban mobility by increasing the road capacity and safety, reducing the pollution and many other tasks.

Agriculture

The jobs involved in agriculture are not straightforward. Moreover, they include many repetitive tasks. Therefore, the use of self-navigating robots can easily outperform humans, hence reduce the cost and boost the performance.

Medical/Surgical Applications

Autonomous mobile robots can play a vital role in assisting doctors in surgical procedures, helping the patients by moving supplies such as medications, linens and food and basically by accomplishing more in less time.

1.4. Literature review

To get a grasp of the published work concerning this topic, a number of related papers have been briefly explained in this section in order to provide a basis for the proposed navigation solution.

1. Aided Navigation Techniques for Indoor and Outdoor Unmanned Vehicles

The navigation technique proposed in this paper [22], investigates the use and advantages of using Wireless Sensor Network (WSN) for indoor, and GPS for outdoor navigation. This system uses EKF to estimate the robot location by utilizing a laser range finder(indoor) and an IMU(outdoor) as the main measurement units. The system periodically corrects the estimate by integrating absolute measurements from a WSN interface(indoor), and a GPS(outdoor). Although no field test was conducted, the proposed solution was simulated using USARSim and Player/Stage. The result from the simulation proved the advantage of using external absolute sensors in improving the accuracy of navigation for both outdoor and indoor environments.

2. LOBOT: Low-Cost, Self-Contained Localization of Small-Sized Ground Robotic Vehicles

The work in [23] proposes a low-cost, self-contained localization system referred to as LOBOT, for small-sized ground robotic vehicles. One advantage of the

proposed system is that it does not require external reference facilities, expensive hardware, careful tuning or strict calibration. Another advantage is its capability of operating under various indoor and outdoor environments. LOBOT uses wheel encoders, an accelerometer and a magnetic field sensor as the measurements for local relative positioning. It utilizes infrequent GPS measurements for correcting the drifting error associated with local positioning sensors.

One of the key points of this paper is the adaptive approach used in GPS sampling. The more frequent GPS sampling is likely to result in better correction of positioning; However, more frequent GPS sampling also means significantly higher cost of power consumption [24], [25], [26]. LOBOT adjusts its GPS sampling frequency, according to the magnitude of the cumulative error of the local relative positioning. When the cumulative error of the local relative positioning between the current GPS sampling and its preceding GPS sampling increases, LOBOT increases its GPS sampling frequency accordingly; otherwise, LOBOT reduces its GPS sampling frequency.

3. GPS-compatible Indoor-positioning Methods for Indoor-outdoor Seamless Robot Navigation

The work in [27] is concerned with the problem of autonomous navigation with a Micro Aerial Vehicle (MAV) in indoor environments. The proposed system employs a mobile processor to address multi-floor mapping with loop closure, localization, planning, and autonomous control, including adaptation to aerodynamic effects during traversal through spaces with low vertical clearance or strong external disturbances, in real time. A laser range finder sensor is used as the main source of information for estimating position and yaw using Iterative Closest Point (ICP) algorithm [28], while a Kalman Filter (KF), similar to [29] is used to fuse the IMU data with redirected laser scans to provide altitude estimation.

The problems of mapping and drift compensation are addressed via a simplified occupancy grid-based incremental Simultaneous Localization And Mapping (SLAM) algorithm. This algorithm was chosen instead of particle filter-based oc-

cupancy grid [30] and feature-based methods [31, 32] due to the limited on-board processing power.

The performance of the on-board estimator was compared to ground truth, where ground truth is defined by a sub-millimeter accurate Vicon motion tracking system, which shows that the estimation tends to be more accurate while the robot controls along a specified trajectory.

4. Using the Kinect as a Navigation Sensor for Mobile Robotics

The work in [33] investigates the suitability of the Xbox Kinect optical sensor for navigation and SLAM. The prototype presented, uses the 3D point cloud data captured by Kinect to create 3D model of the environment. Then, it projects the 3D model to a 2D plane for 2D localization. A laptop mounted on a Pioneer III(P3-DX) robot is responsible for the SLAM and localization processing. The Robot System's software is based on the Robot Operating System (ROS), which provides libraries and tools including drivers for the Kinect, robot base, and laser scanner. RGBD SLAM [34] was used for 3D SLAM but found to be very slow, thus the robot speed was reduced for the algorithm to work in real-time.

The presented prototype was compared with traditional solutions with a laser scanner (Hokuyo URG-04LX) in terms of SLAM performance and suitability as a navigation sensor. The results show that the use of the Kinect sensor is viable. However, its narrow field of view, short depth range and the high processing requirements, limit its range of applications in practice.

5. WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation

Due to the perfect rolling constraints, a WMR is a typical example of a non-holonomic system. This paper tries to show the efficiency of DFL as a tool, simultaneously valid for both trajectory tracking and set point regulation problems. The quality of the proposed approach was assessed by implementing it on the laboratory prototype SuperMARIO, a two-wheel differentially driven mobile robot. Its performance was compared with several existing control techniques

such as linear and nonlinear feedback design in a number of experiments. The result of this comparison in terms of performance, ease of parameter tuning, sensitivity to non-idealities, generalizability to more complex WMRs, and relation with tracking controllers is gathered in Figure 2.


	Smooth time-varying stabilization	Nonsmooth time-varying stabilization	Design with polar coordinates	Dynamic feedback linearization
Achieved performance	very slow erratic	slow erratic	fast natural	fast natural
Ease of control tuning	a few parameters problematic	a few parameters critical	simple	simple PD ξ initialization
Sensitivity to nonidealities	many backups	backups and sampled feedback	good	good integral action
Generalization to other WMRs	yes if chained form	yes if chained form	no	yes if chained form
Relation with tracking control	extended from tracking	none	none	same PD control law

Figure 2: A comparison of the posture stabilization controllers implemented on SU-
PERMARIO [35].

6. Trajectory Tracking for a Mobile Robot

This project uses I-O SFL, a common approach used in controlling non-linear systems to solve the trajectory tracking problem, on a Pioneer differential drive robot. The kinematic model of this robot is based on unicycle model [36]. Input-Output linearization is a well known systematic approach to the design of trajectory tracking controllers.

The idea of the I-O SFL is to consider an output to control a point B out of the wheel axis, in particular at a distance b from it. In this way, we obtain that the point (x_B, y_B) is no more subject to the kinematic constraints and that it can move instantaneously in all direction.

It should be emphasized that the reference trajectory may exhibit a path with discontinuous geometric tangent without the need for the robot to stop and correct its orientation, The detailed explanation of this technique is available in Section 4.2.

7. Stereo Vision based indoor/outdoor Navigation for Flying Robots

Motivated by creating a usable platform in disastrous situations such as the Fukushima nuclear plant meltdown, the authors of this article [37] present an autonomous navigation system for a quad-rotor flying robot. The presented work utilizes an EKF to fuse the visual odometry with the data from an IMU to create a solution robust to challenging indoor/outdoor environments.

To calculate the visual odometry, the depth image is computed from rectified stereo images by Semi-Global Matching (SGM). The depth image is not only used for obstacle avoidance and mapping, but it also serves as base for visual odometry, which works on subsequent left camera images. The visual odometry is fused with IMU data for getting a system state estimate that is used for mapping and control. For error estimation, corresponding to the EKF update step, we need to use two measurement sources. Most of the time, the IMU acceleration measurement is dominated by the gravity vector. This pseudo gravity measurement for roll and pitch angle stabilization, which is especially important for flying platforms is fused with the second measurement update provided by the stereo odometry system. A low level PD attitude controller along with a PID position controller is used to control the robot. The trajectory is generated by linearly interpolating the position between way-points and low-pass filtering the result. The experimental result shows that the relative loop closure error of the presented navigation system on the 60m trajectory was less than 2% without conducting an actual loop closure.

8. Multi-Sensor Fusion for Robust Autonomous Flight in Indoor and Outdoor Environments with a Rotorcraft MAV

The design objective of this article [38] is to design a modular sensor-fusion filter that is easily extensible even for inexperienced users. This means that the amount of coding and mathematical deviation for the addition/removal of sensors should be minimal. One of the drawbacks of the popular EKF is the requirement of computing the Jacobian matrices, which is proven to be difficult and time consuming for a complex MAV system. For this reason the employed derivative-free Un-

scented Kalman Filter (UKF) based approach [39].

One of the issues in fusing multiple measurements is the possibility of the measurements arriving out-of-order to the filter; that is a measurement that corresponds to an earlier state arrives after the measurement that corresponds to a later state. This violates the Markov assumption of the Kalman Filter. The presented system, addresses these two issues by storing measurements in a priority queue, where the top of the queue corresponds to the oldest measurement. A pre-defined a maximum allowable sensor delay t_d of $100ms$ was set for our MAV platform. Newly arrived measurements that correspond to a state older than t_d from the current state (generated by state propagation) are directly discarded.

The experimental result after a total flight time of approximately 8 minutes, for a distance of 445 meters with an average speed of 1.5 m/s, demonstrate the robustness of framework in large-scale, indoor and outdoor environments.

9. A Navigation Subsystem For an Autonomous Robot Lawn Mower

As the topic suggests, this thesis describes a low-cost navigation system, suitable for outdoor commercial mobile robots. The proposed navigational algorithm utilizes two GPS receivers, an IMU and wheel encoders to provide measurements for a 7-state Kalman Filter [40] to accurately estimate the physical state of the robot. The real-time control of the robot happens in a National instruments (NI) sbRIO. The NI sbRIO platform has a 40 MHz Field-Programmable Gate Array (FPGA) unit which allows the robot to parse all the critical low-level data such as GPS and IMU on board to avoid latency and communication problems. When using low-cost off the shelf sensor for navigation, the optimality of classic Kalman filter can not be guaranteed. Thus, methods such as [41] can be applied on this system to improve the estimation result.

10. Three-state Extended Kalman Filter for Mobile Robot Localization

This report describes a similar approach as citeLawnMower with a 3-state discrete EKF with the measurement taken from the odometry, fiber-optic gyroscope, and the angular measurements to the ground markers (obtained from the video frames

taken during motion).

The author suggests that the overall filtering performance can be improved by:

- Online adjustment of the system and measurement covariance matrices based on the statistical properties of the incoming data.
- Extending the state of the Filter to include the translational and rotational velocities.
- Improved real time data correlation.
- Increased external measurement data frequency.
- Improved external data precision.

11. Particle Filters in Robotics

The work in this article [42] examines some of the innovative techniques that use particle filters to solve perceptual problems in robotics, by providing pointers to in-depth articles on the use of particle filters in robotics. Before the introduction of probabilistic methods, most was focused on planning and control problems under the assumption of fully modeled, deterministic robot and robot environments. This changed radically in the mid- 1980s, when the paradigm shifted towards reactive techniques. Particle filters are approximate techniques for calculating posteriors in partially observable controllable Markov chains (see [43]) with discrete time.

The author describes the application of particle filters in low dimensional spaces for robot localization. Mobile robot localization addresses the problem of estimation of a mobile robot's pose relative to a given map from sensor measurements and controls. According to the paper, the most difficult variant of the localization is the kidnapped robot problem [44]. This problem was reported, for example, in the context of the RoboCup soccer competition [45], in which judges picked up robots at random occasions and placed them somewhere else. In the context of localization, particle filters are commonly known as MCL [46]. In most variants of the mobile localization problem, particle filters have been consistently found to outperform alternative techniques, including parametric probabilistic techniques such as the KF and more traditional techniques. MCL has been implemented with

as few as 50 samples [47] on robots with extremely limited processing and highly inaccurate actuation, such as the soccer playing AIBO robotic shown in Figure 3.



Figure 3: Particle filters have been used successfully for on-board localization of soccer-playing Aibo robots with as few as 50 particles [47].

12. KLD-Sampling: Adaptive Particle Filters

The algorithm presented in this technical report is a variation of particle filters for state estimation, where it uses a statistical approach to increase the efficiency of particle filters by adapting the size of sample sets on-the-fly. 'The key idea of the KLD-sampling method is to bound the approximation error introduced by the sample-based representation of the particle filter.'

13. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications

Kinect sensor, as a low-cost consumer-grade range camera, has the potential to be used in mapping applications where accuracy requirements are less strict [48]. This paper discusses the calibration of the Kinect sensor and provides an analysis of the accuracy and resolution of its depth data which are essential in the realization of its potential. It has been showed that the random error of depth measurement increases with increasing distance to the sensor, and ranges from a few millimeters up to about 4 cm at the maximum range of the sensor. According to the article the errors and imperfections in the sensors output may originate from three main sources:

- The sensor.
- Measurement setup.
- Properties of the object surface.

The theoretical and experimental analysis of the geometric quality of the Kinect's depth data shown in the article provides a better understanding of how the sensor should be configured in order to achieve better quality data.

1.5. Software

Listed below, is the set of tools used in this thesis to develop the software that can perform the navigation task:

1.5.1. Programming language.

C# has been chosen as the main programming language of this thesis over traditional C/C++, because, as a part of Microsoft.Net framework it has a number of modern features that some are listed below:

- It has native garbage-collection.
- It has an enormous list of standard libraries, that are useful for achieving the objective of the thesis.
- It allows for both managed and native code blocks.
- It is more flexible when it comes to source code organization.

C# is an interesting language that, although it may run slower than C/C++ in some situations, but its rapid development paradigm makes it the superior choice among modern programming languages. Below is the list of external software libraries that are used in this thesis to enhance the already rich standard library of C#.

1.5.2. Used Libraries.

A set of software libraries has been used in conjunction with the standard C# libraries to improve the overall functionality of the software.

- OpenTK¹: low-level C# library that wraps OpenGL, OpenCL and OpenAL.
- GMap.NET²: An open source .NET control that enables the use of routing, Geocoding, directions and maps from online map providers such as Google, Yahoo! etc..

¹<http://www.opentk.com>

²<https://greatmaps.codeplex.com>

- EmguCV³: is a cross platform .Net wrapper to the OpenCV image processing library.
- Microsoft Kinect Software Development Kit (SDK) ⁴: This SDK enables developers to create applications that support gesture and voice recognition, using Kinect sensor technology on computers running Windows.

In addition to the mentioned external libraries, a custom graphical simulation environment has been developed for this thesis using low-level OpenGL Application Programming Interface (API).

1.6. Hardware

A group of hardware and devices has been employed in this thesis to test the performance of the proposed navigation solution.

1. NI LabVIEW Robotics Starter Kit: Robotics Platform for Teaching, Research, and Prototyping

- Motors: Pitsco Education 12 VDC motors featuring 152 rpm and 300 oz-in. of torque
- Encoders: Optical quadrature encoders with 400 pulses per revolution
- Processor: 400 MHz Freescale real-time processor
- Memory: 128 MB DRAM, 256 MB nonvolatile storage
- Connection: RS232 serial port for peripheral devices
- Weight: 3.6 kg (7.9 lb)

2. MIDG IIC: A GPS aided inertial navigation system (INS) for use in applications requiring attitude, position, velocity, acceleration, and angular rates for navigation or control

- Input Voltage: 10 VDC - 32 VDC
- Power: 1.2W max (including GPS antenna)
- Position Accuracy: 2m (CEP) with WAAS/EGNOS available, 3 m (CEP) otherwise

³<http://www.emgu.com/>

⁴<http://www.microsoft.com/en-us/download/details.aspx?id=40278>

- Data Output Rates: Position , Velocity, attitude, rates, accelerations - 50 Hz
GPS measurements - 4 Hz
- Output: RS422 async., 115200 baud (configurable), 8-N-1
- Weight: 55 grams

3. Microsoft Kinect for Xbox 360

- Viewing angle: 43°vertical by 57°horizontal field of view
- Frame rate (depth and color stream): 30 Frames Per Second (FPS)
- Resolvable Depth: 0.8m to 4.0m
- Latency: approximately 90ms with processing

1.7. Thesis Overview

This thesis is organized in eight major chapters:

- Chapter 2 discusses the overall hardware and software architecture of the robot. This chapter introduces the method of communication between major parts of the system and goes into detail about how it is implemented in the software.
- Chapter 3 explains the motion model that is used throughout this thesis.
- Chapter 4 is comprised of analyzing, implementing and testing two trajectory following algorithms. In this chapter a comparison has been made between the trajectory following methods, based on the proposed objectives of the robot.
- Chapter 5 focuses on the use of EKF for localization, and explains how it is being used for sensor fusion by combining GPS, Heading sensor and encoder data.
- Chapter 6 discusses the process of Monte Carlo Localization; a variation of particle filter algorithm. It also discusses how the depth measurements from a Kinect sensor is used to localize the robot in indoor environments where GPS signals are blocked.
- Chapter 7 offers a solution for integrating the indoor and outdoor algorithms that involves ray-casting on the indoor map polygon. The resulting hybrid system is able to autonomously switch between the two algorithms based on the location.
- Chapter 8 concludes the thesis by summarizing the content and points out directions for further improvement of the system and future work on the subject.

Chapter 2: System Setup

There are a few limitations to the real-time controller board available on the NI LabVIEW Robotics Starter Kit¹. A major limitation is the limited processing power with only a single RS-232 port. To overcome this limitation, a laptop computer has been added to the system as the main processing unit which can effectively handle all the navigation and localization responsibilities, thus the on-board processing on the robotic platform will be limited to low-level controllers such as the DC motor velocity controller. The laptop will continuously communicate with the robot platform through a serial connection to simultaneously read the velocity outputs and to send the appropriate control commands to the robot controller unit. The general hardware setup can be seen in Figure 4.

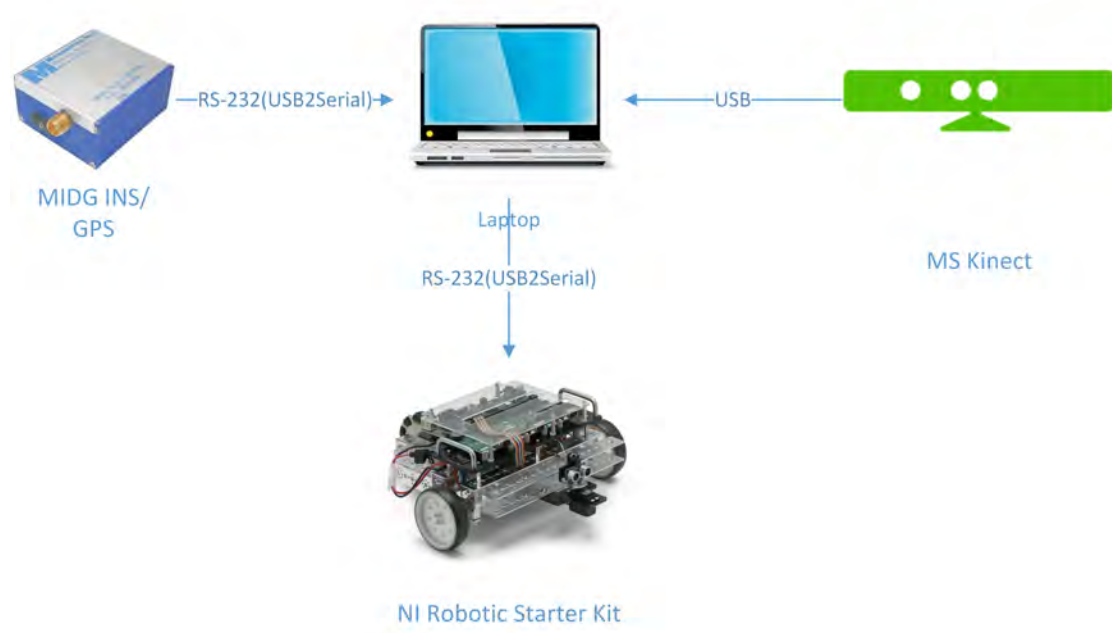


Figure 4: General Hardware Setup

The block diagram shown in Figure 5 illustrates the high-level/low-level system integration.

¹sbRIO-9632.

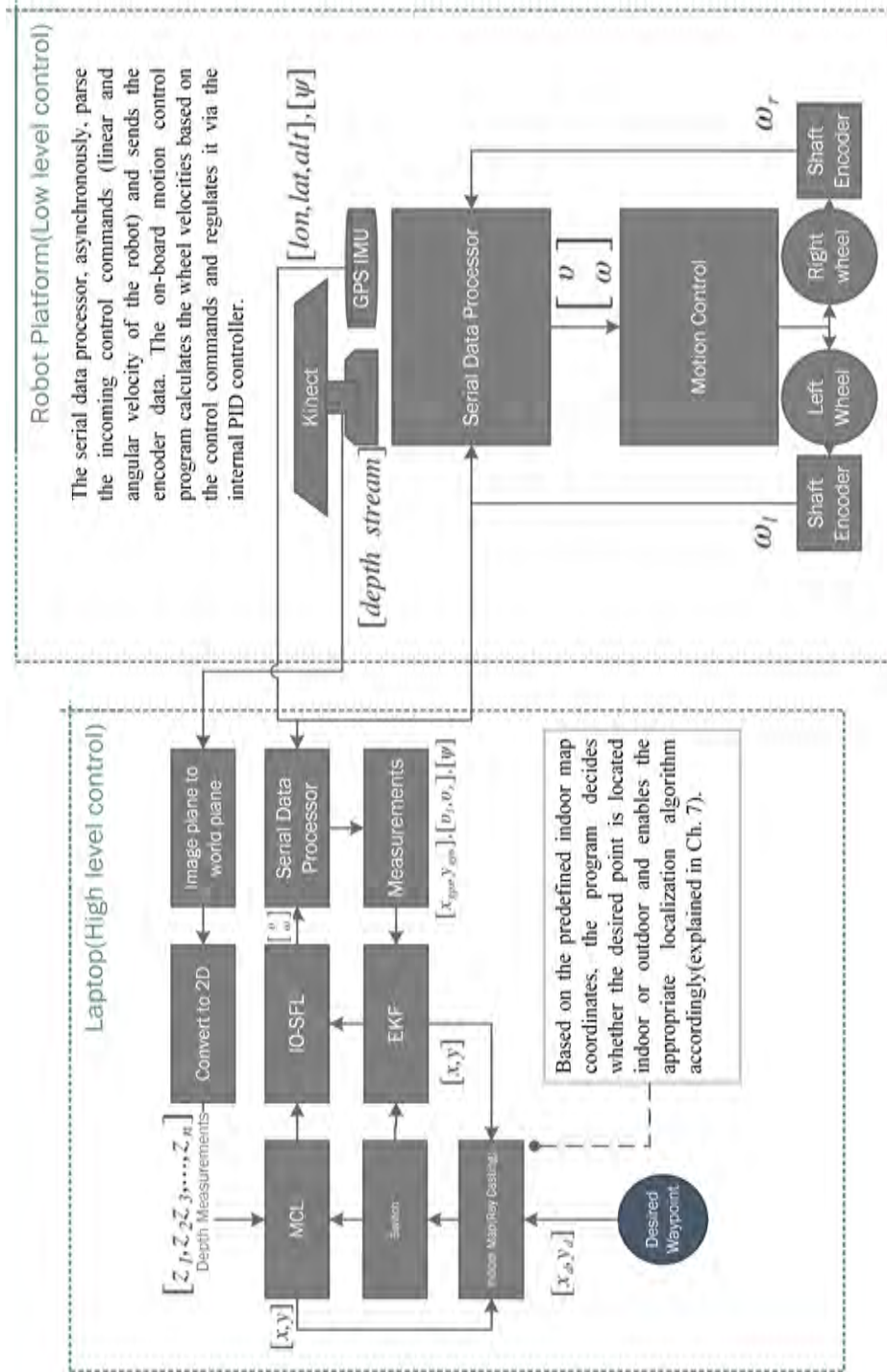


Figure 5: Block Diagram of the system

2.1. Communication

The MIDG IMU/INS package uses the Microbotics binary protocol (mBin) to communicate with the host computer. Due to the simple yet reliable characteristic of this protocol, it has also been selected as the communication protocol between the robotic platform and the laptop.

The mBin protocol is a standard binary package that has the following structure:

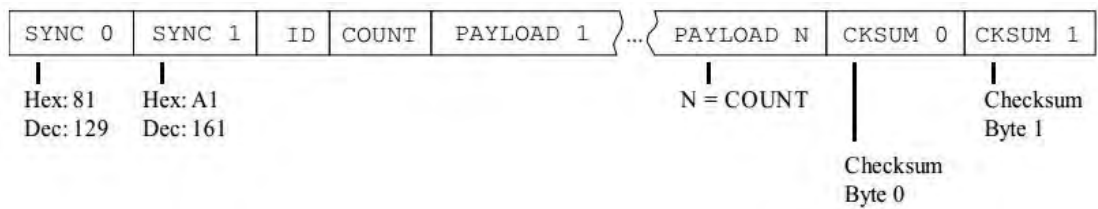


Figure 6: mBin Packet Frame

As can be seen in Figure 6, the packet starts with two predefined sync bytes followed by the message ID. However the most important part of the packet is the Fletcher checksum(CKSUM0 and CKSUM1), which is computed over the bytes that include the message ID, Count byte, and the payload bytes. The basic algorithm can be seen in Figure 7.

Algorithm 2.1.1: COMPUTEFLETCHER(*id, count, payload*)

```

cksum0 ← 0
cksum1 ← 0
for each byte ∈ IDtoPayload
  do {
    cksum0 ← cksum0 + byte
    cksum1 ← cksum0 + cksum1
  }

```

Figure 7: Fletcher Algorithm

The sbRIO-9632 device is programmed using the LabVIEW graphical development environment. To process the information received at the serial port, three separate loops are defined with the following responsibilities:

1. The first loop will add the data bytes received at the serial port to the 'Byte Stream' queue.
2. The Second loop will process the available data in the 'Byte Stream' queue and try to parse the received packets. If successful, the packets will be added to the 'Packet Stream' queue
3. The Third loop will extract the message ID, payload length and the payload data itself from the 'Packet Stream' queue.

The first loop(Figure 8) consists of three other loops. One to write the available data in the 'Write Stream' queue to the serial port's write buffer. The next is a while loop that ensures the data is fully received before reading it from the read buffer. Finally the received bytes are added to the 'Byte Stream' queue.

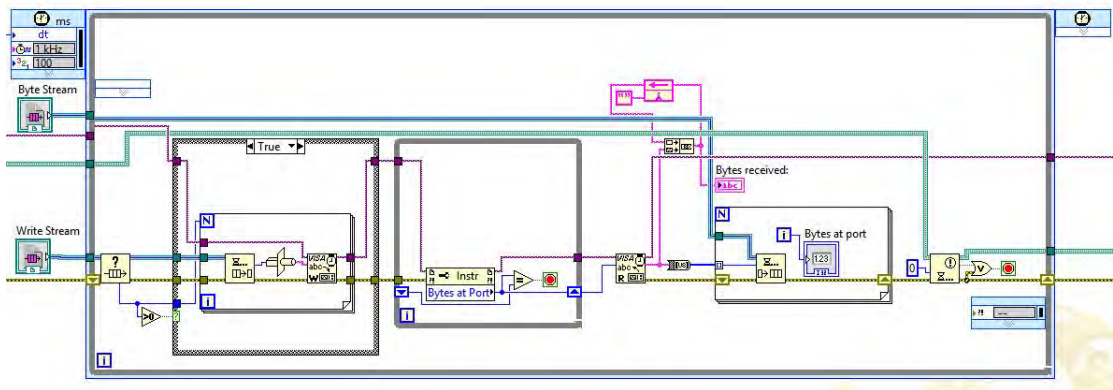


Figure 8: First Loop

The second loop's job is to parse the packet data in the stream queue. It contains a 5 states case structure block:

Initialize This is the first case which will initialize Shift registers and will create a buffer array with 256 elements(the size of the buffer must be greater than the largest packet that may be received at the serial port).

Get byte This case comes after the initialization, and its job is to dequeue one element at a time and add that element in the last position of the buffer array. To achieve this first the array rotated by -1 then the new item is inserted in the $(n - 1)^{th}$ element of the array, where (n) is the size of the array). This means that each new

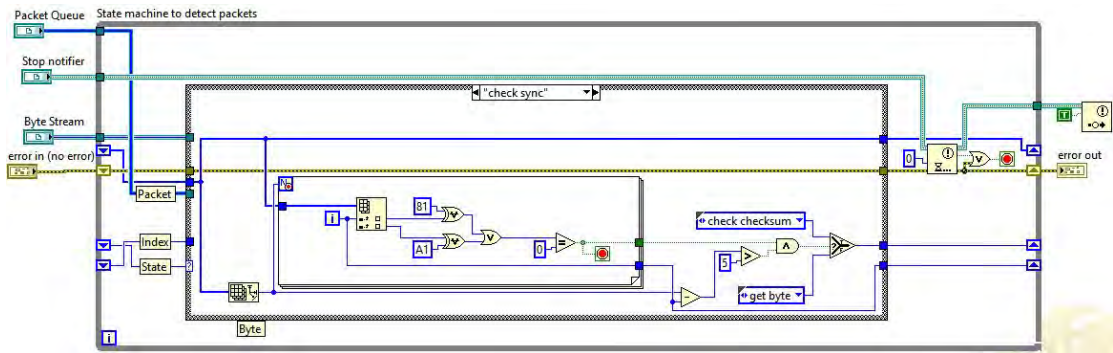


Figure 11: Second Loop: Check Sync

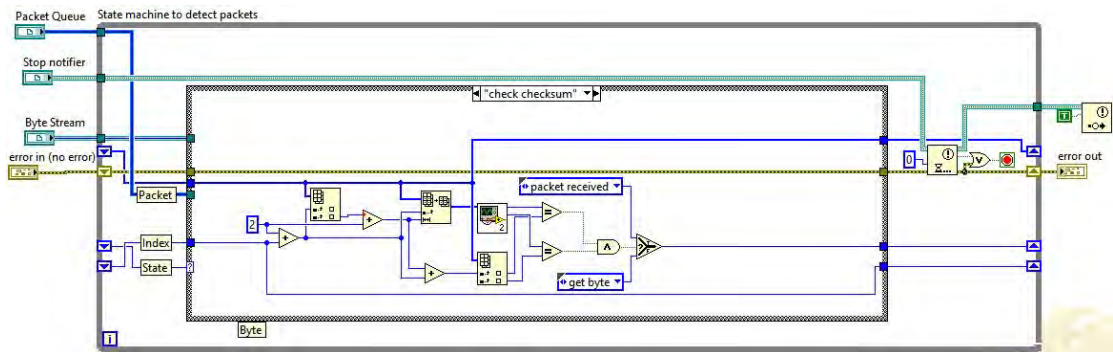


Figure 12: Second Loop: Check Checksum

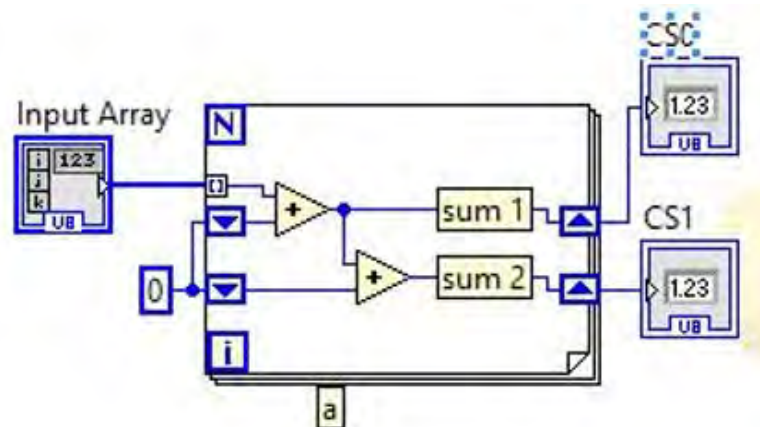


Figure 13: Second Loop: Calculate Checksum

Packet Received This state will check the last two bytes of the received packet with the calculated checksum bytes. If they both match it means the packet is correct and it will change the state to 'packet received'. If not it will go back to 'get byte'.

It should be noted that the Producer/Consumer design pattern that has been used here is based on the Master /Slave pattern which is geared towards enhancing the data sharing between multiple loops running at different rates. As with the standard Master/ Slave design pattern, the Producer/Consumer pattern is utilized to decouple processes

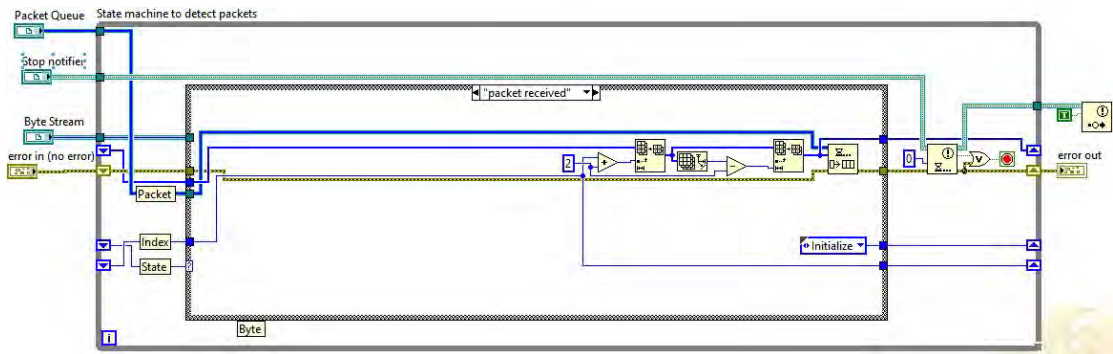


Figure 14: Second Loop: Packet Received

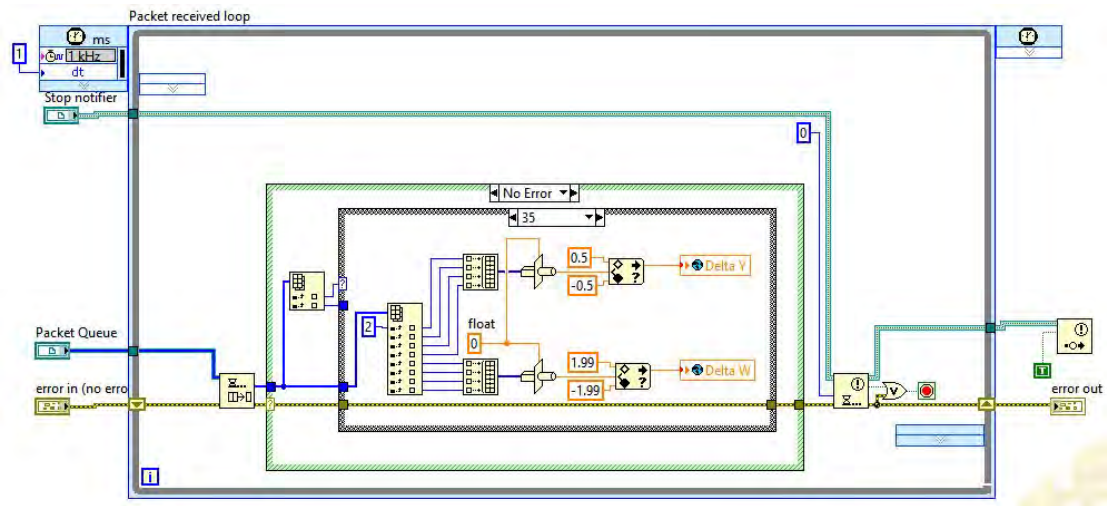


Figure 15: Third Loop

that produce and consume data at different rates. The first loop here (Producer loop) is producing the data which is consumed by the second loop(Consumer loop). The same thing goes for the second and the third loop which makes the second loop the producer of the packet data and the third loop its consumer.

Chapter 3: Robot Motion

3.1. Introduction

This chapter elaborates on the motion model used for this study to model and controls the robot. It starts with examples of the basic motion model of a unicycle and how it is applied to derive the state transition, which plays an essential role both in the trajectory following and the filter design.

3.2. Motion Model

3.2.1. Unicycle.

Kinematics is the geometry of pure motion, when dealing with the kinematics of robots. We are only interested in the motion without reference to force or mass. The robot kinematic state or pose consists of its two-dimensional planar coordinates (x,y) with respect to an external coordinate frame, along with its angular orientation θ . The pose of the robot in its configuration space is described as follows:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \in \mathfrak{R}^3 \quad (1)$$

The orientation of a robot or as is often called, heading suggests that a robot with heading $\theta = 0$ points into the direction of x -axis (Figure 16). Using the following assumptions, we can simplify the motion of differential drive robots to a unicycle model:

- No sliding on the wheels.
- Each wheel adds a non-holonomic constrain to the system(no movement perpendicular to the rolling direction)
- The instantaneous limitation of the wheel does not limit the robot maneuverability.

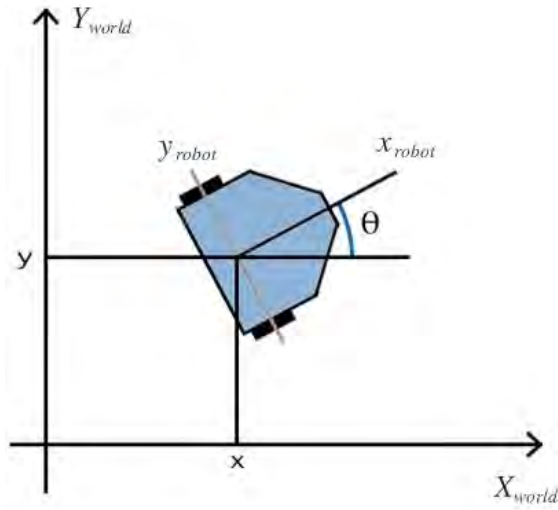


Figure 16: Robot pose, in global configuration space

The kinematic model of a unicycle is usually described by a simple non-linear model [49]:

$$\begin{aligned}
 \dot{x} &= v \cos \theta \\
 \dot{y} &= v \sin \theta \\
 \dot{\theta} &= \omega
 \end{aligned}
 \tag{2}$$

where

v = Linear velocity of the robot

ω = Angular velocity of the robot

For any position (x, y) , the unicycle can point at any direction (θ) . However, we know the fact that the unicycle can not move sideways if there is no slippage. This means that there is a constraint that:

$$\dot{x} = v (\cos \theta \hat{x} + \sin \theta \hat{y})
 \tag{3}$$

Or

$$\dot{y} = \dot{x} \tan \theta
 \tag{4}$$

3.2.2. Exact Motion.

Mobile robots operating in planar frames can be described in two specific motion models based on the available information. Both models are rather complimentary in the type of motion information that is being processed. The first model assumes that the motion data u_t specifies the velocity commands given to the robot's motors. The robotic platform (NI Robotic Starter Kit v.2) that is used in this thesis, like many commercial mobile robots, employs this motion model, meaning that it is actuated by independent linear and angular velocities. The second model assumes that one is provided with odometry information (distance traveled, angle turned). In practice, odometry models tend to be more accurate than velocity models [20]. However, it is only available post-the-fact. Therefore, it cannot be used for motion planning.

To show the kinematics for an ideal, noise-free robot with the velocity model (Figure 17), let $u_t = (v \ \omega)^T$ denote the control at time t . If both velocities are kept at a fixed value for the entire time interval $[t-1, t]$, the robot moves on a circle with radius:

$$r = \left| \frac{v}{\omega} \right| \quad (5)$$

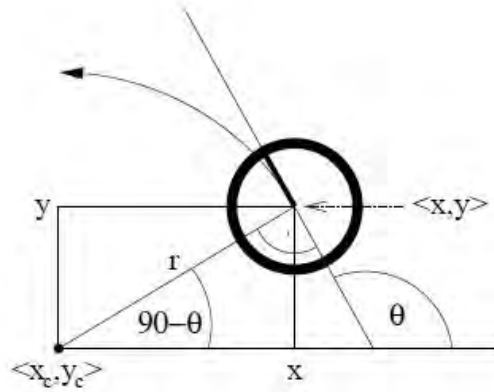


Figure 17: Motion carried out by a noise-free robot moving with constant velocities (v) and (ω) , starting at $(x \ y \ \theta)^T$

In case the robot does not turn at all (i.e., $\omega = 0$), Equation (5) will be equal to infinity, which means the robot is moving on a straight line. A straight line corresponds to a circle with infinite radius, hence we note that r may be infinite.

If we let $x_{t-1} = (x \ y \ \theta)^T$ be the initial pose of the robot, and suppose we keep the velocity constant at $(v \ \omega)^T$ for some time Δt . As one easily shows, the center of the circle is at

$$x_c = x - \frac{v}{\omega} \sin \theta \quad (6)$$

$$y_c = y + \frac{v}{\omega} \cos \theta \quad (7)$$

The variables $(x_c, y_c)^T$ denote this coordinate. After Δt time of motion, our ideal robot will be at $x_t = (x', y', \theta')$ with

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega} \sin(\theta + \omega \Delta t) \\ y_c - \frac{v}{\omega} \cos(\theta + \omega \Delta t) \\ \theta + \omega \Delta t \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} -\frac{v}{\omega} \sin \theta + \frac{v}{\omega} \sin(\theta + \omega \Delta t) \\ \frac{v}{\omega} \cos \theta - \frac{v}{\omega} \cos(\theta + \omega \Delta t) \\ \omega \Delta t \end{pmatrix} \quad (8)$$

Chapter 4: Trajectory Following

4.1. Motion Control

To perform trajectory tracking of WMRs (Figure 18), kinematic models are used to design feedback laws because the dynamic terms can be canceled out via feedback. The output from the controller is then used by the existing low-level PID controller on the platform as the velocity reference for each wheel.

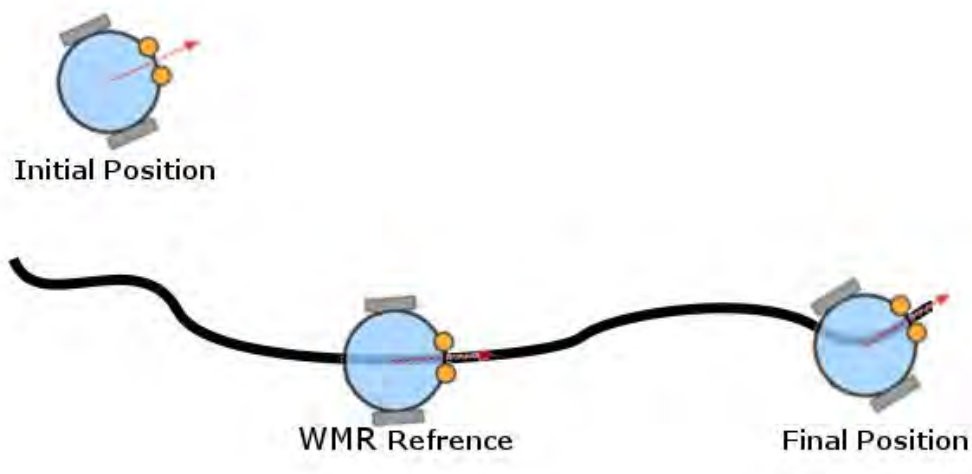


Figure 18: Trajectory Following

The basic control scheme of the system is made of a high-level controller, which will compute the velocity set-points of the motors, and a low-level controller that will have this set-points as an input, to control the real velocity of the motors. Given a desired position, the high-level controller decides the motion of the robot while the low-level controller, controls the motors in a way that the desired motion is achieved. If the control gains in the low-level controller are high enough the delay between the desired and actual velocity will be negligible. To make things easier and more compatible with the robotic platform, NI LabVIEW Robotics is equipped with such controller that accepts linear and angular velocity of the robot as the input and controls the motors velocity internally Figure 19.

The design of the high-level controller is based on the kinematic model of the robot which has to take into account the constraints introduced by the wheels. The complexity of the high-level controller is raised by the nonlinearity that exists in the

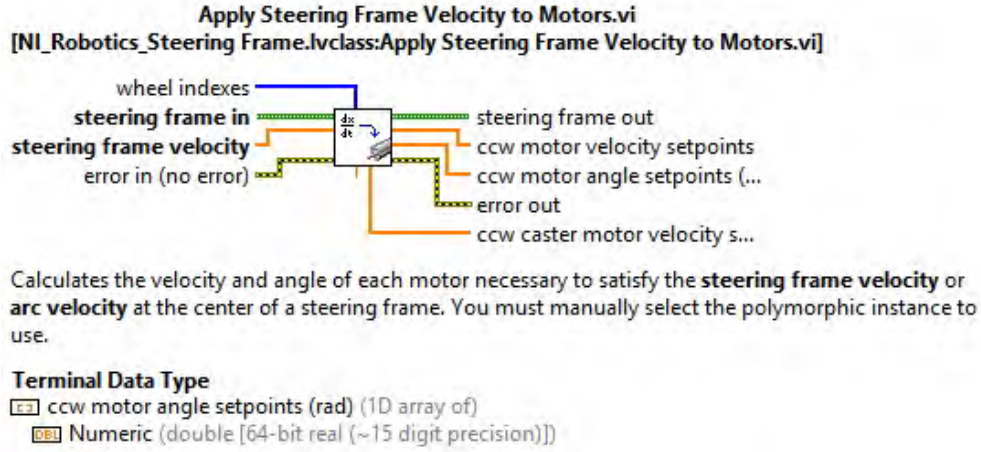


Figure 19: Steering Block(LabVIEW Robotics)

system. The rest of this chapter is focused on the design, test and comparison of two control systems that overcomes the nonlinearity problem.

4.2. Dynamic Feedback Linearization

DFL is an efficient design tool leading to a solution for both trajectory following and set-point regulation problems [35]. The odd nature of the nonholonomic kinematics of a unicycle, indicates that linear control is ineffective, and innovative design techniques are required.

After preliminary attempts, the trajectory tracking problem was globally solved by using DFL in [50] and [51]. The DFL problem consists of finding, a feedback compensator of the form:

$$\begin{aligned}\dot{\xi} &= a(q, \xi) + b(q, \xi) u \\ \dot{\omega} &= c(q, \xi) + d(q, \xi) u\end{aligned}\tag{9}$$

where $\xi \in \mathbb{R}^v$ and $u \in \mathbb{R}^m$, such that the closed loop system (11) is equivalent to:

$$\dot{q} = G(q) \omega \quad q \in \mathbb{R}^n, \omega \in \mathbb{R}^m\tag{10}$$

The procedure [35]: for exact linearization of the unicycle model using DFL is listed below:

1. Based on the unicycle kinematics (2), a new state ξ is introduced in the controller:

$$\begin{aligned}\dot{\xi} &= u_1 \cos \theta + u_2 \sin \theta \\ v &= \xi \\ \omega &= \frac{u_2 \cos \theta + u_1 \sin \theta}{\xi}\end{aligned}\tag{11}$$

2. Then coordinates are transformed with respect to the new state:

$$\begin{aligned}z_1 &= x \\ z_2 &= y \\ z_3 &= \dot{x} = \xi \cos \theta \\ z_4 &= \dot{y} = \xi \sin \theta\end{aligned}\tag{12}$$

3. The resulting system is a linear system with two decoupled integrators:

$$\begin{aligned}\ddot{z}_1 &= u_1 \\ \ddot{z}_2 &= u_2\end{aligned}\tag{13}$$

1. Finally using a simple PD feedback regulator the system can be controlled :

$$u_1 = -k_{p1}x - k_{d1}\dot{x} \quad u_2 = -k_{p2}y - k_{d2}\dot{y}\tag{14}$$

Based on DFL, if we assume the robot follows a smooth path $(x_d(t), y_d(t))$ with no discontinuity, by modifying only the 4th step, we can design an exponentially stabilizing feedback to stabilize the trajectory tracking error:

$$\begin{aligned}u_1 &= \ddot{x}_d + k_{p1}(x_d - x) + k_{d1}(\dot{x}_d - \dot{x}) \\ u_2 &= \ddot{y}_d + k_{p2}(y_d - y) + k_{d2}(\dot{y}_d - \dot{y})\end{aligned}\tag{15}$$

To obtain the actual control inputs, the result from Equation (15) should be directly sent to the dynamic compensator Equation (11) in step 1. The result is only valid if the dynamic feedback compensator Equation (11) does not meet the singularity $v = \xi = 0$. This may only happen during the initial transient of an asymptotic tracking problem.

The controller has been tested by simulating the robot motion with the kinematic model 5 in Matlab, Tracking the 8 figured trajectory of Figure 20. The travelled path by the robot can be observed in Figure 21 while the changes in the robot pose (Figure 22) and velocities (Figure 23 and Figure 24) proves the effectiveness of the algorithm.

$$x_{des} = 0.5\sin\left(\frac{t}{20}\right), \quad y_{des} = 0.5\sin\left(\frac{t}{40}\right), \quad t \in [0, T] \quad (16)$$

With initial conditions defined by:

$$x_0 = 0.1, \quad y_0 = 0.1, \quad \theta_0 = \frac{\pi}{2} \quad (17)$$

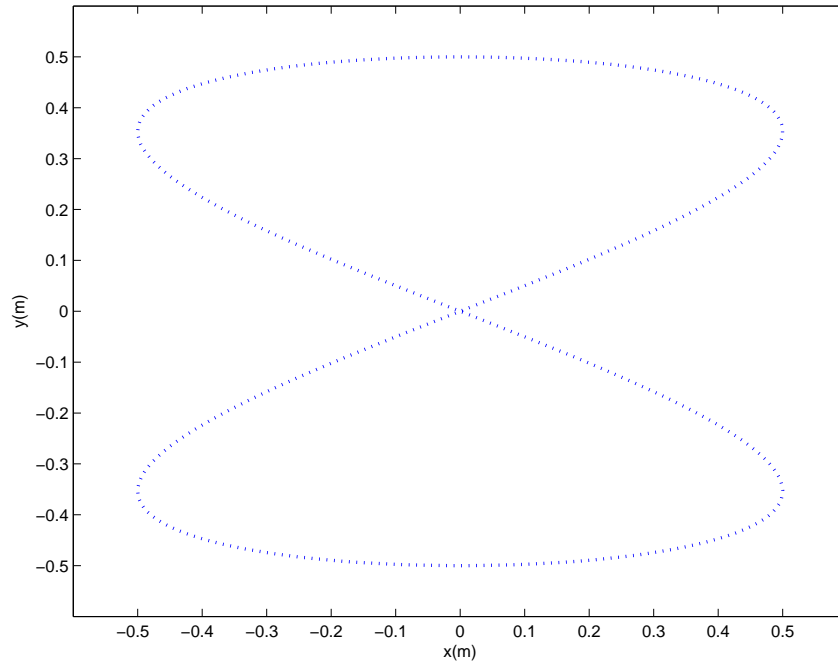


Figure 20: Reference Trajectory of The Robot

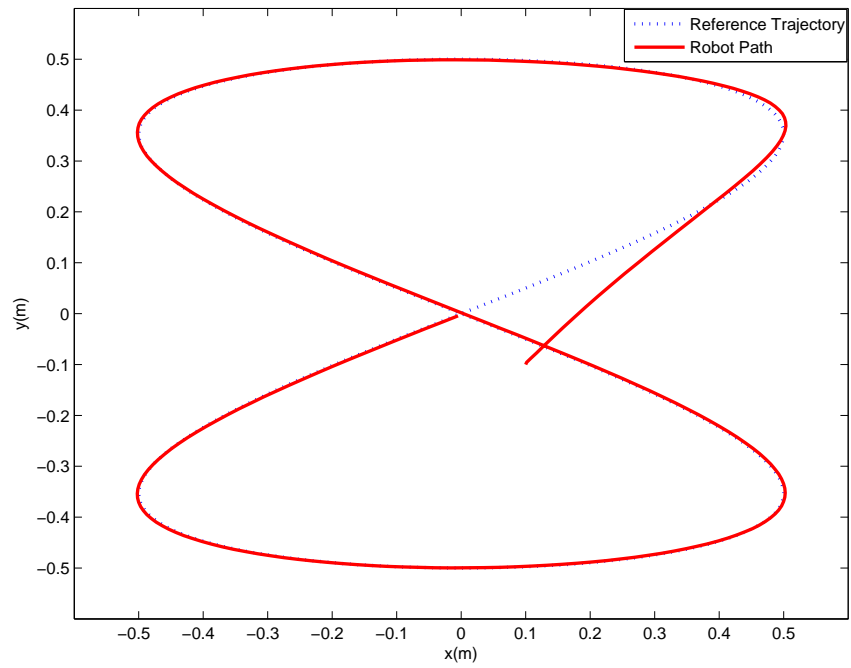


Figure 21: DFL: Reference and Robot Trajectories

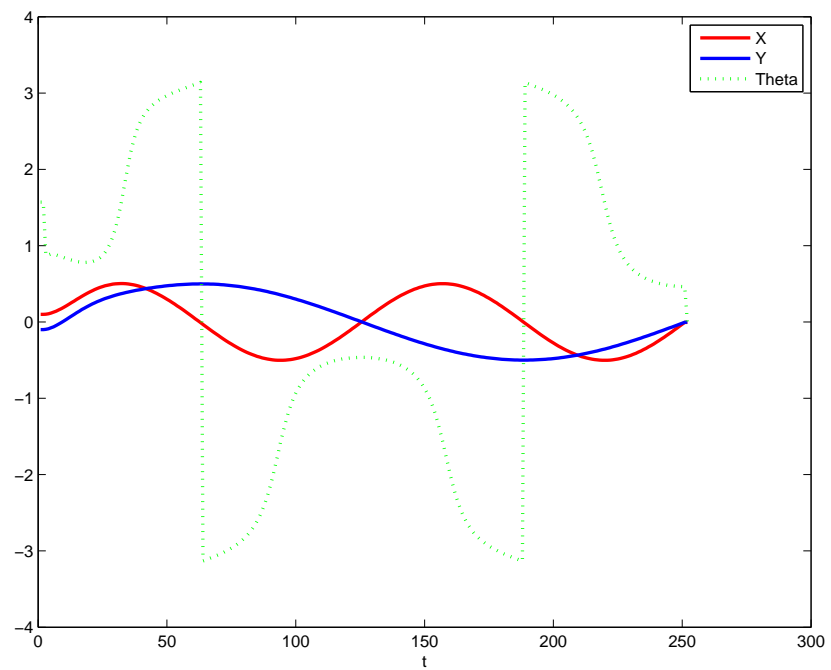


Figure 22: DFL: Robot motion: $x(m), y(m), \theta(rad)$

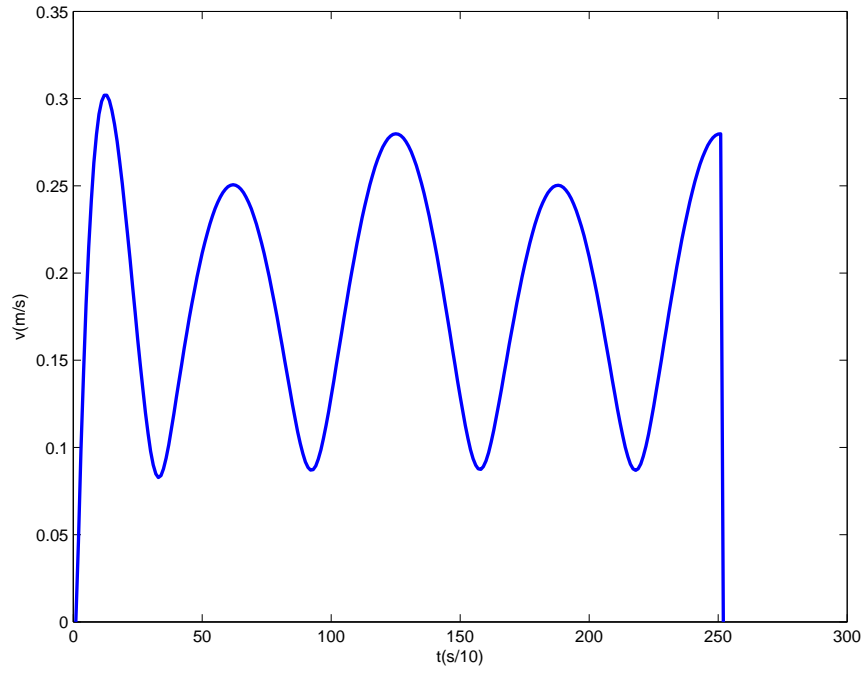


Figure 23: DFL: Linear velocity

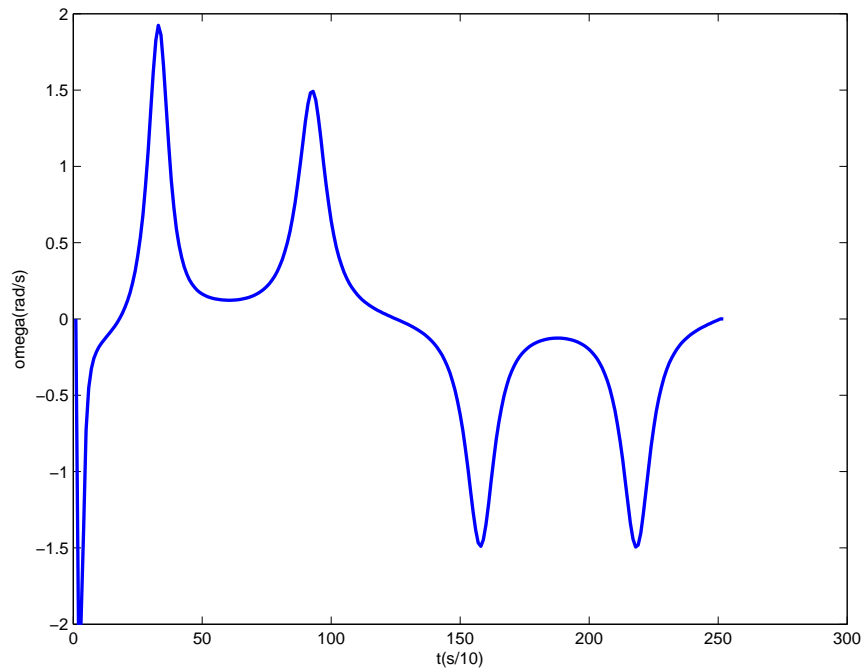


Figure 24: DFL: Angular velocity

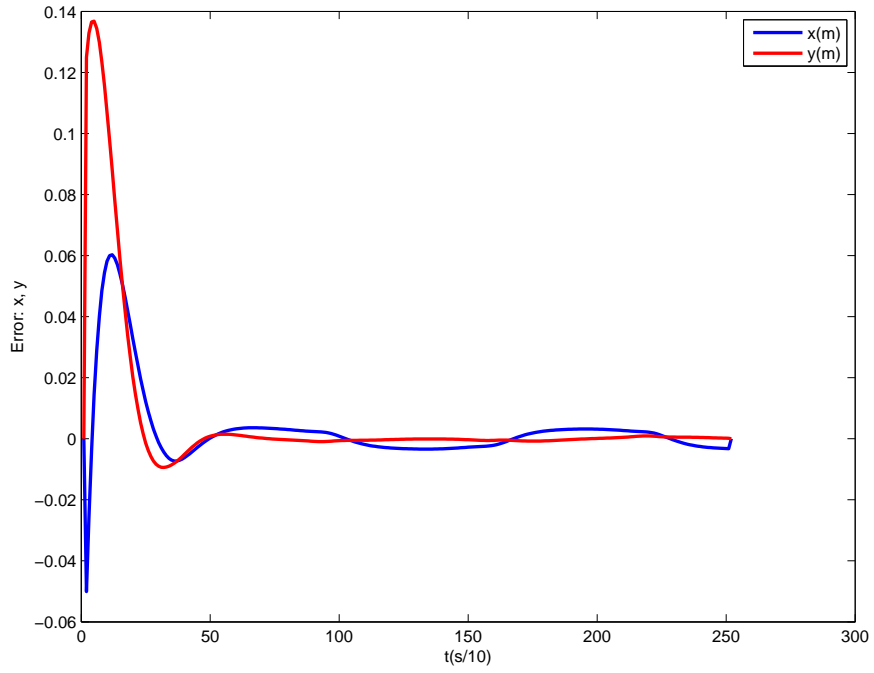


Figure 25: DFL: Cartesian error (x,y)

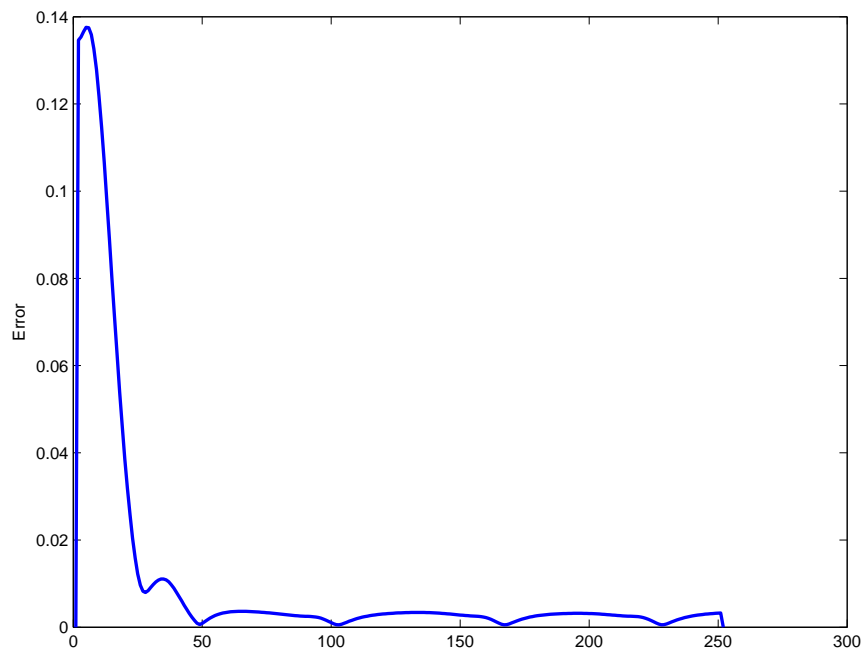


Figure 26: DFL: Norm of the error

One of the main disadvantages of using DFL is that the implementation of the control scheme is rather complex. This complexity is arisen by critical state initialization of the dynamic controller and problems when linear velocity is 0 (start and stop). Another big disadvantage of DFL method is its inability to follow trajectories with sharp edges (it is impossible with constant velocity as the robot has to stop and turn). In this case, tracking will be temporarily lost. Such behavior can be observed when we set the robot to follow a square shaped trajectory Figure 27.

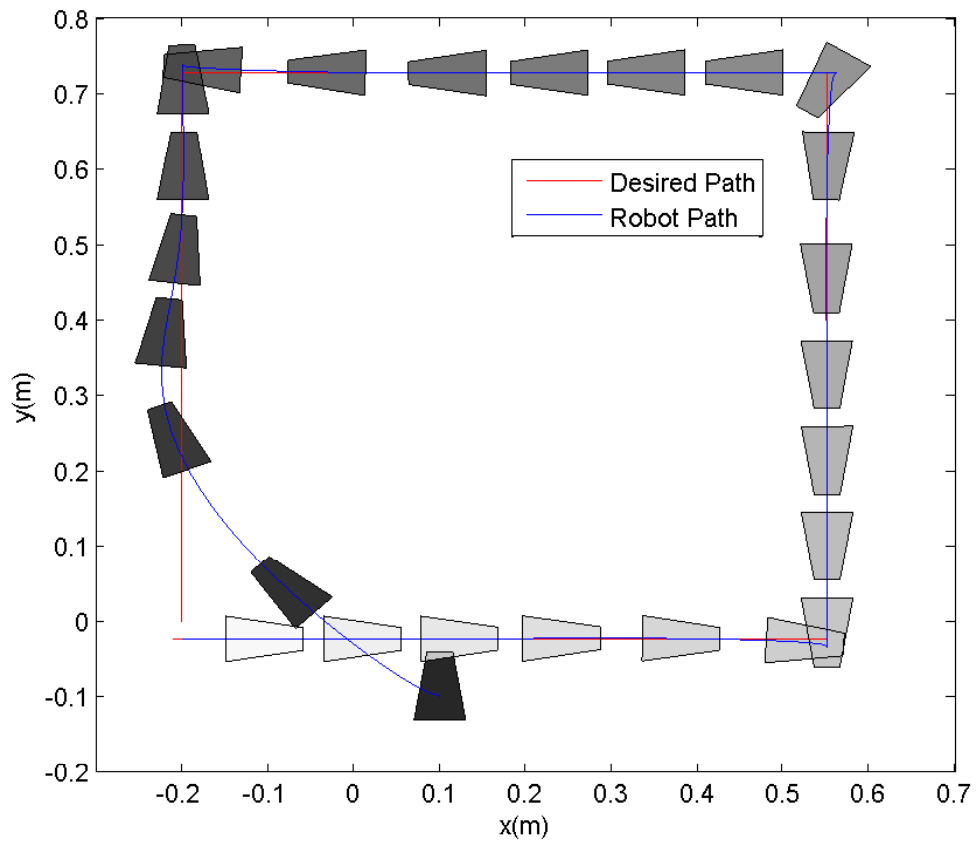


Figure 27: DFL: Robot following a square shaped trajectory.(brighter the color, the closer the robot is to the final position)

Due to temporary loss of tracking at the sharp edges of the path, the angular velocity will drastically increase (Figure 28 which results in complete change in robot heading).

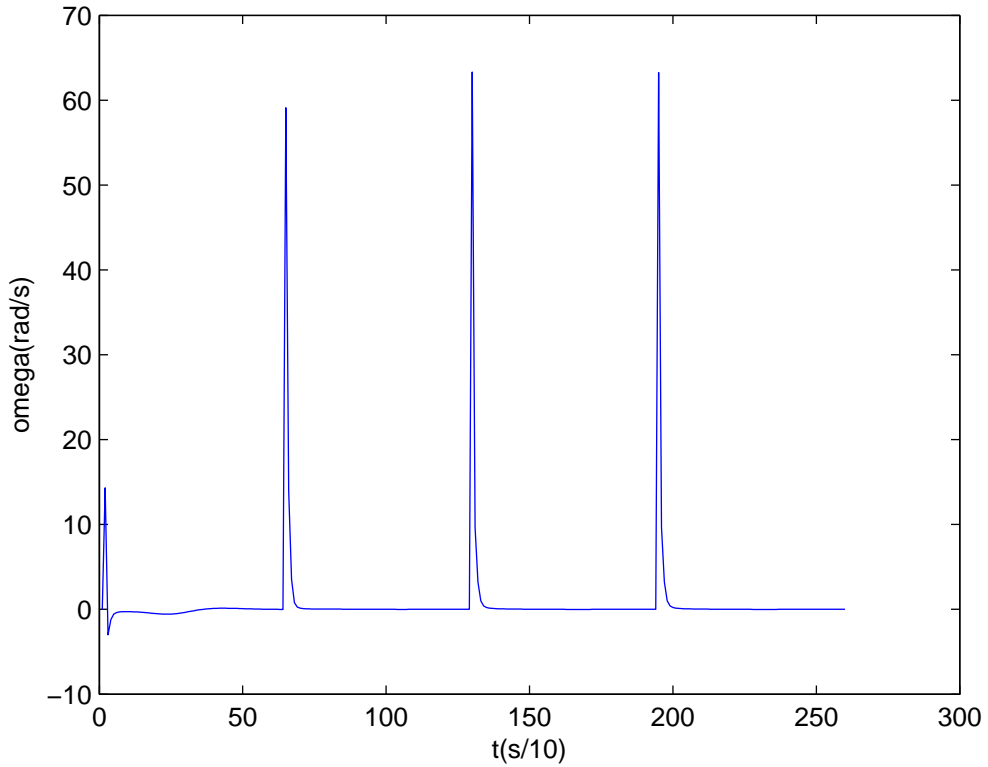


Figure 28: DFL: Angular velocity of the robot following a square shaped trajectory.

4.3. Input-Output State Feedback Linearization

The limitation of DFL to follow a path with tangent discontinuities is enough motivation to look for other solutions for the problem. By taking a point b outside the wheel axle of the unicycle model, as the output(reference point) of the system, it is possible to control the robot motion with a constant linear velocity regardless of the path curvature as it is not subjected to kinematic constraints and can move in any direction instantaneously [52].

$$\begin{aligned}x_B &= x + b\cos\theta \\y_B &= y + b\sin\theta\end{aligned}\tag{18}$$

Such solution can be illustrated by a kid pulling a toy car Figure 30:

1. The kinematic model of the robot with respect to the coordinate transformation:

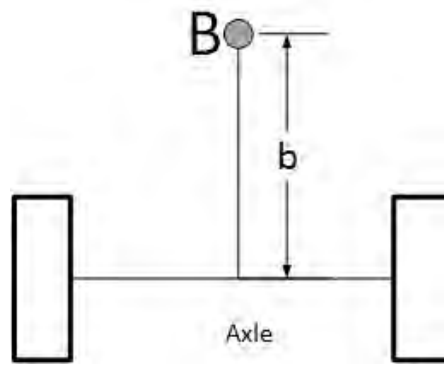


Figure 29: Point B outside the wheel axle with distance $b \neq 0$

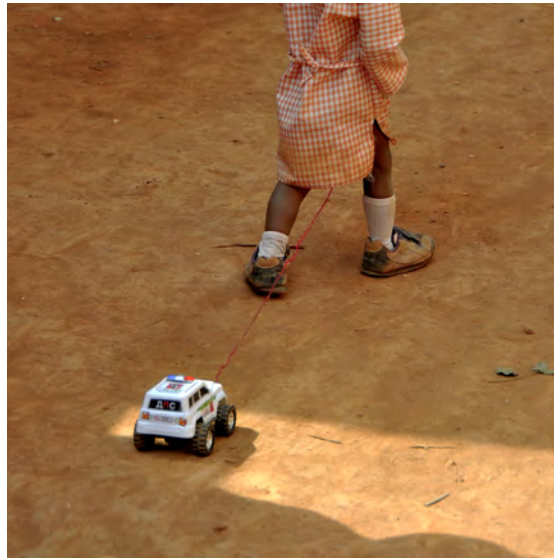


Figure 30: Pulling a toy car with a piece of rope

$$\begin{aligned}
 \dot{x}_b &= v \cos \theta - \omega b \sin \theta \\
 \dot{y}_b &= v \sin \theta - \omega b \cos \theta \\
 \dot{\theta} &= \omega
 \end{aligned} \tag{19}$$

2. The dependence on the inputs is invertible in the first two equations:

$$\det \begin{bmatrix} \cos \theta & -b \sin \theta \\ \sin \theta & b \cos \theta \end{bmatrix} = b \neq 0 \tag{20}$$

3. The resulting system is a linear system with two decoupled integrators:

$$\begin{aligned}
\dot{x}_b &= v_{dx} \\
\dot{y}_b &= v_{dy} \\
\begin{bmatrix} \dot{x}_b \\ \dot{y}_b \end{bmatrix} &= \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \\
\begin{bmatrix} v \\ \omega \end{bmatrix} &= \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix}^{-1} \begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix} \\
&= \begin{bmatrix} v_{dx}\cos\theta + v_{dy}\sin\theta \\ \frac{1}{b}(v_{dy}\cos\theta - v_{dx}\sin\theta) \end{bmatrix}
\end{aligned} \tag{21}$$

4. Given a trajectory (x_{des}, y_{des}) , it is possible to find v_{dx} and v_{dy} that guarantee the asymptotic tracking:

$$\left. \begin{aligned} v_{dx} &= \dot{x}_{des} + k_1(x_{des} - x_B) \\ v_{dy} &= \dot{y}_{des} + k_2(y_{des} - y_B) \end{aligned} \right\} \Rightarrow \begin{aligned} \dot{e}_x + k_1 e_x &= 0 \\ \dot{e}_y + k_2 e_y &= 0 \end{aligned} \quad e_x, e_y \rightarrow 0 \tag{22}$$

The controller has been tested against the same 8 shaped trajectory Figure 20 in matlab. Figure 31 shows the robot path along the reference trajectory. By observing the changes in $x(m), y(m), \theta(rad)$ in Figure 32 along with the controller outputs (Figure 33 and Figure 34), it can be concluded that the controller is effectively following the trajectory while maintaining minimum cartesian error (Figure 35).

I-O SFL is a very straightforward method for trajectory tracking. Its ease of implementation along with its performance over any trajectory with or without tangent discontinuities makes it a superior controller over DFL method. This feature improves the overall maneuverability of the robot by having the ability to make sharper turns with constant linear velocity (Figure 37). It should be noted that when using I-O SFL, we have no direct way to control the angle of the robot (θ) while following a trajectory, but instead the robot will try to correct its angle over time as it is being pulled by the point B. This behavior is more apparent in Figure 37 where the robot naturally aligns itself with the trajectory over a period of time depending on the length b .

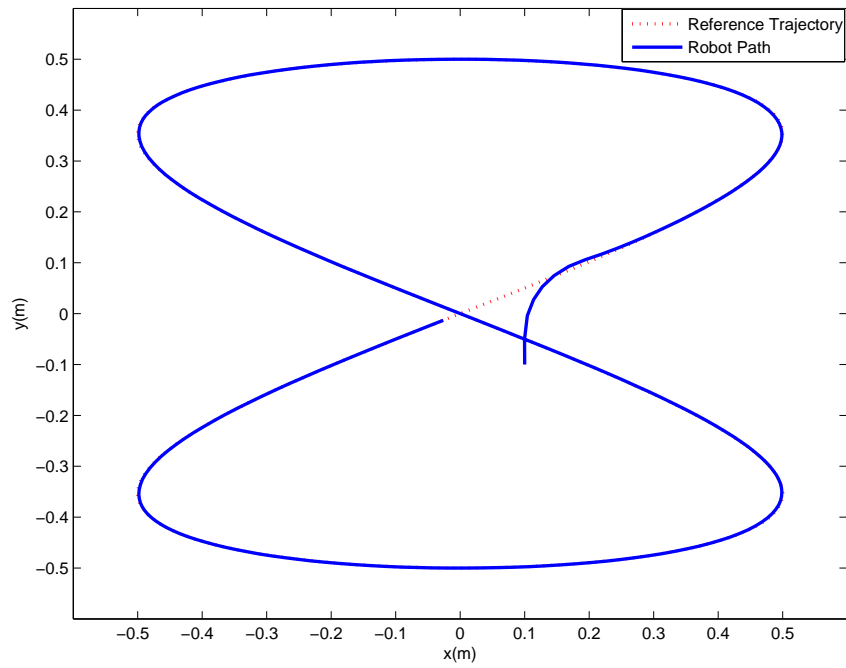


Figure 31: I-O SFL: Reference and Robot Trajectories

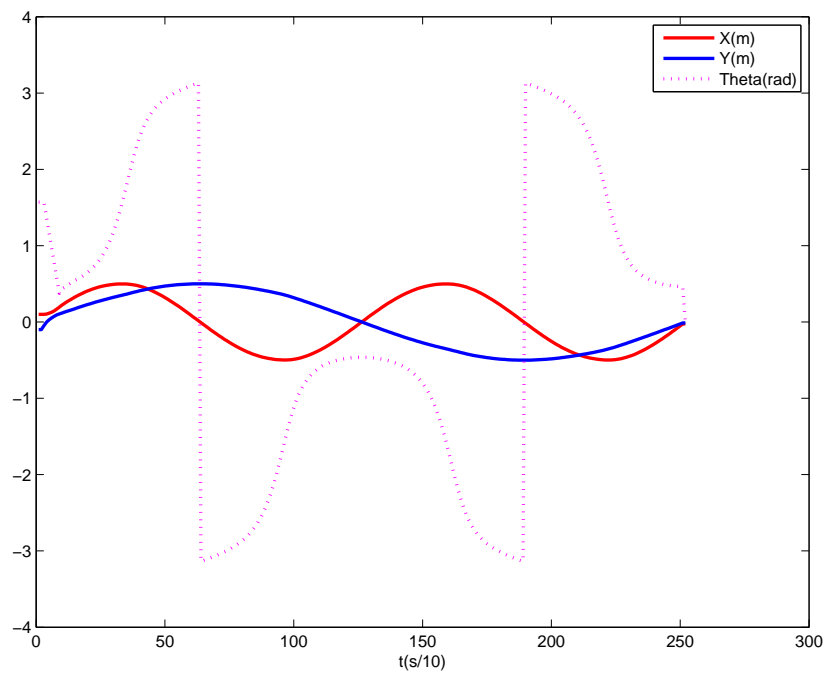


Figure 32: I-O SFL: Robot motion: $x(m)$, $y(m)$, $\theta(rad)$

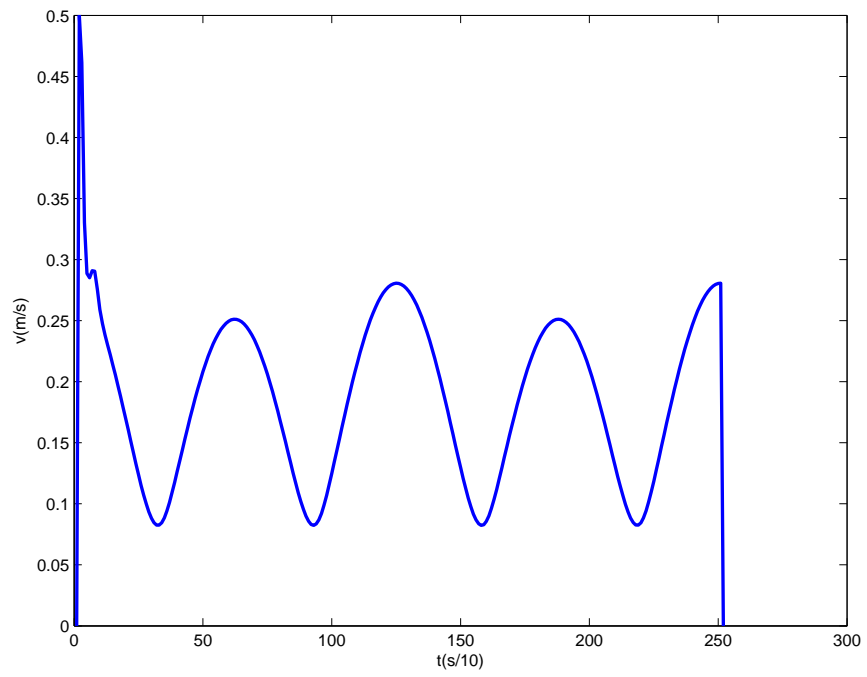


Figure 33: I-O SFL: Linear velocity

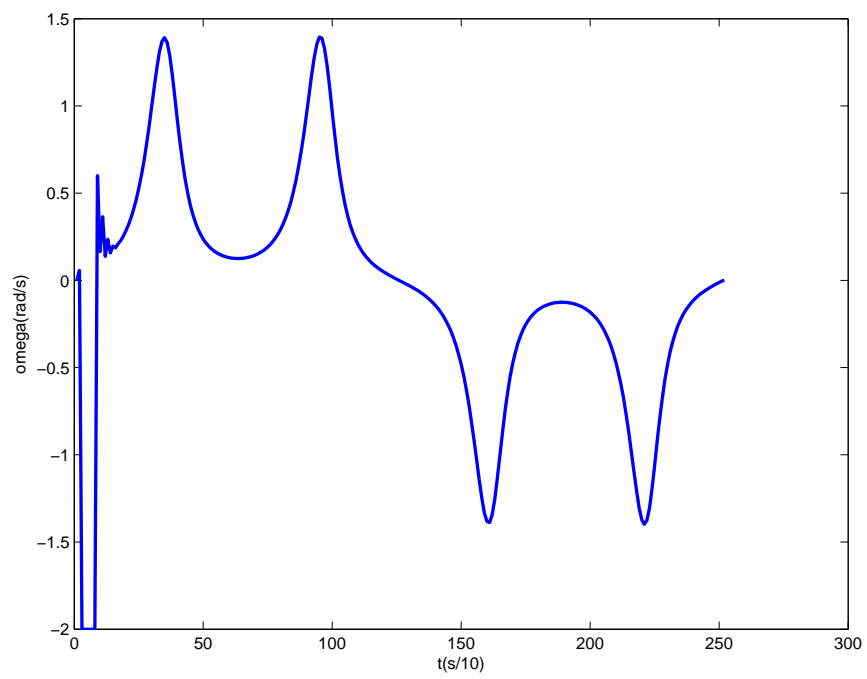


Figure 34: I-O SFL: Angular velocity

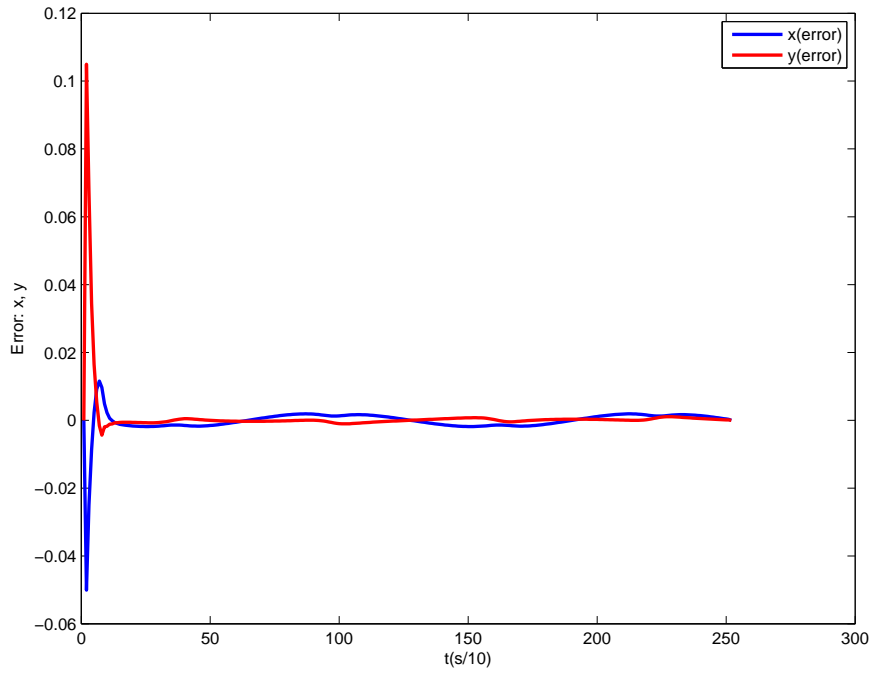


Figure 35: I-O SFL: Cartesian error (x,y)

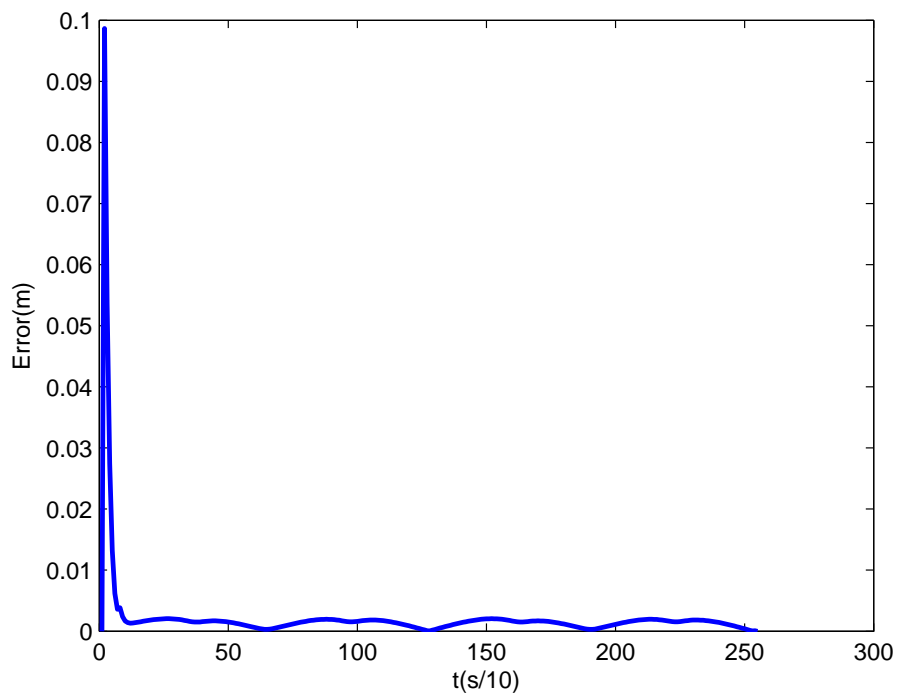


Figure 36: I-O SFL: Norm of the error

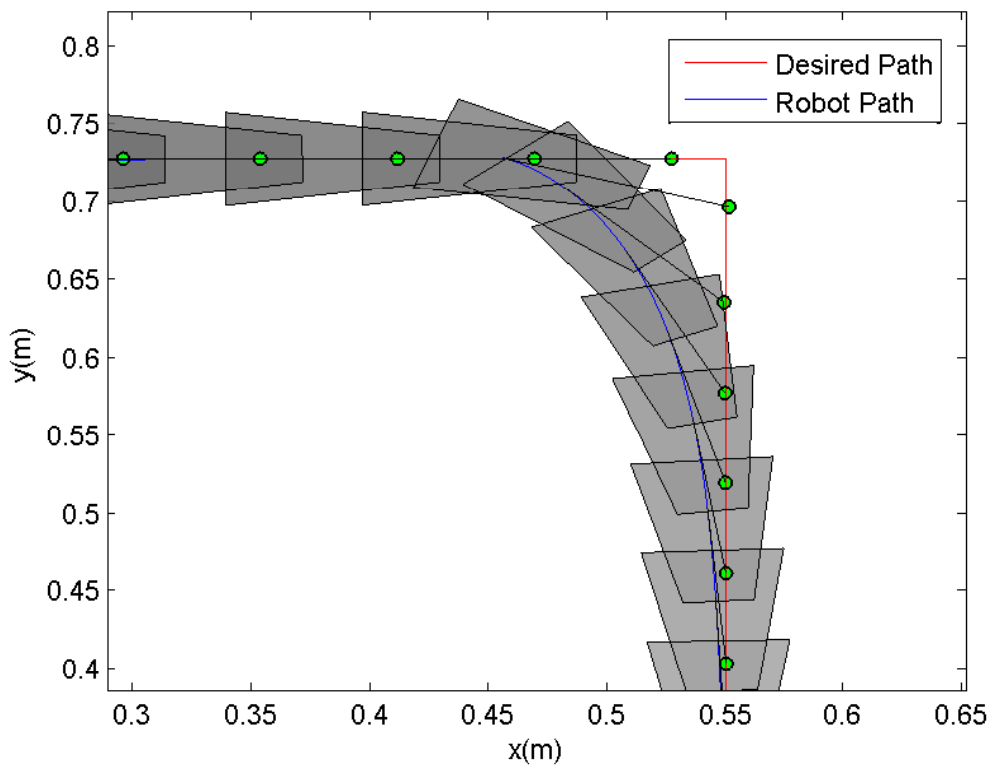


Figure 37: I-O SFL: point B (yellow) is used by the robot to track a square shaped path

Chapter 5: Outdoor Navigation

Globally navigating a mobile robot requires the ability to locate its position when placed in any unknown location and to update that information while moving to fulfill other mission objectives (in this case navigation through the waypoints). Based on the measurement method, robot localization can be divided into three categories:

- Relative localization or dead-reckoning

This method can be achieved using sensors such as wheel encoders or visual odometry with the advantage of data being always available. The problem with relative localization arises from the fact that each measurement is based on the last measurement which will result in the accumulation of the error.

- Absolute localization

In the early 60's the American Navy Navigation Satellite System (NNSS) was developed, using 'Transit' Satellites, to provide a global position fixing system for the US Navy's Polaris submarines. This system became fully operational in 1964 and was made available to the general public in 1967 by Presidential order [53]. Since then, the GPS became a de facto standard for absolute position measurements in outdoor environments. The main disadvantage of absolute localization (whether it is coming from a GPS or an active beacon) is that the data is not always available(eg. A GPS will lose signal in a forest).

- Sensor Fusion

Combining both absolute and relative measurements into one system, will result in a more accurate, more complete and more dependable localization system. Depending on the type of sensors, methods, such as KF [54] or Particle Filter, can be used to attain a reliable localization system.

KF is essentially a set of mathematical equations that implement a predictor-corrector type estimator which is optimal in the sense that it minimizes the estimated error covariance when some presumed conditions are met. Since the time of its introduction, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. This is likely due in large part to advances in digital computing that made the use of the filter practical because

of the relative simplicity and robust nature of the filter itself. Rarely do the conditions necessary for optimality actually exist, and yet the filter apparently works well for many applications in spite of this situation.

The algorithm (Figure 38) is a two-step process, in the first step(prediction), KF estimates the current state variables and their uncertainties. Once the next measurements(prone to noise and random errors) are observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The KF algorithm is computationally quite efficient. because of its recursive nature, it can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

Algorithm 5.0.1: KALMANFILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

$$\begin{aligned} \bar{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^T + R_t \\ K_t &= \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \\ u_t &= \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \\ \Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t \\ \mathbf{return} & (u_t, \Sigma_t) \end{aligned}$$

$\bar{\mu}_t$ = Predicted state estimate
 $\bar{\Sigma}_t$ = Predicted covariance estimate
 K_t = Near-optimal Kalman gain
 u_t = Updated state estimate
 Σ_t = Updated covariance estimate

Figure 38: The Kalman filter algorithm for linear Gaussian state transitions and measurements. [20]

5.1. Extended Kalman Filter

Unfortunately, in practice the assumption of linear state transitions and linear measurements with added Gaussian noise are rarely fulfilled. For example, a mobile

robot platform with differential drive and non-holonomic constraints typically moves on a circular trajectory, which cannot be described by linear next state transitions. So an attempt was made to apply this filtering technique to nonlinear systems. By adapting techniques from calculus, namely (first order) Taylor Series expansions, EKF, approximately linearize a model about a working point. The EKF algorithm in Figure 39, in many ways, is similar to KF algorithm with the most important differences gathered in Table 1

Algorithm 5.1.1: EXTENDEDKALMANFILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

$$\begin{aligned} \bar{\mu}_t &= g(\mu_t, \mu_{t-1}) \\ \bar{\Sigma}_t &= G_t \Sigma_{t-1} G_t^T + R_t \\ K_t &= \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \\ u_t &= \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) \\ \Sigma_t &= (I - K_t H_t) \bar{\Sigma}_t \\ \mathbf{return} & (u_t, \Sigma_t) \end{aligned}$$

Figure 39: The Extended Kalman filter algorithm (for full mathematical derivation of the EKF refer to [20])

Table 1: KF Vs EKF [20]

	Kalman filter	Extended KF
state prediction (Line 2)	$A_t \mu_{t-1} + B_t \mu_t$	$g(\mu_t, \mu_{t-1})$
measurement prediction (Line 5)	$C_t \bar{\mu}_t$	$h(\bar{\mu}_t)$

The linear equations in prediction step of KF are replaced by their nonlinear generalizations in EKF. In addition EKF uses jacobians G_t and H_t instead of corresponding linear matrices A_t , B_t and C_t .

For the detailed mathematical derivation of EKF refer to [20].

5.1.1. Filter Design.

The readings from the wheel encoders, IMU, and the GPS can be fused together to form a much more reliable reading which can effectively reduce the localization error. Almost every commercially available sensor is associated with some level of reading

uncertainties. In our case noisy sensors include the wheel encoders(integration error, wheel slippage, etc.), heading sensor (magnetic interference,etc.) and GPS(ionosphere and troposphere delays, number of visible satellites,etc.). The integrated system which resulted from EKF provide superior performance over either GPS or odometry based positioning system.

As an example to visualize EKF, one can imagine a human walking towards a door. As you move, your body tells your brain how far you moved from the last known position, while your eyes can tell you how far you exactly are from the door. Now go through the same process with your eyes closed. On the first step your body can tell you how much you moved with a little uncertainty, but since you have no way to measure your distance from the door, with each step this uncertainty grows. So, after a few step the best you can do is to make a rough guess about your location. In this conceptual example, the brain functioning happens in two discrete steps: 1) Prediction step (how your body moves), and 2) Measurement (Measuring the distance using your eyes). As mentioned before the EKF algorithm works in a similar two-step process by guessing the internal state of the system, using the system and measurement models. The system model is necessary to predict the system state based on the previous state. Then the estimated state is updated by measurement model, given the sensor data. The first step of the filter design is to choose the states which consists of all the parameters that need to be estimated:

$$X = \begin{bmatrix} x & y & \theta & v & \omega \end{bmatrix}^t \quad (23)$$

The motion model (8) derived in chapter 3 can be used to model the system:

$$\begin{aligned} x_k &= x_{k-1} - \frac{v}{\omega} \sin\theta + \frac{v}{\omega} \sin(\theta + \omega\Delta t) \\ y_k &= y_{k-1} + \frac{v}{\omega} \cos\theta - \frac{v}{\omega} \cos(\theta + \omega\Delta t) \\ \theta_k &= \theta_{k-1} + \omega\Delta t \\ v_k &= v_{k-1} \\ \omega_k &= \omega_{k-1} \end{aligned} \quad (24)$$

Measurement model purpose is to tell the system what parameters the sensor is measuring, there are three different sensors used for measurement in our system. So, we need three measurement models:

- **Encoder**

Encoders will give the information about total displacement of the robot since its last known position, which in our case, will be calculated by multiplying the state vector (5.2.2) with the following measurement matrix:

$$H_{enc} = \begin{bmatrix} 0 & 0 & 0 & 1 & \frac{b}{2} \\ 0 & 0 & 0 & 1 & -\frac{b}{2} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (25)$$

Where b is the distance between the wheels.

- **Heading**

Heading value will only affect θ so the measurement matrix is pretty straight forward:

$$H_{enc} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (26)$$

- **GPS**

GPS measures the location of the robot in x and y (ECEF) so if we put the GPS receiver antenna exactly on the robot axis the measurement matrix would be as simple as:

$$H_{GPS} = \begin{bmatrix} x_{gps} \\ y_{gps} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} \quad (27)$$

But in practice that is usually not the case as there would be an offset from where the antenna is to the robot axis. Using simple trigonometry we can easily consider this offset in the measurement calculations:

To generate a realistic sensor reading, measurements have been simulated by adding Gaussian noise to the true value of the measured parameter taken from the state vector. This set of simulations shows the ability of EKF to correct the robot path given the following configuration:

$$\begin{aligned}
 x_0 = y_0 = \theta_0 &= 0 \\
 v &= 0.4ms \\
 \omega &= 0.2rad/s \\
 \text{Encoder Noise} &\Rightarrow \text{Mean} \simeq 0, \quad \text{Variance} \simeq 1 \\
 \text{GPS Noise} &\Rightarrow \text{Mean} \simeq 0, \quad \text{Variance} \simeq 0.05
 \end{aligned}
 \tag{29}$$

Apart from the white noise, an error of magnitude $0.4m/s$ has been added to the value of right encoder between the 100th and 110th iteration to simulate the error caused by wheel slippage. Figure 41 shows the filter in action when only the encoder and heading data are available.

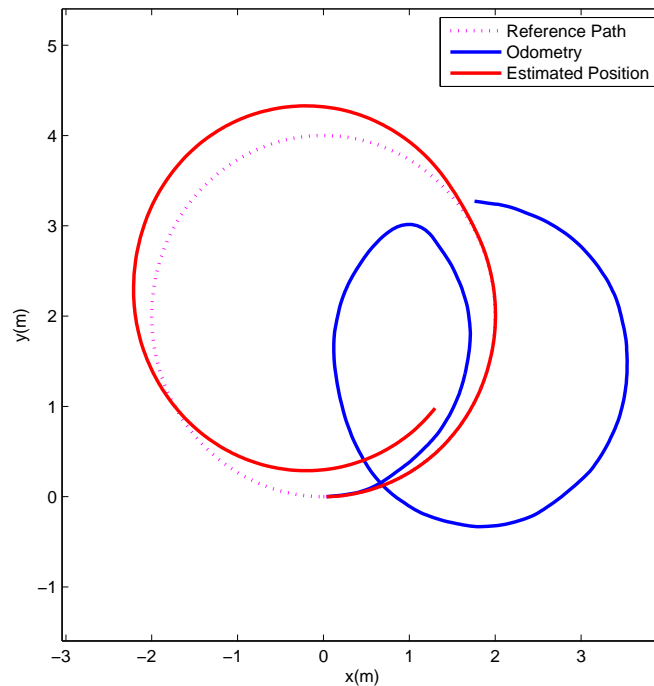
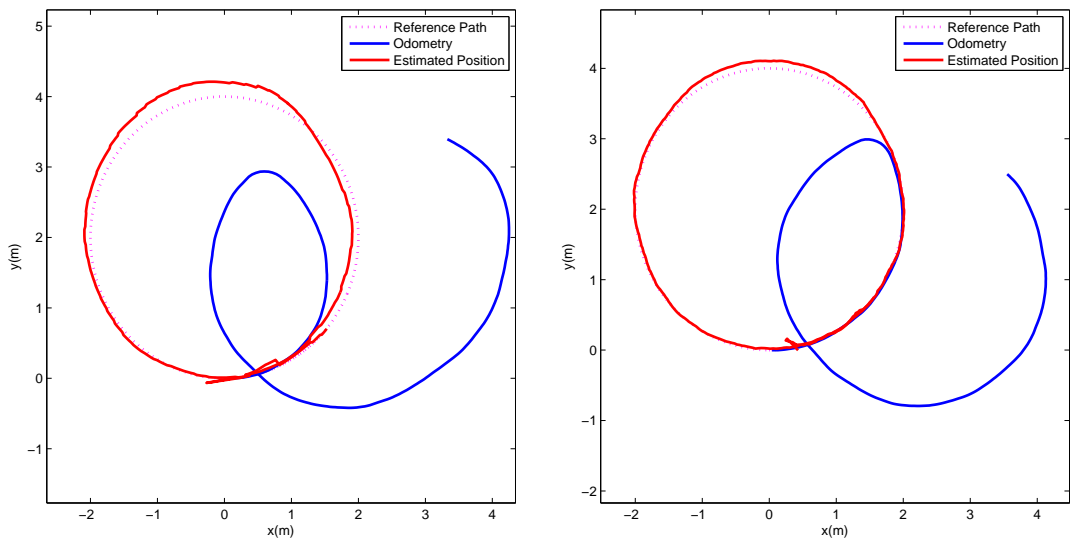


Figure 41: EKF with encoder and heading measurement

It should be noted that a measurement update step will only happen if the measurement data is available, thus in case of multiple sensors, measurements updates may happen with different rates. The next example shows how the measurement update rate can affect the filter performance.

It can be concluded from the result in Figure 42 that EKF can effectively minimize the error caused by imperfect sensors such as encoders and GPS by fusing their data together. The accuracy of the filter has a direct relation with the frequency of measurements. For example, if instead of every fifth iteration the GPS measurement happens on every single iteration, the outcome will become closer to the expected path (Figure 42b).



(a) GPS update on every fifth iteration (b) GPS update on every iteration
 Figure 42: EKF with encoder, heading and noisy GPS measurements

In the next simulation, estimated data from EKF is used as the input to the trajectory following algorithm (I-O SFL) using the same configuration defined in Equation (29) to follow a square shaped trajectory (Figure 43). The robot motion can be observed in Figure 44 as it moves through the path.

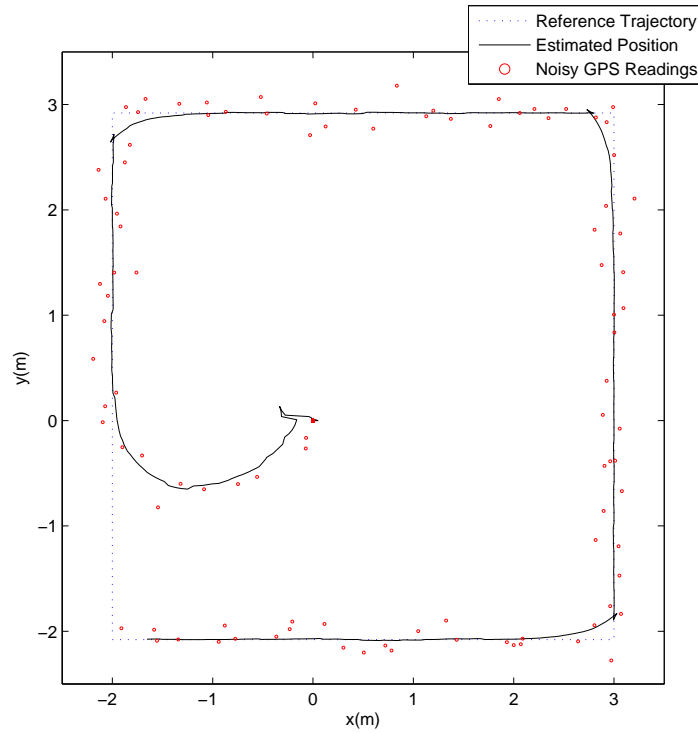


Figure 43: EKF is used with the I-O SFL algorithm to follow the square shaped trajectory

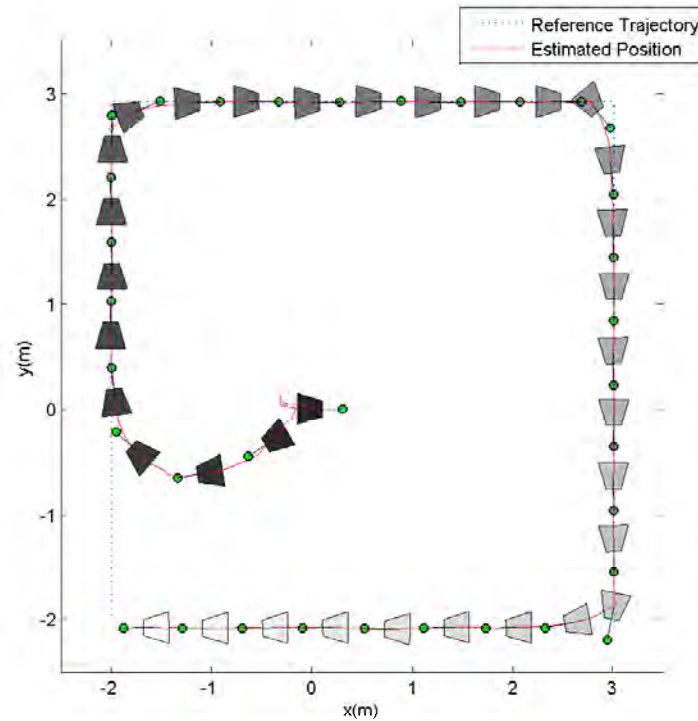


Figure 44: Overall robot pose along the path(robot's color becomes brighter as it gets closer to the end of the path)

5.2. Navigation Software

5.2.1. GUI.

The main Graphical User Interface (GUI) is divided into two tabs; the first tab, as shown in Figure 45 displays the essential information regarding the sensors such as wheel velocities (given by the encoders) and IMU/GPS data. It also includes a graphical representation of the robot in a gridded environment that is useful for observing the robot behavior during the navigation process. Highlighted in blue is the information about the serial port configuration, likewise red is the encoder data, magenta shows the IMU/GPS data with the artificial horizon and black represents EKF and navigation configuration.

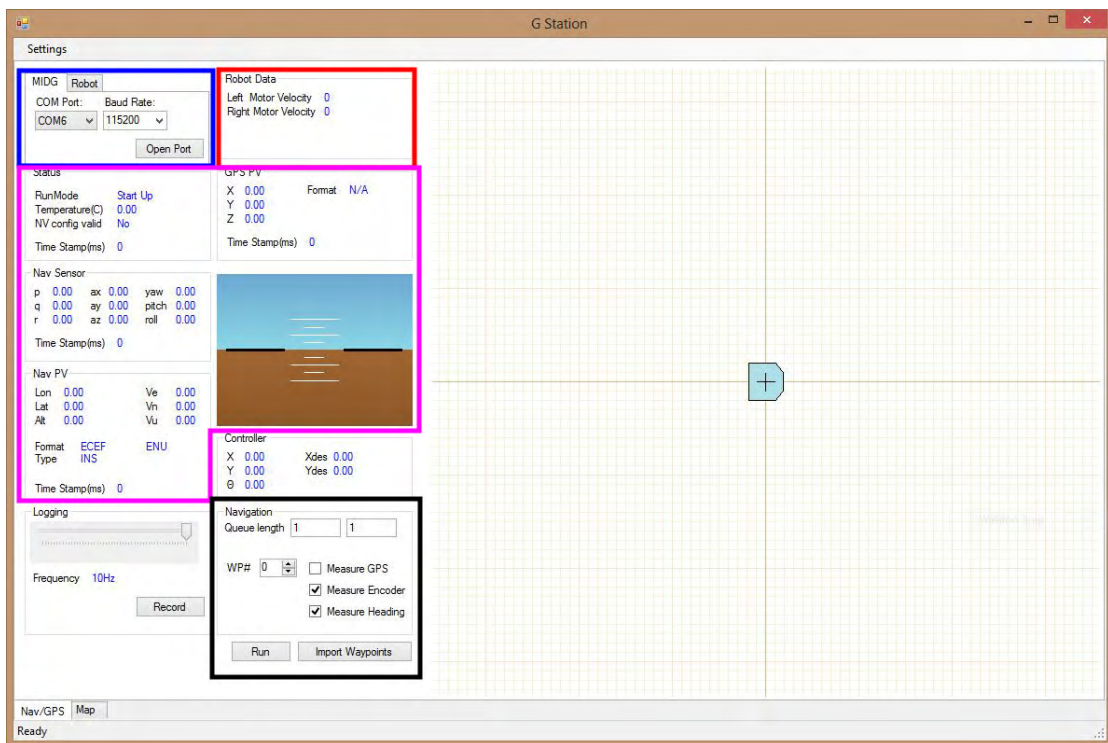


Figure 45: Main features of the GUI

The second tab provides the user with the option to choose waypoints by clicking on a zoomable map provided by a list of most popular online map providers Figure 46

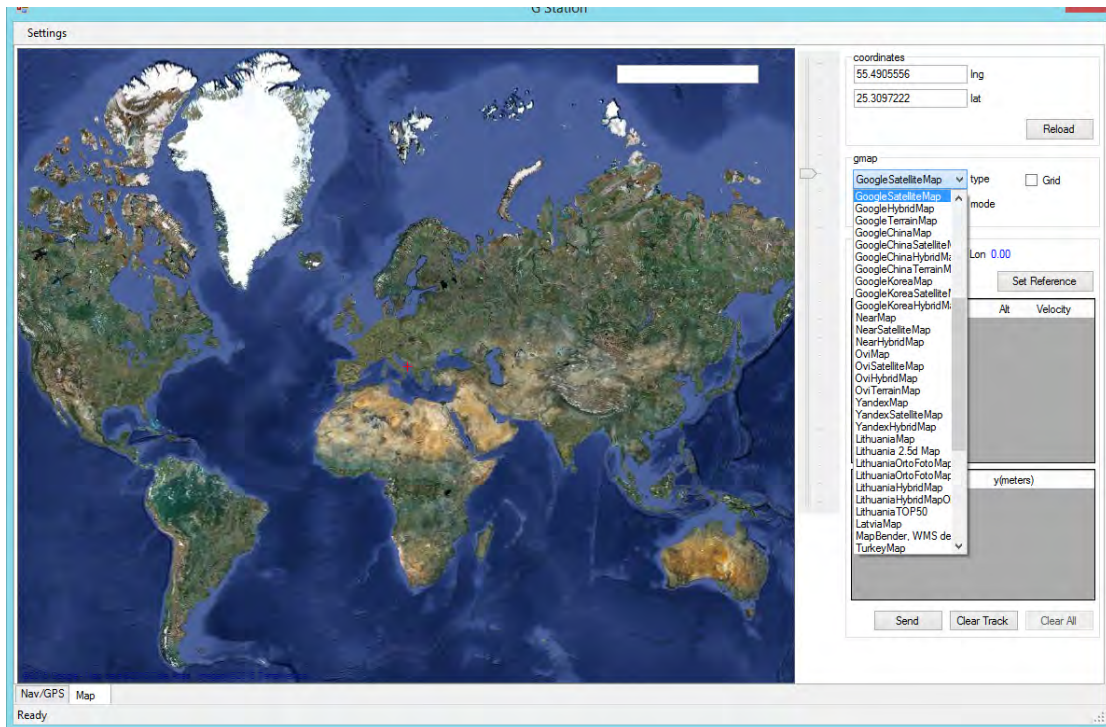


Figure 46: Map features include a list of map providers, local caching of the map, zooming and panning)

5.2.2. Geodetic Transformation.

A Geodetic system or geodetic datum is defined as 'A set of constants specifying the coordinate system for a collection of points on the Earth surface.' [55].

Datums are used to transform positions indicated on maps to their real position on Earth and vice versa. For satellite geodesy, a global geodetic datum is defined. For example World Geodetic System 1984 (WGS84) is an Earth-centered, Earth-fixed terrestrial geodetic datum which is based on a consistent set of constants and model parameters that describe the Earth's size, shape, and gravity and geomagnetic fields. WGS84 is the standard U.S. Department of Defense definition of a global reference system for geospatial information which is the reference system for the Global Positioning System (GPS) [56].

The process of waypoint selection encompasses a set of coordinate transformations that allows the software to translate the coordinates chosen by the user to their respective geodetic coordinates and vice versa. This process starts by converting the local (x,y) coordinate of the mouse cursor to its intended geodetic location (Latitude-Longitude-Altitude (LLA)) on the map. The main idea behind this conversion is to save

the geodetic coordinate of each pixel on the time of map rendering in an array so it can be accessed later. This functionality is provided by GMap.Net control ¹.

The selected point in LLA coordinate system is then converted to the Earth-Centered, Earth-Fixed (ECEF) Cartesian coordinate system using the WGS84 parameters (Figure 47).

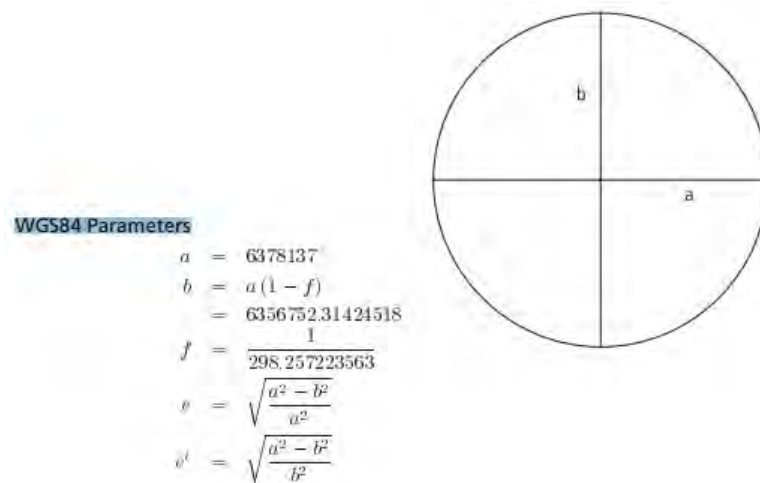


Figure 47: WGS84 Ellipsoid Parameters [57]

This LLA to ECEF conversion(in meters) is performed using the closed formulas shown below.

$$\begin{aligned} X &= (N + h)\cos\varphi\cos\lambda \\ Y &= (N + h)\cos\varphi\sin\lambda \\ Z &= \left(\frac{b^2}{a^2}N + h\right)\sin\varphi \end{aligned} \quad (30)$$

φ = latitude

λ = longitude

h = height above ellipsoid(meters)

N = radius of curvature(meters), defined as: $= \frac{a}{\sqrt{1 - e^2\sin^2\varphi}}$

¹More information can be found at: <https://greatmaps.codeplex.com>

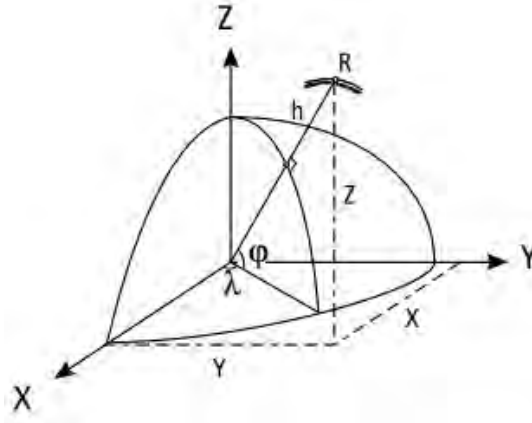


Figure 48: ECEF and Reference Ellipsoid [57]

Finally, the ECEF coordinate is further transformed into what will be termed here the Local Tangent Plane (LTP). This is an orthogonal, rectangular, reference system defined with its origin at an arbitrary point on the Earth's surface. This system is written as e, n, u and forms a right-handed coordinate system with a strong analogy to the usual x, y, z coordinates. The great advantage of the LTP system is that its axes coincide with the expectation of people on the ground concerning such ingrained things as up, and north, which ECEF coordinate system does not [58]. The ECEF to LTP transformation is involved with a set of simple rotations and translations:

$$X_{enu} = \begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\cos\lambda \sin\varphi & -\sin\lambda \sin\varphi & \cos\varphi \\ \cos\varphi \cos\lambda & \cos\varphi \sin\lambda & \sin\varphi \end{pmatrix} \cdot \begin{pmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{pmatrix} = R_t(X - X_0) \quad (31)$$

where

λ = origins latitude

φ = origins longitude

X_0 = origin in ECEF coordinate

Having a reference point as the origin of the LTP is essential to this transformation. Thus, the software restricts the user from selecting the waypoints before choosing a reference point on the map. An example of such selection is shown in Figure 49.

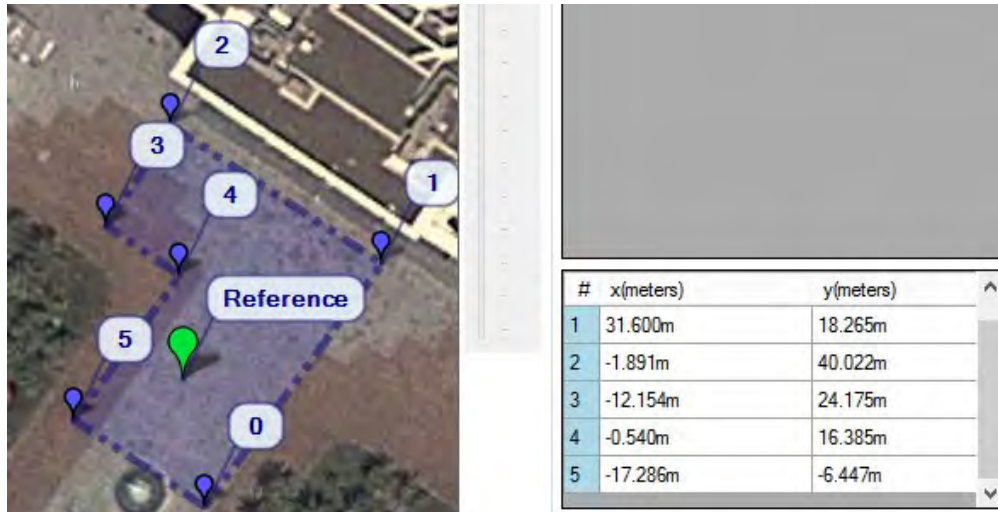


Figure 49: A screen shot of a portion of the GUI.

The accuracy of this transformation can be confirmed using the Google Maps online map. The top part of the Figure 50 shows the position of a point on the map with respect to the selected reference point. The calculated distance ($\sqrt{-93.316^2 + 61.781^2} \approx 111.91m$) is very close to the Google Map result(111.83m) shown on the bottom.

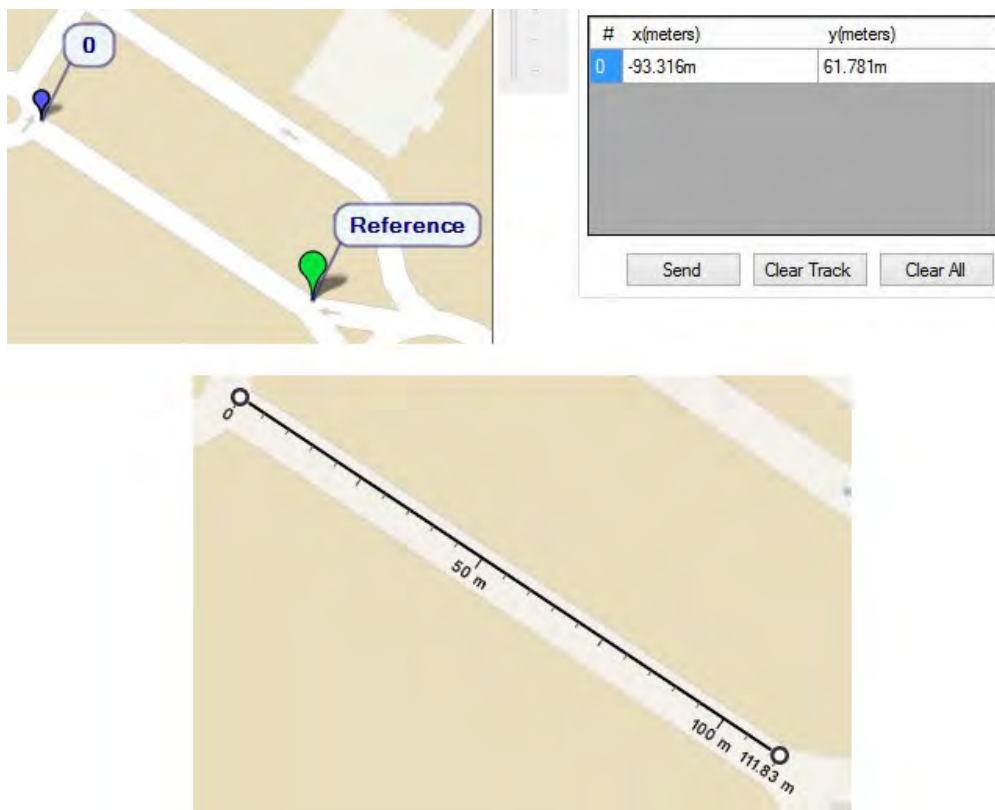


Figure 50: Conversion result from the software compared with the same conversion from google map engine.

5.3. Practical Results

To evaluate the outdoor navigation system, it has been tested in a range of outdoor environments, applying different trajectories. The recorded data from the robot was used in Matlab to create the plots to evaluate the results. No path planning algorithm was used in this experiment. Only point to point navigation with the condition that if the robot is within 40cm of the way-point, the robot will consider that it reached the target waypoint and will switch the target to the next way-point(Figure 52). The 40cm way-point boundary that is used in the following experiment is approximately equal to the size of the robot platform. By defining such boundary, the navigation process will be handled faster. By reducing the radius of this boundary the robot will slow down and will try to reach the exact location of each waypoint before moving to the next one. This tolerance is adjustable in the software based on the navigation objective. This simple algorithm can be observed in Figure 51.

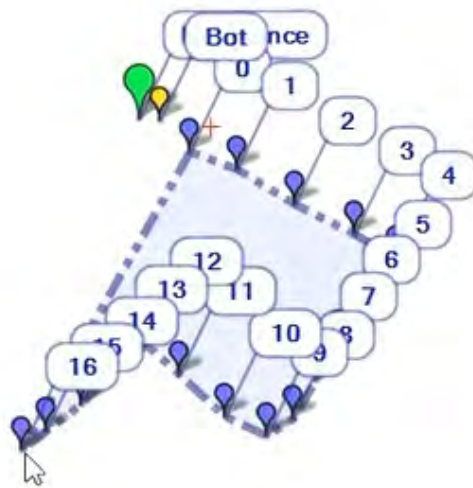
Algorithm 5.3.1: TARGETWAYPOINT($x_{estimate}, y_{estimate}, x_{desired}, y_{desired}$)

if $\left(\sqrt{(x_{estimate} - x_{desired})^2 + (y_{estimate} - y_{desired})^2} \leq 0.4 \right)$
then return (*NextWaypoint*)

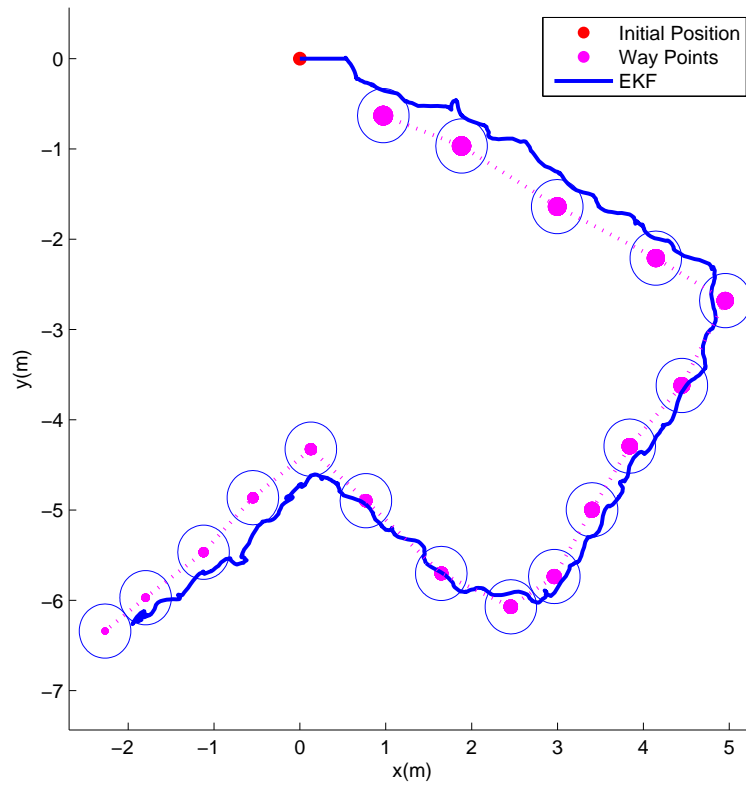
else return (*CurrentWaypoint*)

Figure 51: A simple algorithm that checks if the robot has reached the target way-point

All the variables were recorded during the experiment and used to create Matlab plots to analyze the result. Figure 54 shows the norm of the Cartesian error from the experiment in Figure 52. The dotted line shows the time when the robot changes its destination way-point after reaching the current way-point. The error values at each way-point are gathered in Table 2 and Table 3. The effectiveness of EKF can be observed in Figure 53 by comparing the noisy GPS readings with the output of the filter.



(a) A set of way-points chosen by the user on the software's map.



(b) Final result plotted in Matlab using recorded data.

Figure 52: Outdoor navigation using a set of 17 way-points

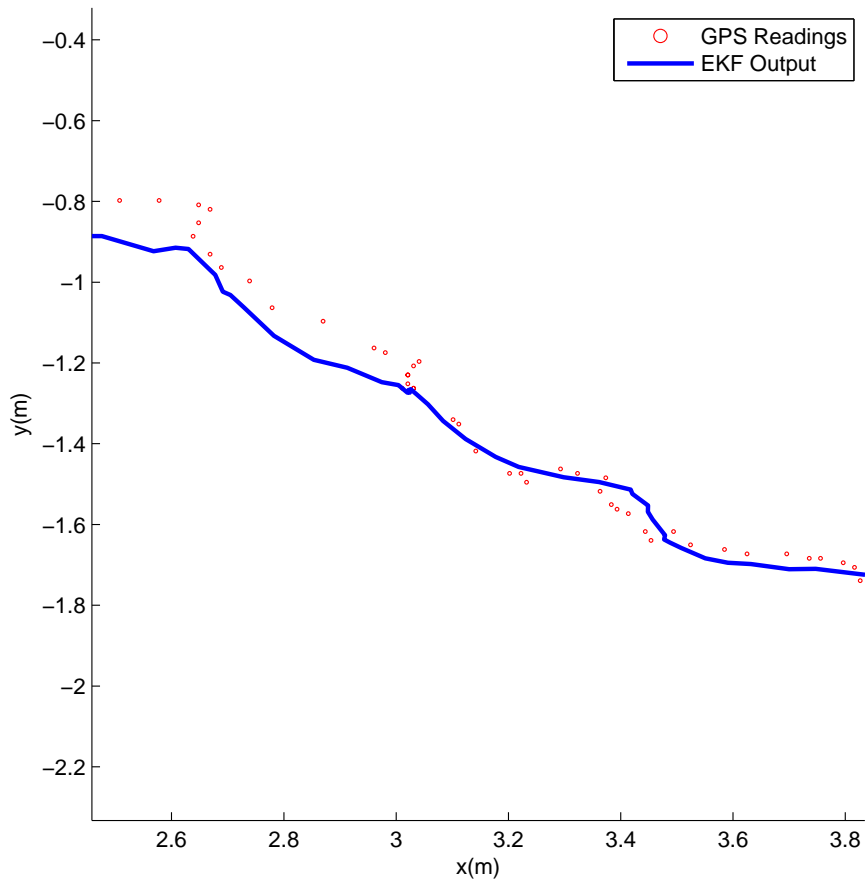


Figure 53: EKF output versus GPS output.

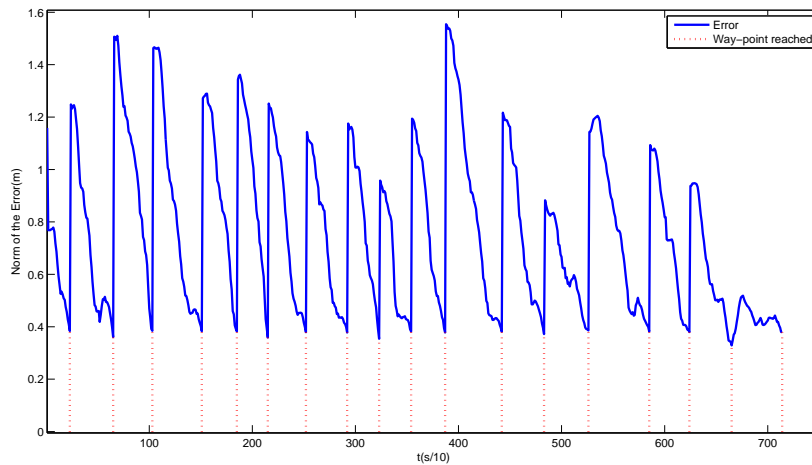


Figure 54: Norm of the Cartesian error: the dotted lines shows the error in time when the robot reaches the current way-point and changes its destination to the next one.

Table 2: Outdoor navigation error: way-points 0 to 7

Way-point #	0	1	2	3	4	5	6	7
Error in x[m]	0.16	0.01	-0.01	0.15	0.16	-0.27	-0.35	-0.34
Error in y[m]	-0.35	-0.36	-0.38	-0.35	-0.35	-0.24	-0.15	-0.17
Error Norm[m]	0.38	0.36	0.38	0.38	0.38	0.36	0.38	0.38

Table 3: Outdoor navigation error: way-points 8 to 16

Way-point #	8	9	10	11	12	13	14	15	16
Error in x[m]	-0.3	-0.37	-0.33	-0.22	-0.19	-0.39	-0.36	-0.38	-0.37
Error in y[m]	-0.19	-0.09	0.19	0.31	0.32	0	0.13	-0.03	-0.09
Error Norm[m]	0.35	0.38	0.38	0.38	0.37	0.39	0.38	0.38	0.38

Figure 55 and Figure 56 shows the outdoor navigation conducted with a set of way points that forms a circular path. The errors shown in Tables 2, 3, 4 and 5 are calculated by subtracting the robot pose estimated by the EKF from the desired way-point, at the moment when the control point b (refer to section 4.3) reaches the way-point. Since the control point b is located outside the robot axis the minimum possible error is equal to the distance from the robot axis to the point b which in this case is 0.45 meters.

The repeatability of the outdoor navigation solution is hindered by the accuracy of the GPS. This means that if we make the robot run through the same set of way-points twice, although the position error with respect to the waypoints is bounded in the algorithm, but due to the limited accuracy of the GPS the waypoints chosen on the map may not exactly point to the same location on earth.

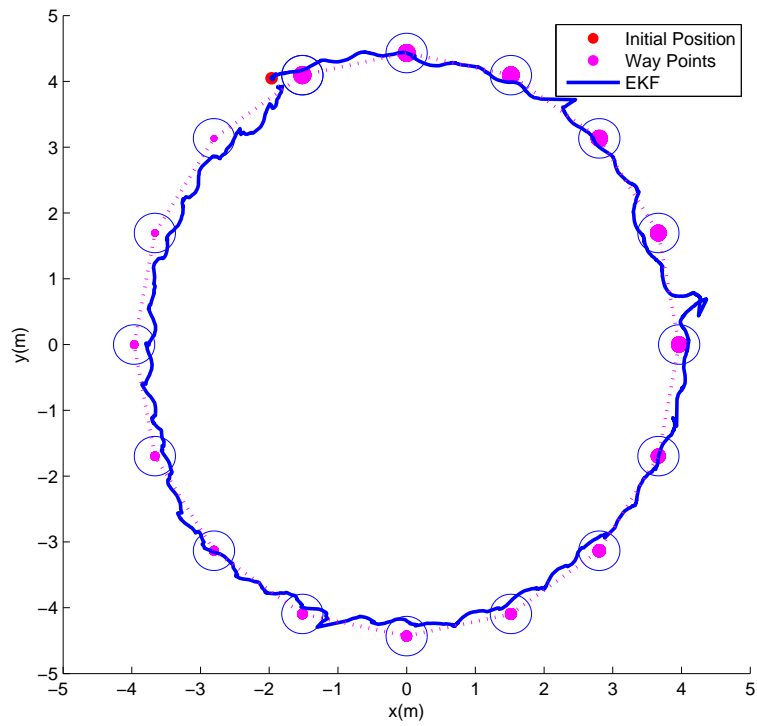


Figure 55: Outdoor Navigation with a circular path

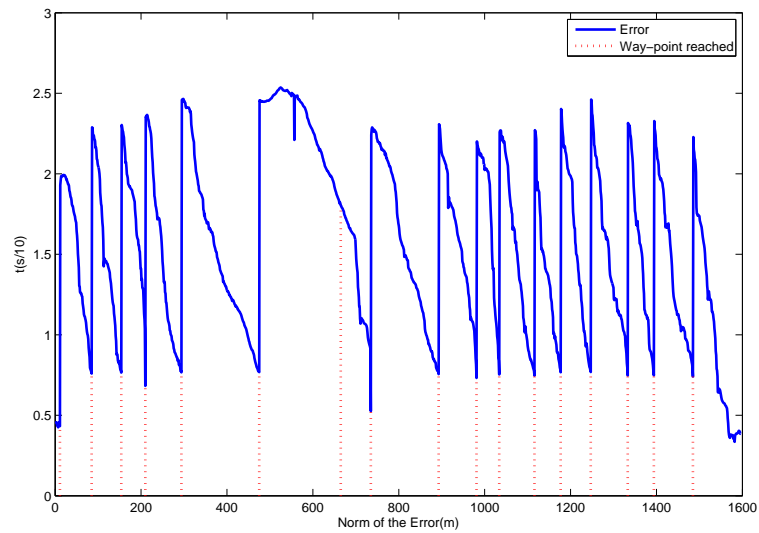


Figure 56: Norm of the Cartesian error: the dotted lines shows the error in time when the robot reaches the current way-point and changes its destination to the next one.

Table 4: Outdoor navigation(circular path) error: way-points 0 to 7

Way-point #	0	1	2	3	4	5	6	7
Error in x[m]	0.43	0.76	0.75	0.35	0.32	0.11	-0.26	-0.41
Error in y[m]	0.05	0.04	-0.16	-0.59	-0.70	-0.76	-0.45	-0.63
Error Norm[m]	0.43	0.76	0.77	0.68	0.77	0.77	0.53	0.76

Table 5: Outdoor navigation(circular path) error: way-points 8 to 15

Way-point #	8	9	10	11	12	13	14	15
Error in x[m]	-0.54	-0.73	-0.74	-0.61	-0.36	-0.18	0.01	0.23
Error in y[m]	-0.49	-0.19	0.07	0.47	0.68	0.73	0.75	0.71
Error Norm[m]	0.73	0.75	0.75	0.77	0.77	0.75	0.75	0.74

Chapter 6: Indoor Navigation

The navigation solution that was discussed in the last chapter relies on position information received by the GPS. GPS is a space-based navigation system which needs an unobstructed line of sight to four or more GPS satellites [59]. Any such system would be rendered useless once it is inside an indoor environment. Hence, a separate solution has been proposed in this chapter that can accurately estimate the robot location, provided a predefined 2D map of the environment. This solution is able to solve global localization problem, which means no prior knowledge of the robot states is needed, and in some instances, it is able to solve the kidnapped robot problem. This approach is called the MCL algorithm, also known as particle filter localization comparatively easy to implement .

6.1. Monte Carlo Localization (MCL)

MCL algorithm (Figure 60), despite its relatively short existence, is arguably one of the most popular algorithms in robotics. It tends to work well across a range of localization problems and it is comparatively easy to implement [20]. Our goal in robot localization is to estimate the state of the robot at current time-step, given information about the initial state and all measurements up to the current time. The basic MCL algorithm (shown in Figure 57) represents the $bel(x_t)$ by a set of M particles $\chi^t = \{x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}\}$. Line 1 in algorithm 57 samples from the motion model, using particles from present belief as starting points. The beam measurement model is then applied in line 2 to determine the importance weight of that particle. The initial belief $bel(x_0)$ is obtained by randomly generating M such particles from the prior $p(x_0)$ distribution, and assigning the uniform importance factor M^{-1} to each particle [20]. The functions motion model, is implemented by the motion model derived in Chapter 3, while the measurement model implementation is discussed in the next section of this chapter.

Figure 58 illustrates one iteration of MCL. The top row of the figure shows the exact density, whereas the lower panel shows the particle-based representation of that

Algorithm 6.1.1: $\text{MCL}(\chi_{t-1}, u_t, z_t, m)$

```

 $\bar{\chi}_t = \chi_t = \emptyset$ 
for  $m \leftarrow 1$  to  $M$ 
  do  $\begin{cases} x_t^{[m]} = \text{SAMPLE\_MOTION\_MODEL}(u_t, x_{t-1}^{[m]}) & (1) \\ w_t^{[m]} = \text{MEASUREMENT\_MODEL}(z_t, x_t^{[m]}, m) & (2) \\ \bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle \end{cases}$ 
for  $m \leftarrow 1$  to  $M$ 
  do  $\begin{cases} \text{draw } i \text{ with probability } \propto w_t^{[i]} \\ \text{add } x_t^{[i]} \text{ to } \chi_t \end{cases}$ 
return  $(\chi_t)$ 

```

Figure 57: Monte Carlo Localization Algorithm [20]

density. Column (a) shows a cloud of particles representing the uncertainty about the robot position. In this example, the robot position is fairly localized, but its orientation is still unknown. Column (b) shows what happens to our belief state after we are commanded the robot to move exactly one meter since the last time-step. We now know the robot to be somewhere on a circle of 1-meter radius around the previous location. Column (c) shows what happens when the robot observes a landmark, half a meter away, somewhere in the top-right corner: the top panel shows the likelihood of the posterior density and the bottom panel illustrates how each sample is weighted according to this likelihood. Finally, the last column (d) shows the effect of resampling from this weighted set, and this forms the starting point for the next iteration [60].

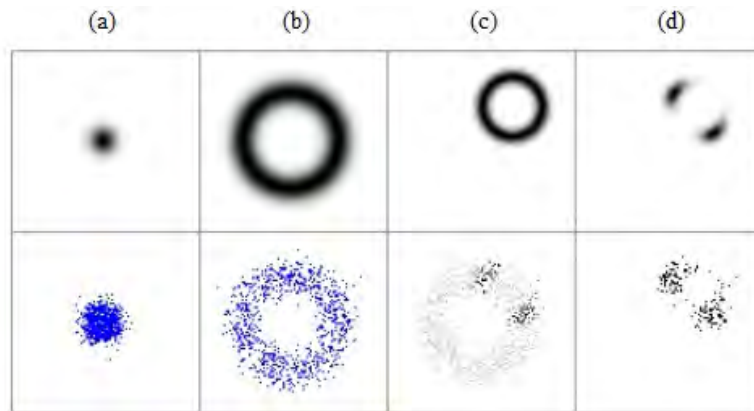


Figure 58: The probability densities and particle sets for one iteration of the algorithm [60].

The MCL algorithm in its present form, is able to solve global localization problem but it can not recover from robot kidnapping problem. Such problem is quite obvious in Figure 59: As the position is acquired, particles at places other than the most likely pose gradually disappear. At some point, particles only 'survive' near a single pose, and the algorithm is unable to recover if this pose happens to be incorrect. Shown in yellow is the true location of a robot operating in a room. Due to the symmetry that exists in the map, there exist an ambiguity about the true location of the robot. In this particular example all the particles are concentrated in a location that gives the exact measurements as the true location of the robot which happens to be incorrect.

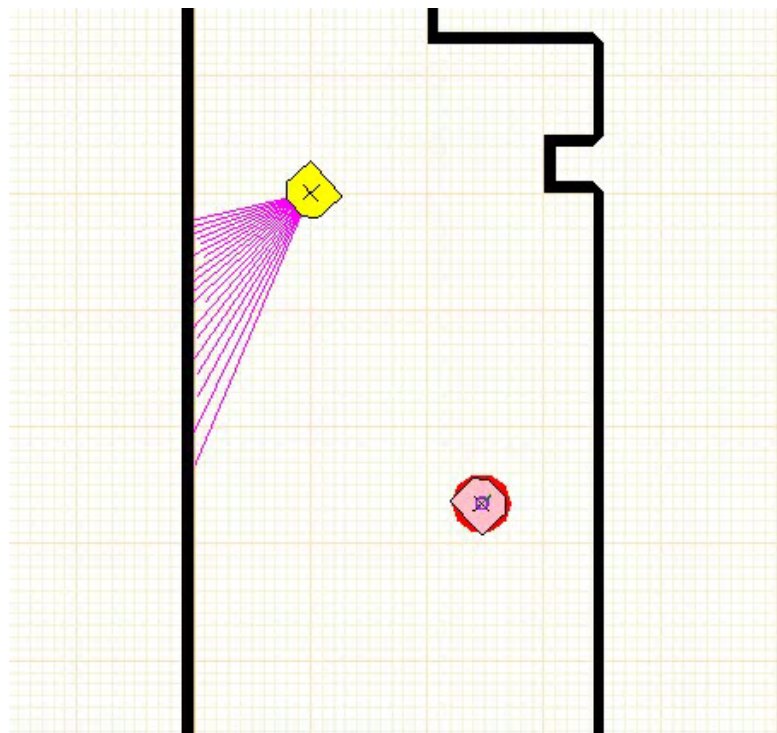


Figure 59: Kidnapped robot problem

Fortunately, this problem can be overcome by injecting random particles into the particle sets. This process can be mathematically justified, by assuming the probability that the robot might get kidnapped, therefore generating a fraction of random states in the motion model. Even if the robot does not get kidnapped, the random particles add an additional level of robustness. This solution, presented in [20], is adaptive, in that it tracks the short-term and the long-term average of the likelihood $p(z_t|z_{t-1}, u_t, m)$. Figure 60 shows the MCL algorithm with the mentioned adaptive algorithm.

Algorithm 6.1.2: $\text{MCL}(\chi_{t-1}, u_t, z_t, m)$

```

static =  $w_{fast}, w_{slow}$ 
 $\bar{\chi}_t = \chi_t = \emptyset$ 
for  $m \leftarrow 1$  to  $M$ 
  do
     $x_t^{[m]} = \text{SAMPLE\_MOTION\_MODEL}(u_t, x_{t-1}^{[m]})$  (1)
     $w_t^{[m]} = \text{MEASUREMENT\_MODEL}(z_t, x_t^{[m]}, m)$  (2)
     $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
     $w_{average} = w_{average} \frac{1}{m} w_t^{[m]}$ 
 $w_{slow} = w_{slow} + \alpha_{slow} (w_{average} - w_{slow})$ 
 $w_{fast} = w_{fast} + \alpha_{fast} (w_{average} - w_{fast})$ 
  for  $m \leftarrow 1$  to  $M$ 
    do
      with probability  $\max(0.0, 1.0 - w_{fast} \setminus w_{slow})$  do
        add random pose to  $\chi_t$ 
      else
        draw  $i$  with probability  $\propto w_t^{[i]}$ 
        add  $x_t^{[i]}$  to  $\chi_t$ 
      endwith
return  $(\chi_t)$ 

```

Figure 60: Modified Monte Carlo Localization Algorithm [20]

Figure 61 shows a variation of particle filter algorithm that adapts the size of the sample set on-the-fly, but what is important for us in this algorithm is the normalization of the weights that happens after the sampling.

There are a few possibilities of how to choose the estimated posterior of the robot based on the converged particles and their weighing factors. For example one could choose the particle with the highest weight as the one closest to the true posterior of the robot, another option is to take the average of particles with highest weighing factors. Although these may yield to a solution, the problem with methods of this kind is that it will completely ignore the distribution which, in fact, disagrees with the main idea of the particle filter. A better approach is to multiply each particle state by its weight and sum them up to obtain the posterior state.

```

Inputs:  $S_{t-1} = \{\langle x_{t-1}^{(i)}, w_{t-1}^{(i)} \rangle \mid i = 1, \dots, n\}$  representing belief  $Bel(x_{t-1})$ ,
control measurement  $u_{t-1}$ , observation  $z_t$ , bounds  $\varepsilon$  and  $\delta$ , bin size  $\Delta$ 

 $S_t := \emptyset, n = 0, k = 0, \alpha = 0$  /* Initialize */
do /* Generate samples ... */
  Sample an index  $j(n)$  from the discrete distribution given by the weights in  $S_{t-1}$ 
  Sample  $x_t^{(n)}$  from  $p(x_t \mid x_{t-1}, u_{t-1})$  using  $x_{t-1}^{(j(n))}$  and  $u_{t-1}$ 
   $w_t^{(n)} := p(z_t \mid x_t^{(n)})$ ; /* Compute importance weight */
   $\alpha := \alpha + w_t^{(n)}$  /* Update normalization factor */
   $S_t := S_t \cup \{\langle x_t^{(n)}, w_t^{(n)} \rangle\}$  /* Insert sample into sample set */
  if ( $x_t^{(n)}$  falls into empty bin  $b$ ) then /* Update number of bins with support */
     $k := k + 1$ 
     $b := \text{non-empty}$ 
     $n := n + 1$  /* Update number of generated samples */
  while ( $n < \frac{1}{2\varepsilon} \chi_{k-1, 1-\delta}^2$ ) /* ... until K-L bound is reached */
  for  $i := 1, \dots, n$  do /* Normalize importance weights */
     $w_t^{(i)} := w_t^{(i)} / \alpha$ 
return  $S_t$ 

```

Figure 61: KLD-Sampling algorithm [61]

6.2. Kinect

6.2.1. Calibration.

Kinect sensor is equipped with two built-in cameras; an RGB camera and an Infrared (IR) depth sensor which comes with an IR. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible [62].

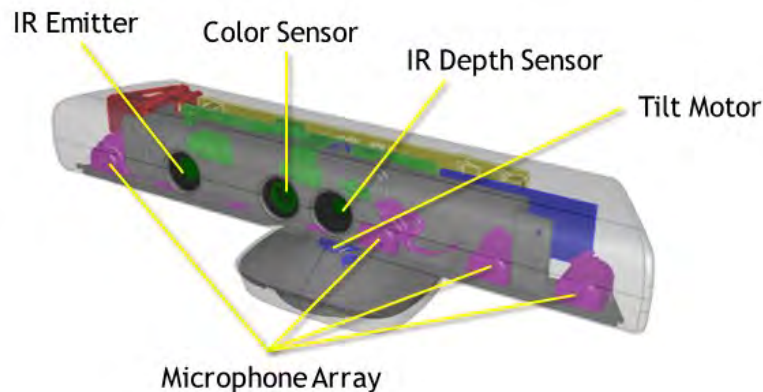


Figure 62: Kinect Sensor Components

Kinect's IR depth sensor uses the same geometrical setup of any traditional pinhole camera model. The projection of a point in three dimensional space onto a pinhole camera's image plane loses the point's depth information. Thus, inverting the projection of a point results in just the beam from the camera's focal point through that point rather than the point itself. However, the image from a depth camera contains points depths instead of color or light information. To compute the actual point in space corresponding to a point on the camera's projected image, the projection process can be easily inverted using the camera's intrinsic parameters.

The vector \vec{r} in Figure 63 defines the ray from the camera's focal point to the nearest object that maps to image coordinates (x_i, y_i) . Knowledge of z from the depth image pixel value enables finding the x and y that originally produced x_i and y_i , respectively, up to the precision allowed by the image's discretization [63].

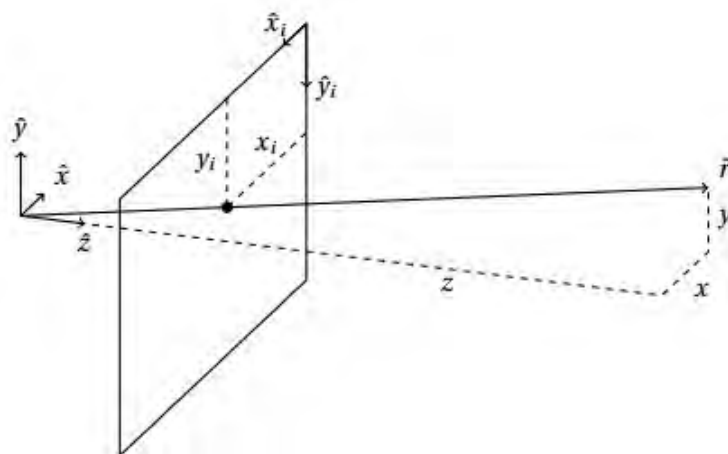


Figure 63: Depth camera imaging geometry based upon the pinhole camera model.

A standard camera calibration is performed to estimate the calibration parameters including the camera's intrinsic using 'kinect-stereo-calib' tool which is a part of MRPT¹ package (Figure 64). A total of 16 images were used by the software which resulted in overall calibration accuracy of 0.650283 pixels as Root Mean Square (RMS) error. The computed parameters can be seen in Table 6.

¹Mobile Robot Programming Toolkit : <http://www.mrpt.org>

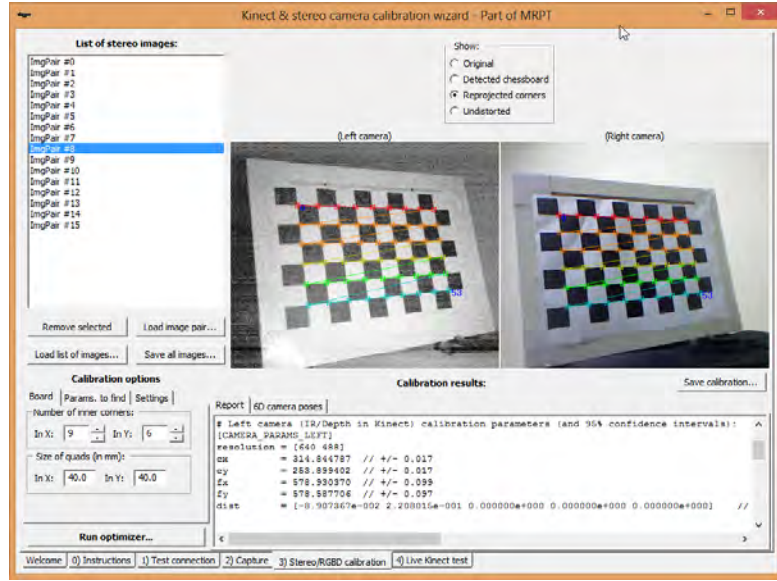


Figure 64: Kinect calibration using a checker board

Table 6: Calibration Parameters

Calibration Parameter		IR Image
Resolution[Pixels]	$W \times H$	640×488
Focal Length[Pixels]	f_x	585.790641 ± 0.100
	f_y	584.429454 ± 0.098
Principal Point Offset[Pixels]	c_x	314.618673 ± 0.017
	c_y	253.087399 ± 0.017
Radial Lens Distortion	$K1$	$-8.712269e - 002$
	$K2$	$2.153357e - 001$
	$K3$	$1.994357e - 002$
Decentering Lens Distortion	$T1$	$-1.784296e - 003$
	$T2$	$1.293932e - 003$

The kinect depth output can be mapped to metric 3D space using the following formula [63, 64]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = z\mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -(u - c_x)\frac{z}{f_x} \\ -(v - c_y)\frac{z}{f_y} \\ z \end{bmatrix} \quad (32)$$

Where

$x, y, z =$ World Coordinates

$u, v =$ Image Coordinates

$z =$ Depth Value

$k =$ Intrinsic Matrix

$f_x, f_y =$ Focal Length

$c_x, c_y =$ Principal Point Offset

(33)

Kinect's depth image size is (640,480) pixels with a 57 degree horizontal Field Of View (FOV) [62]. To consider all the 640 measurements will add unnecessary computational load to the system and its a rather excessive to do so, instead only 20 equally distant points have been taken to form a 57 degree range measurements unit Figure 65.

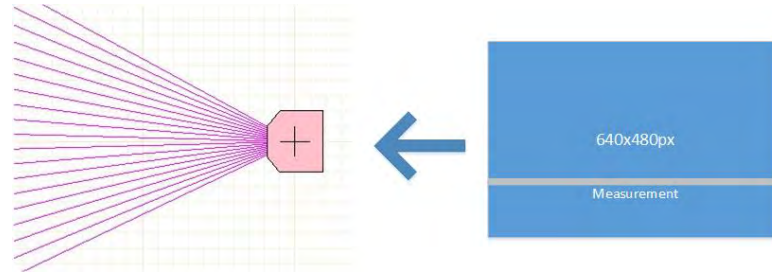


Figure 65: Depth image to range finder.

6.2.2. Measurement.

A frame from the Kinect's depth stream contains a 2D array of pixels. Each pixel holds the Cartesian distance, in millimeters, from the camera plane to the nearest object at that particular (x, y) coordinate, as shown in Figure 66. The (x, y) coordinates of a depth frame do not represent physical units in the room; instead, they represent the location of a pixel in the depth frame [65].

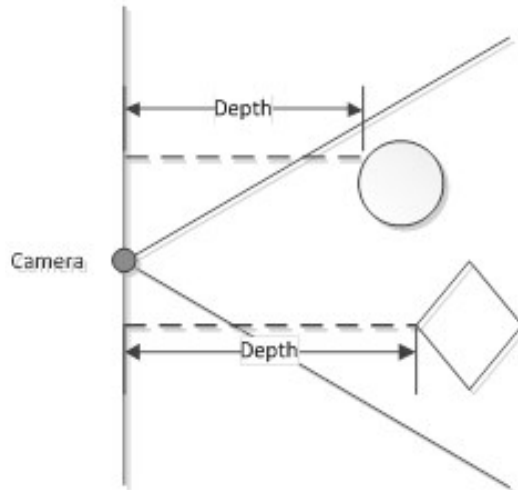


Figure 66: Depth stream values [65]

As illustrated in Figure 67, the default range in the Kinect sensor used in this thesis (Kinect for Xbox 360) can effectively detect distance from objects located from $0.8m$ up to $4.0m$.

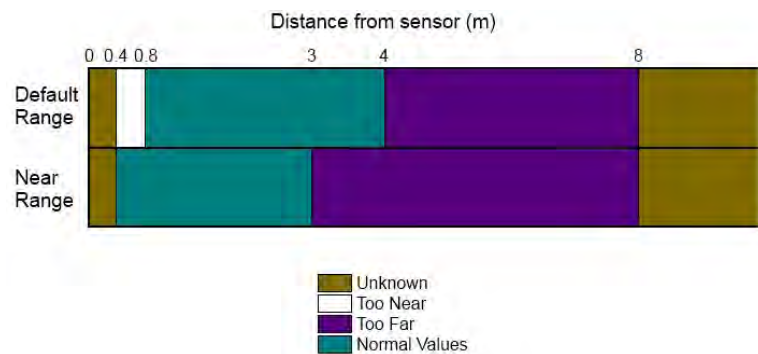
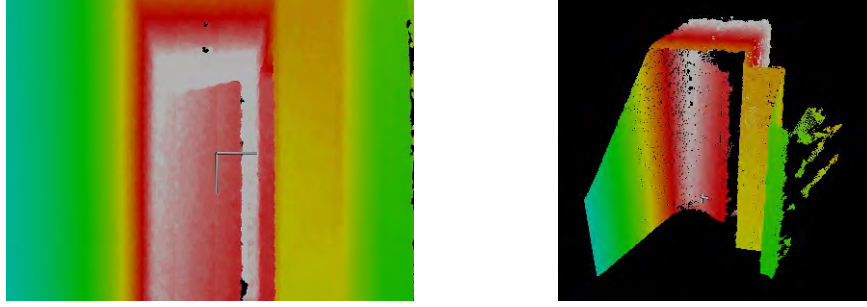


Figure 67: Depth Range [65]

Using (32), 2D depth pixels can be mapped to 3D space(Figure 68). This projection is very useful for our localization problem as it can be used to derive our measurement model which is an essential part of our MCL algorithm. Although, in this thesis we are only interested in 2D localization of the robot, it is also possible to use the depth information for 3D localization.

The measurement model used in our particle system is very similar to the model of range finders, thus the depth data is manipulated to imitate a 2D range finder sensor with measurement errors modeled in a similar manner. There are three types of measurement errors in our model, all of which are essential to making this model work: small



(a) Color Coded 2D Depth Frame

(b) Depth Frame Mapped to 3D Space

Figure 68: A frame from the depth stream and its 3D projection that shows a door at the end of a corridor

measurement noise , errors due to unexpected objects and errors due to failures to detect objects [20].

1. **Correct range with local measurement noise.** Even in a perfect scenario the value returned by sensor is subject to error [20]. For kinect this error arises from sensor itself or properties of the object surface [48] an example of such can be seen in Figure 69, smooth and shiny surfaces that appear overexposed in the infrared image (the lower part of the box).

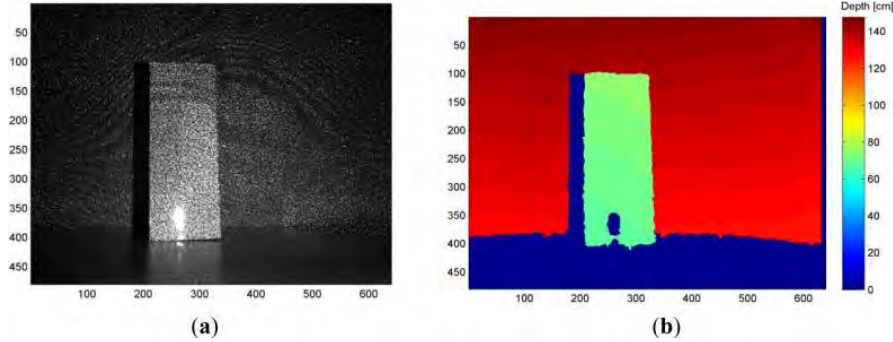


Figure 69: (a) Infrared image of the pattern of speckles projected on a sample scene. (b) The resulting depth image [48].

The range of the values measured by kinect is limited to the interval $[0.8; 4]$ (Figure 67), thus the measurement probability is given by:

$$p_{hit}(z_t^k | x_t, m) = \begin{cases} \eta \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) & \text{if } 0.8 \leq z_t \leq 4 \\ 0 & \text{otherwise} \end{cases} \quad (34)$$

where z_t^k is calculated from x_t and m via ray tracing, and $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2)$ denotes the univariate normal distribution with mean z_t^{k*} and variance σ_{hit}^2 hit:

$$\mathcal{N}\left(z_t^k, z_t^{k*}, \sigma_{hit}^2\right) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{1}{2} \frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}} \quad (35)$$

The normalizer η evaluates to:

$$\eta = \left(\int_{0.8}^4 \mathcal{N}\left(z_t^k, z_t^{k*}, \sigma_{hit}^2\right) dz_t^k \right)^2 \quad (36)$$

2. **Unexpected objects.** Environments of mobile robots are dynamic, whereas maps m are static [20]. As a result, objects not contained in the map can cause Kinect to produce surprisingly short ranges, at least when compared to the map. A typical example of moving objects is when people share the operational space of the robot. One way to deal with such objects is to treat them as a part of the state vector and estimate their location. Another, much simpler approach, is to treat them as sensor noise. Treated as sensor noise, unmodeled objects have the property that they cause ranges to be shorter than z_t^{k*} , not longer. Mathematically, the probability of range measurements in such situations is described by an exponential distribution. The parameter of this distribution, λ_{short} , is an intrinsic parameter of the measurement model. According to the definition of an exponential distribution we obtain the following equation for $p_{short}(z_t^k | x_t, m)$:

$$p_{short}(z_t^k | x_t, m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_t^k} & \text{if } 0.8 \leq z_t^k \leq z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \quad (37)$$

As in the previous case, we need a normalizer η since our exponential is limited to the interval $[0.8; z_t^{k*}]$. Because the cumulative probability in this interval is given as:

$$\int_{0.8}^{z_t^{k*}} \lambda_{short} e^{-\lambda_{short} z_t^k} dz_t^k = -e^{-\lambda_{short} z_t^{k*}} + e^{-\lambda_{short} 0} = 1 - e^{-\lambda_{short} z_t^{k*}} \quad (38)$$

the value of η can be derived as:

$$\eta = \frac{1}{1 - e^{-\lambda_{short} z_t^{k*}}} \quad (39)$$

3. **Failures.** Sometimes, obstacles are missed altogether [20]. With Kinect, failure can happen due to lighting condition which influences the correlation and measurement of disparities [48]. Direct sunlight or any source of IR interference is going to affect the depth-data information, which can lead to outliers or gap in the resulting point cloud. A typical result of sensor failures are max-range measurements: the sensor returns its maximum allowable value z_{max} . Since such events are quite frequent, it is necessary to explicitly model max-range measurements in the measurement model. We will model this case with a point-mass distribution centered at z_{max} :

$$p_{max} \left(z_t^k | x_t, m \right) = I(z = z_{max}) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases} \quad (40)$$

These four different distributions are now mixed by a weighted average, defined by the parameters z_{hit} , z_{short} and z_{rand} with $z_{hit} + z_{short} + z_{rand} = 1$ (Figure 70).

$$p_{max} \left(z_t^k | x_t, m \right) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit} \left(z_t^k | x_t, m \right) \\ p_{short} \left(z_t^k | x_t, m \right) \\ p_{rand} \left(z_t^k | x_t, m \right) \end{pmatrix} \quad (41)$$

Algorithm 6.2.1: BEAM_RANGE_FINDER_MODEL(z_t, x_t, m)

```

q = 1
for k ← 1 to K
  do {
    compute  $z_t^{k*}$  for the measurement  $z_t^k$  using ray casting
     $p = z_{hit} \cdot p_{hit} \left( z_t^k | x_t, m \right) + z_{short} \cdot p_{short} \left( z_t^k | x_t, m \right) + z_{rand} \cdot p_{rand} \left( z_t^k | x_t, m \right)$ 
     $q = q \cdot p$ 
  }
return (q)

```

Figure 70: Algorithm for computing the likelihood of a depth measurement z_t , assuming conditional independence between the individual depth measurements in the image.

6.3. Simulation

The first simulation (Figure 72) shows the MCL algorithm in action with 5000 particles. At start (a) all the particles are randomly distributed in the map except for small gaps near the walls that are physically impossible for the robot to be in due to the size of the robot. After a few steps (b, c) the particles get more concentrated. The last image, (d) shows that most of particles are converged in two possible locations that are closest to the true location of the robot, finally in (e) all the particles are converged. Estimated location of the robot can be seen in (f).

The size of the particle set used in MCL can be varied depending upon the size of the map and the required accuracy of the localization. Figure 72 demonstrates the MCL with a particle set of 1000 particles in the same map. This simulation is an example of how MCL can fail due to relative symmetry that exist in the map (c), in (d) the algorithm tries to recover from this failure by inserting random particles in the map.

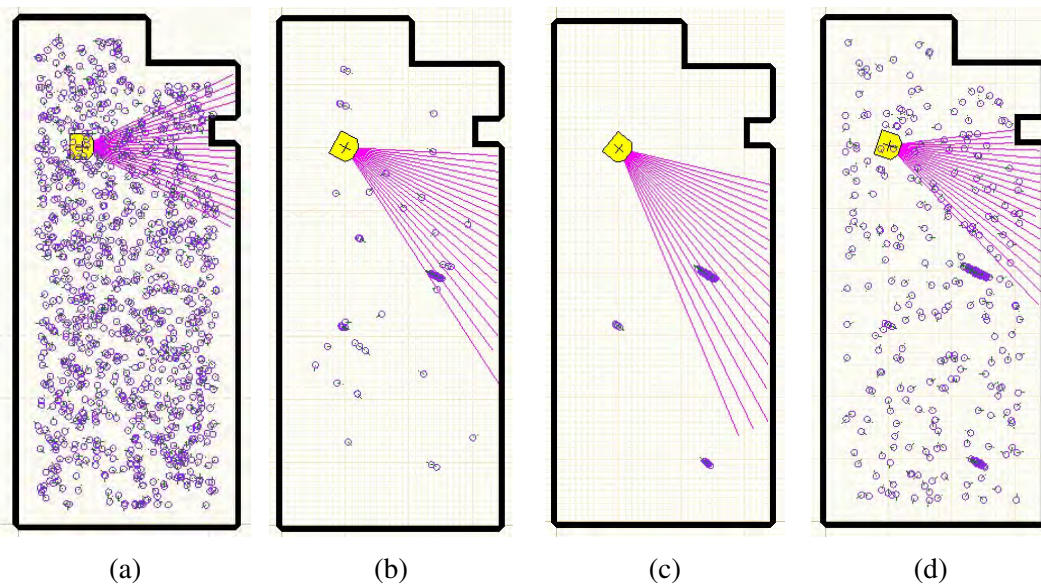


Figure 71: Monte Carlo Localization with 1000 particles

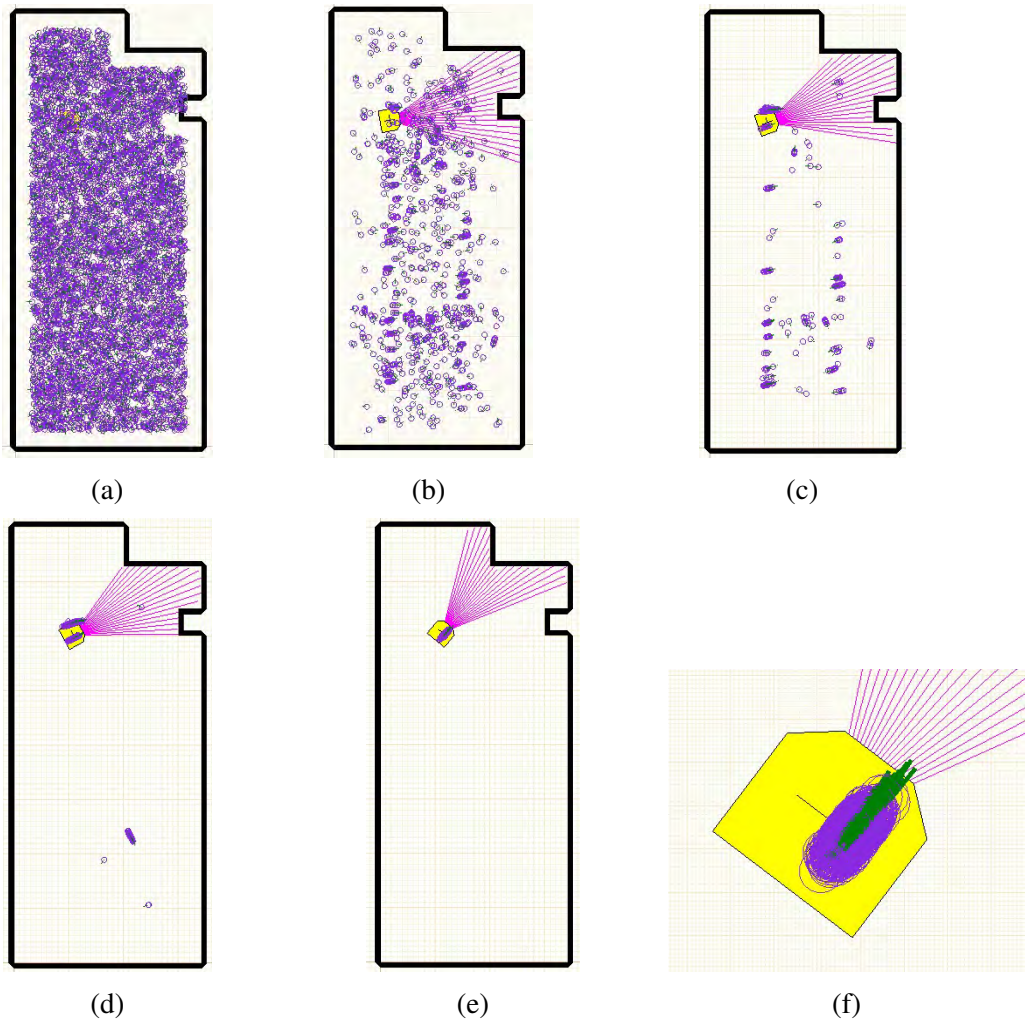


Figure 72: Monte Carlo Localization with 5000 particles

The size of the particle set can be significantly reduced by taking advantage of the last known location of the robot given by the EKF. After all, the MCL will only come into effect when the GPS is lost, thus the whereabouts of the robot can be used in the initial distribution of the particles. This approach will increase the likelihood of a successful localization while reducing the computational load of the algorithm. This approach has been successfully tested in two different maps (Figure 73 (a) to (d) and (e) to (h)) with only 200 particles.

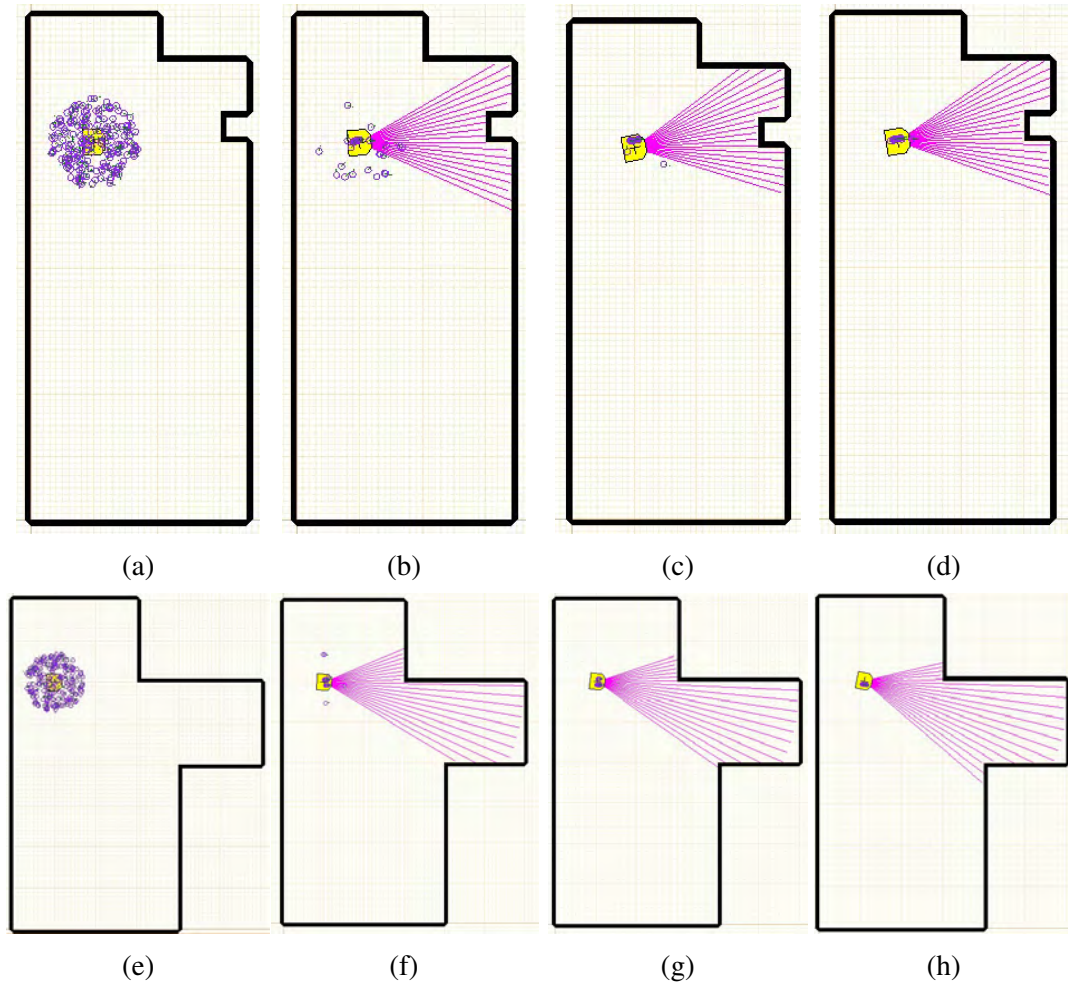


Figure 73: Monte Carlo Localization with 200 particles

6.4. Practical Results

The indoor navigation algorithm has been tested extensively using different maps. To measure the accuracy, location of the waypoints was marked on the ground and then manually measured with respect to the robot position. The experiment depicted in Figure 74, has been conducted in the Engineering Building 1 in American University of Sharjah (AUS) with a set of 800 particles. It shows the robot following a set of way points, leading it approximately $6m$ in a straight line and then back to the starting position.

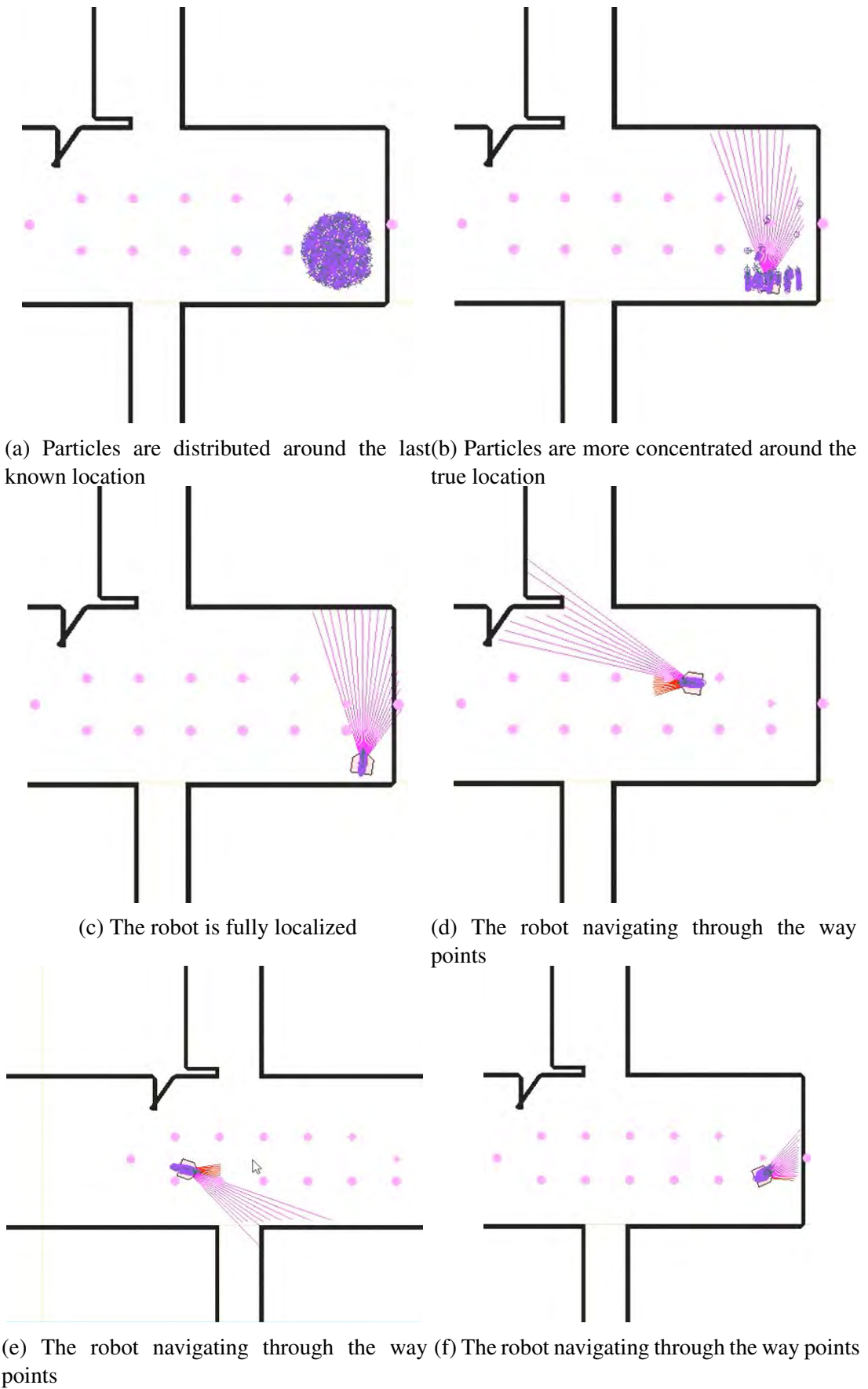


Figure 74: Indoor Navigation with a set of 800 particles

Using the data recorded during this test, it is possible to plot the robot trajectory estimated by the MCL in matlab (Figure 75).

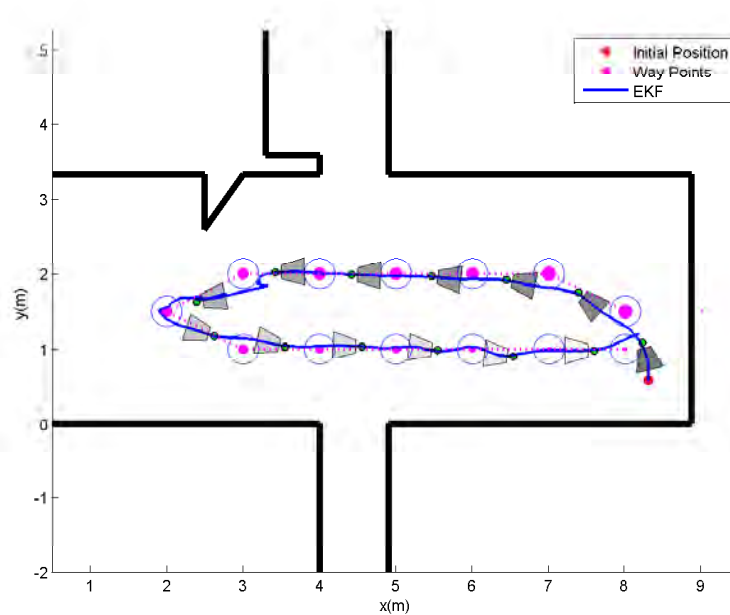


Figure 75: Robot trajectory shown in blue, has been recorded during the practical test in AUS

To test the accuracy of the localization, the MCL output has been compared to the actual position of the robot, measured manually. The accuracy was shown to be highly dependent on the availability of depth measurement. Due to the limited range of the Kinect sensor, it is unable to provide data for objects located outside of its range. In this particular test an average error of 6.72% with a standard deviation of 3.1% , was measured on ten random locations of the map. The larger errors were recorded in parts of the map, where there was no object anywhere within 4m of the Kinects field of view.

Chapter 7: Hybrid System (Indoor/Outdoor)

Two solutions presented in this thesis are proved to be useful in their designated environments. This chapter discusses the integration of the two systems and provides an inclusive solution for both indoor and outdoor environments.

One way to determine if the robot has entered an indoor environment is to monitor the number of satellite used by the GPS. A GPS needs at least 4 different satellite signals to workout position in 3-dimensions. One can add a condition in the programs main loop to switch to the indoor algorithm, if the number of Satellite Vehicle (SV) drops below 4 (or 3). One problem with this assumption is that there may be some instances where the GPS momentarily loses the satellite connection while still located outdoor, while the outdoor algorithm can still handle such situations by relying only on encoder and heading measurements. The indoor algorithm will be rendered useless under the outdoor conditions such as sunlight which interferes with the Kinect's IR emitter. Another problem with this method is the delay that exist between the time robot enters the indoor map and the SV number changes in GPS output. During this delay the robot will still be using the outdoor algorithm which will yield to erroneous localization. A safer approach would be to check the first way-point position against the indoor map. At the starts, to see if the point is located inside or the outside of the map polygon(point-in-polygon (PIP)) and then to run the appropriate localization algorithm accordingly. From this point forward, the same process is repeated, but with the estimated position of the robot instead of the way-point at the end of each iteration of the main loop. The PIP problem can be resolved by using a simple ray-casting algorithm (Figure 76). The idea is to test how many times a ray starting for the point p intersects the sides of the polygon (the indoor map is defined as a polygon). The number of intersections is an even if the point is outside, and it is odd if inside.

Algorithm 7.0.1: RAYCASTING($p, polygon$)

```
Count  $\leftarrow$  0
for each  $side \in polygon$ 
  do if  $ray\_intersects\_segment(P, side)$ 
  then  $count \leftarrow count + 1$ 
if  $is\_odd(count)$ 
  then return ( $inside$ )
  else return ( $outside$ )
```

Figure 76: Ray Casting Algorithm

A general flowchart of this process can be viewed in Figure 77

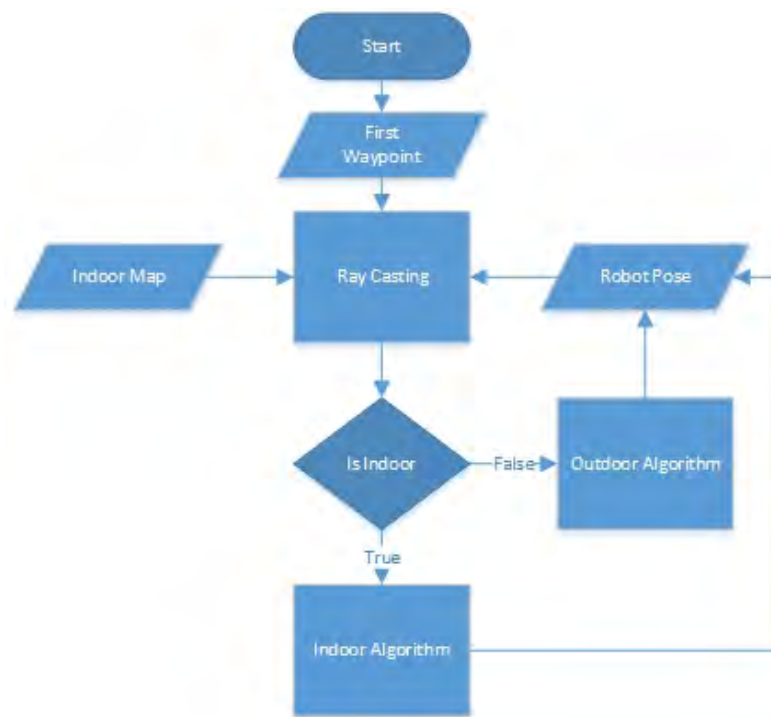


Figure 77: Ray Casting Flowchart

7.1. Practical Result

The area shown in Figure 78 has been used to test the hybrid system. By placing the starting way-point on the middle of the bridge the robot starts following the way-points towards the building. One problem here is that the accuracy of the GPS deteriorates sharply as the robot approaches the building. This localization error is corrected by the MCL once the robot enters the indoor map.



Figure 78: The hybrid system has been tested by placing it on the bridge between the two engineering buildings in AUS

The collected data from the experiment is used to plot overall robot trajectory (Figure 79).

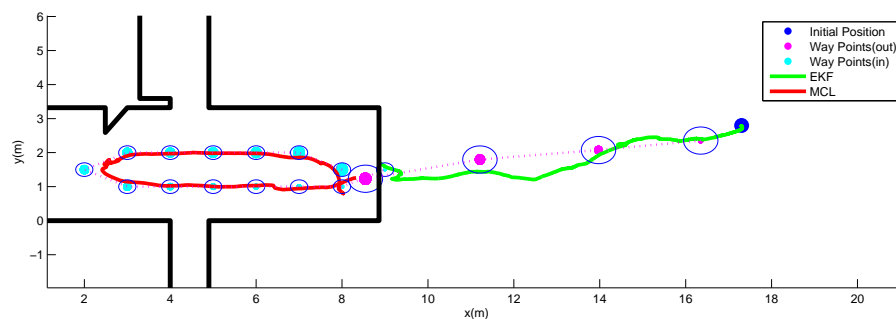


Figure 79: Robot trajectory during the hybrid test, plotted using the recorded data.

Chapter 8: Conclusion and Future Work

8.1. Summary

This chapter concludes this thesis by outlining its contents, and suggesting how the work presented here can be extended or improved.

The trajectory following problem requires a unique solution for both indoor and outdoor navigation, thus by considering the thesis objective, two different approaches analyzed in chapter 4 that lead to the conclusion that the I-O SFL method is better suited for this application than the DFL.

A GUI, which is developed for this thesis, consists of a simulation environment that provides real time illustration of the robot movements based on the running algorithm and sensor readings. Using the GUI all the parameters such as output of the algorithm and sensor data can be monitored. It also provides the option to interactively add or modify the way-point.

For outdoor navigation, data from the GPS, odometry and IMU (heading) are fused using EKF, to improve the position accuracy of the robot. This data is then used in the trajectory following algorithm to calculate the command parameters needed to control the robot velocity.

Another probabilistic method based on MCL is used for indoor navigation that relies on the depth measurements of a Kinect sensor. The depth data of the Kinect is accurately projected from the image plane to the world coordinates using the intrinsic parameters of its camera retrieved after calibration. To use these projected points a function is developed that takes the points as input and provides an output similar to the output from a two-dimensional range finder sensor.

The different approaches explored in this thesis, when combined together, provide a comprehensive solution to global navigation problems. The current software is separated into two different executables; one for indoor navigation and one for outdoor navigation. While, each program can be run on its own, there is an option available that automates the switching operation. This means that, the last known location of the robot is passed as command-line arguments between the programs once the robot enters an

indoor environment or vice versa. It is worth mentioning that the controllers developed in this thesis are based on the kinematics of wheeled mobile robots that can only move in 2 dimensions. The same techniques can be modified and then applied for navigation in 3D space using platforms such as quad-rotors.

8.2. Future Work

There are several ways in which the navigation system presented here can be improved. A list of such improvements is gathered here.

- Using a single on board processor to control both, the robot motion and the localization algorithm, which will effectively eliminate the communication latency that exists in the current system.
- The covariance matrix plays a fundamental role in the GPS data adjustment, using approaches such as [66] can provide a faster and more smooth filter.
- Using Compute Unified Device Architecture (CUDA) platform, the computationally expensive algorithms such as particle filters can be processed in parallel in the Graphics Processing Unit (GPU), thus more particles can be used in less time.
- KLD-Sampling [61] provides a statistical approach for particle filters that adapts the size of sample sets during estimation that can improve the efficiency of particle filter.
- The indoor and outdoor executables can be merged together, by rearranging the GUI to provide an interface that is appropriate for both indoor and outdoor navigation.

References

- [1] J. Borenstein, H. R. Everett, L. Feng, and D. K. Wehe, "Mobile robot positioning: Sensors and techniques," *J. Field Robotics*, vol. 14, no. 4, pp. 231–249, 1997.
- [2] Y. S. Khraisat, M. Al-Khateeb, Y. Abu-Alreesh, A. Ayyash, and O. Lahlouh, "Gps navigation and tracking device." *ijim*, vol. 5, no. 4, pp. 39–41, 2011.
- [3] C. Magnusson, K. Rasmussen, and D. Szymczak, "Navigation by pointing to gps locations," *Personal and Ubiquitous Computing*, vol. 16, no. 8, pp. 959–971, 12 2012.
- [4] R. Zhu and H. A. Karimi, "Automatic selection of landmarks for navigation guidance," *Transactions in GIS*, 2014.
- [5] C. Tessier, M. Berducat, R. Chapuis, and F. Chausse, "A new landmark and sensor selection method for vehicle localization and guidance," in *Intelligent Vehicles Symposium, 2007 IEEE*, June 2007, pp. 123–129.
- [6] A. Howard and L. Kitchen, "Navigation using natural landmarks," *Robotics and Autonomous Systems*, vol. 26, no. 2, pp. 99–115, 1999.
- [7] C. McGillem and T. Rappaport, "A beacon navigation method for autonomous vehicles," *Vehicular Technology, IEEE Transactions on*, vol. 38, no. 3, pp. 132–139, Aug 1989.
- [8] E. Brassart, C. Pegard, and M. Mouaddib, "Localization using infrared beacons," *Robotica*, vol. 18, pp. 153–161, 2000.
- [9] W. Eom, J. Park, and J. Lee, "Hazardous area navigation with temporary beacons," *International Journal of Control, Automation and Systems*, vol. 8, no. 5, pp. 1082–1090, 2010.
- [10] S. Taneja, B. Akinci, J. H. Garrett, L. Soibelman, and H. A. Karimi, "Effects of positioning data quality and navigation models on map-matching of indoor positioning data," *Journal of Computing in Civil Engineering*, p. 4014113, 2014.
- [11] M. Ren and H. A. Karimi, "A hidden markov model-based map-matching algorithm for wheelchair navigation," *The Journal of Navigation*, vol. 62, no. 3, pp. 383–395, 2009.
- [12] L. Yaojun, P. Quan, Z. Chunhui, and Y. Feng, "Scene matching based ekf-slam visual navigation," in *Control Conference (CCC), 2012 31st Chinese*, July 2012, pp. 5094–5099.
- [13] J. Inthiam and C. Deelertpaiboon, "Self-localization and navigation of holonomic mobile robot using omni-directional wheel odometry," in *TENCON 2014 - 2014 IEEE Region 10 Conference*, Oct 2014, pp. 1–5.

- [14] H. Azartash, N. Banai, and T. Nguyen, “An integrated stereo visual odometry for robotic navigation,” *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 414–421, 2014; 2013.
- [15] B. Barshan and H. Durrant-Whyte, “Inertial navigation systems for mobile robots,” *Robotics and Automation, IEEE Transactions on*, vol. 11, no. 3, pp. 328–342, Jun 1995.
- [16] M. Al-Sharman, M. Abdel-Hafez, and M. Al-Omari, “Attitude and flapping angles estimation for a small-scale flybarless helicopter using a kalman filter,” *Sensors Journal, IEEE*, vol. 15, no. 4, pp. 2114–2122, April 2015.
- [17] A.-H. M. F. J. M. A. J. M. A. Saadeddin, Kamal, “Performance enhancement of low-cost, high-accuracy, state estimation for vehicle collision prevention system using anfis,” *Mechanical Systems and Signal Processing*, vol. 41, no. 1-2, pp. 239–253, 2013.
- [18] M. Jaradat and M. Abdel-Hafez, “Enhanced, delay dependent, intelligent fusion for ins/GPS navigation system,” *Sensors Journal, IEEE*, vol. 14, no. 5, pp. 1545–1554, May 2014.
- [19] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*, 1st ed. Springer Publishing Company, Incorporated, 2007.
- [20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [21] M. Jaradat, M. BaniSalim, and F. Awad, “Autonomous navigation robot for landmine detection applications,” in *Mechatronics and its Applications (ISMA), 2012 8th International Symposium on*, April 2012, pp. 1–5.
- [22] G. Tuna and K. Gulez, “Aided navigation techniques for indoor and outdoor unmanned vehicles,” in *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, 2012, pp. 1–4.
- [23] G. Zhan and W. Shi, “Lobot: Low-cost, self-contained localization of small-sized ground robotic vehicles,” vol. 24, no. 4, pp. 744–753, 2013.
- [24] N. Priyantha, D. Lymberopoulos, and J. Liu, “Eers: Energy efficient responsive sleeping on mobile phones,” *Proceedings of PhoneSense*, pp. 1–5, 2010.
- [25] J. Paek, J. Kim, and R. Govindan, “Energy-efficient rate-adaptive gps-based positioning for smartphones,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 299–314.
- [26] Z. Zhuang, K.-H. Kim, and J. P. Singh, “Improving energy efficiency of location sensing on smartphones,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 315–330.

- [27] Y. Sakamoto, T. Ebinuma, K. Fujii, and S. Sugano, “Gps-compatible indoor-positioning methods for indoor-outdoor seamless robot navigation,” in *Advanced Robotics and its Social Impacts (ARSO), 2012 IEEE Workshop on*, May 2012, pp. 95–100.
- [28] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. IEEE, 2001, pp. 145–152.
- [29] S. Grzonka, G. Grisetti, and W. Burgard, “Towards a navigation system for autonomous indoor flying,” in *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE, 2009, pp. 2878–2883.
- [30] G. Grisetti, C. Stachniss, and W. Burgard, “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling,” in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 2432–2437.
- [31] J. Civera, A. J. Davison, and J. Montiel, “Inverse depth parametrization for monocular slam,” *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 932–945, 2008.
- [32] F. Dellaert and M. Kaess, “Square root sam: Simultaneous localization and mapping via square root information smoothing,” *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.
- [33] A. Oliver, S. Kang, B. C. Wünsche, and B. MacDonald, “Using the kinect as a navigation sensor for mobile robotics,” in *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, ser. IVCNZ ’12. New York, NY, USA: ACM, 2012, pp. 509–514.
- [34] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, “Real-time 3d visual slam with a hand-held rgb-d camera,” in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, vol. 180, 2011.
- [35] G. Oriolo, A. De Luca, and M. Vendittelli, “Wmr control via dynamic feedback linearization: design, implementation, and experimental validation,” *Control Systems Technology, IEEE Transactions on*, vol. 10, no. 6, pp. 835–852, Nov 2002.
- [36] J. M. J.C. Alexander, “On the kinematics of wheeled mobile robots,” *Int. J. Robotics Research*, vol. 8, pp. 15 – 27, 1989.
- [37] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, “Stereo vision based indoor/outdoor navigation for flying robots,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 2013, pp. 3955–3962.
- [38] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, “Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft mav,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 4974–4981.

- [39] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Proc. of AeroSense: The 11th Int. Symp. on Aerospace/Defense Sensing, Simulations and Controls*, 1997.
- [40] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME—Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [41] K. Saadeddin, M. Abdel-Hafez, M. Jaradat, and M. Jarrah, "Optimization of intelligent-based approach for low-cost ins/gps navigation system," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, May 2013, pp. 668–677.
- [42] S. Thrun, "Particle filters in robotics," in *in Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.
- [43] J. Míguez, D. Crisan, and P. M. Djurić, "On the convergence of two sequential monte carlo methods for maximum a posteriori sequence estimation and stochastic global optimization," *Statistics and Computing*, vol. 23, no. 1, pp. 91–107, Jan. 2013.
- [44] S. Engelson and D. McDermott, "Error correction in mobile robot map learning," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, 1992, pp. 2555–2560.
- [45] I. Noda, S. Suzuki, H. Matsubara, M. Asada, and H. Kitano, "Robocup-97: The first robot world cup soccer games and conferences." *AI Magazine*, vol. 19, no. 3, pp. 49–59, 1998.
- [46] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1, pp. 99–141, 2001.
- [47] S. Lenser and M. M. Veloso, "Sensor resetting localization for poorly modelled mobile robots." in *ICRA. IEEE*, 2000, pp. 1225–1232.
- [48] K. Khoshelham, "Accuracy analysis of kinect depth data," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-5/W12, pp. 133–138, 2011.
- [49] *Control of Unicycle Type Robots Tracking, Path Following and Point Stabilization*, vol. IV. Jornadas de Engenharia Electrónica e Telecomunicações e de Computadores, November 2008.
- [50] A. D. Luca and M. D. D. Benedetto, "Control of nonholonomic systems via dynamic compensation," *Kybernetika*, vol. 29, no. 6, pp. 593–608, 1993.
- [51] B. d'Andrea novel, G. Bastin, and G. Campion, "Dynamic feedback linearization of nonholonomic wheeled mobile robots," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, 1992, pp. 2527–2532.

- [52] B. d'Andréa Novel, G. Campion, and G. Bastin, "Control of nonholonomic wheeled mobile robots by state feedback linearization," *Int. J. Rob. Res.*, vol. 14, no. 6, pp. 543–559, Dec. 1995.
- [53] T. Hughes, "Electronic aids to navigation," *Journal of Navigation*, vol. 45, pp. 446–446, 9 1992.
- [54] L. Sahawneh, M. Al-Jarrah, K. Assaleh, and M. Abdel-Hafez, "Real-time implementation of gps aided low-cost strapdown inertial navigation system," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1-4, pp. 527–544, 2011.
- [55] S. C. W. Kwok. Geodetic datum transformation. Geodetic Survey Section, Lands Department, Hong Kong. [Accessed June 14, 2015]. [Online]. Available: <http://www.geodetic.gov.hk/smo/gsi/data/pdf/transformation.pdf>
- [56] National Imagery and Mapping Agency, "Department of defense world geodetic system 1984: its definition and relationships with local geodetic systems," National Imagery and Mapping Agency, St. Louis, MO, USA, Tech. Rep. TR8350.2, Jan. 2000.
- [57] "Users handbook on datum transformations involving wgs 84," International Hydrographic Organization, Tech. Rep., July 2003.
- [58] B. Parkinson and J. Spilker, *Global Positioning System: Theory and Applications*. American Institute of Aeronautics and Astronautics, 1996, no. v. 1.
- [59] L. H. C. J. Hofmann-Wellenhof, B., *Global positioning system: theory and practice*, B. Hofmann-Wellenhof, Ed. Springer-Verlag Wien, 2001.
- [60] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, 1999, pp. 1322–1328 vol.2.
- [61] D. Fox, "Kld-sampling: Adaptive particle filters," University of Washington, Tech. Rep., 2001.
- [62] Kinect for windows sensor components and specifications. Microsoft. [Accessed June 14, 2015]. [Online]. Available: <https://msdn.microsoft.com/en-us/library/jj131033.aspx>
- [63] M. T. Draelos, "The kinect up close: Modifications for short-range depth imaging," Master's thesis, North Carolina State University, 2012.
- [64] Camera calibration and 3d reconstruction. Intel Corporation, Willow Garage, Itseez. [Accessed June 14, 2015]. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
- [65] Coordinate spaces. Microsoft. [Accessed June 14, 2015]. [Online]. Available: <https://msdn.microsoft.com/en-us/library/hh973078.aspx>

- [66] K. C. Rodrigo Leandro, Marcelo Santos, “An empirical approach for the estimation of gps covariance matrix of observations,” in *61st Annual Meeting of The Institute of Navigation*, 2005, pp. 1098 – 1104.

Vita

Milad Roigari was born in Tehran, Iran on January 12 , 1986. After finishing high school in 2005, he continued his education in Sharif University of Technology and graduated with B.Sc in Mechatronics engineering in 2010. Milad has moved to United Arab Emirates in 2011 to pursue his M.Sc in Mechatronics engineering, where he got a scholarship and worked as a Graduate Teaching assistant with both graduate and undergraduate students on various classes.