IMPLEMENTATION OF AUTONOMOUS NAVIGATION AND OBSTACLE AVOIDANCE ON AN UNMANNED GROUND VEHICLE

by

PRANAV NARESH DESAI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

To my parents Naresh and Anila.

ACKNOWLEDGEMENTS

I would like to thank several people who made this dream possible. I had the opportunity to work with three different professors during the course of this research. I am grateful towards my supervisory professors Dr. Atilla Dogan and Dr. Brian Huff. My frequent talks during this research with Dr. Huff and Dr. Dogan were inspiring, motivating, and fruitful. Thanks to Dr. Huff who constantly motivated me and spend late nights with me at the laboratory. I have learned a lot about how to approach and solve a problems from Dr. Dogan.

I am truly grateful to Dr. Engels without whom I would have never pursued my masters with thesis. I would also like to acknowledge his constant motivation and support. I would also like thank Dr. Gibbs and Dr. Dillon for their support.

I would to thanks my family without which this was not possible. I would also like to acknowledge the support and constant motivation of all my friends. Acknowledgments to room-mates Manoj, Mihir and Kunal for tolerating me and my loud music for two years. I would also like thanks my lab mate George for his support.

November 20, 2009

ABSTRACT

IMPLEMENTATION OF AUTONOMOUS NAVIGATION AND OBSTACLE AVOIDANCE ON AN UNMANNED GROUND VEHICLE

PRANAV NARESH DESAI, M.S.

The University of Texas at Arlington, 2009

Supervising Professors: Dr. Atilla Dogan and Dr. Brian Huff

This thesis presents the implementation of a novel distributed embedded systems approach to real-time obstacle avoidance and guidance for an Unmanned Ground Vehicle (UGV). The mobility, real-time, and limited size requirements of UGVs, result in computationally limited and resource constrained hardware platform. The use of distributed computational resources, such as multiple embedded micro-controllers, enables the distribution of the computing resources for obstacle avoidance and guidance system functionalities. The resulting system's complexity is significantly greater than that of a single high performance processor performing all of the above functions.

The hardware platform is integrated with sensors and micro-controllers to function as the real-time obstacle avoidance and guidance system for a UGV. The sensors include: a GPS receiver, a digital compass, rotary encoders, and a scanning laser range finder. All sensors have been calibrated and characterized for accuracy and reliability. The obstacle avoidance and guidance functionality executes on MPC555 micro-controller. The data strings from the sensors are parsed on IsoPod^{TM}, PlugaPod^{TM}

micro-controllers. The required sensor data are passed over to the MPC555 over CAN network as part of a distributed computing architecture.

A simulation model consisting of the guidance and navigation algorithm along with the tank model was developed. The simulation model performs obstacle avoidance and waypoint navigation successfully. A real-time model to perform obstacle avoidance and waypoint navigation was developed. The real-time model takes inputs as sensor data, constructs a dynamic map of the environment and outputs control signals to navigate the vehicle through obstacles and towards waypoints. The real-time system successfully performs waypoint navigation. The real-time systems constructs an inaccurate local map in real-time environment. An accurate local map is successfully constructed in simulation from the real world data. Due to the erroneous map constructed in real-time, the real-time system does not successfully navigate through the obstacles.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Real-time obstacle avoidance and navigation are key fields of research in the area of autonomous vehicles. The primary requirements of autonomy are to detect or sense changes and react to them without human intervention in a safe and efficient manner. Changes are detected by sensors like temperature, pressure, position, orientation which indicate the current working state of the system. The success of an autonomous vehicle to a large extent depends on how fast and how accurately it can identify a change in real-time and compensate for the change, while moving towards the target destination.

Most algorithms developed for obstacle avoidance and navigation are not validated through implementation on an actual controller controlling a physical vehicle or platform. Those that do get implemented [1] tend to be either on a custom robot or on a much larger and bulky processing unit. The goal of this research is successful implementation of an obstacle avoidance and path planning algorithm [2] on a small, computationally constrained unmanned ground vehicle.

1.2 Problem Statement

There has been an considerable increase in interest [3] in the development of small, cheap and disposable UAVs[1]. Inexpensive unmanned systems that perform mission specific, or limited, functions can be deployed in large quantities over larger

[1]Unmanned Aerial Vehicles

areas providing greater benefits for the same cost as the current high cost multi-function platforms. Size and weight distribution are very important design constraints for small and reliable UGV[2] and UAV platform. An embedded and distributed computing approach can be adopted to distribute weight and processing power. Also, an important factor to consider is real-time implementation. This requirement can differ from case to case, depending on the accuracy, speed, and operation to be performed as well as the time window in which it is performed.

1.3 Thesis Objective

The primary objective of this research is to serve as an intermediate step for implementation of these algorithms [2] on an UAV. The objective of this research is ***to develop and implement a UGV control system utilizing embedded microcontrollers. This control system must enable the UGV to autonomously navigate to the specified waypoints while avoiding obstacles discovered in its path. The control algorithms implemented within this physical control system have been adapted from previous work done by Zengin [2]. This research expands the work of Zengin by developing and implementing environment mapping system capable of detecting and localizing potential obstacles from real-time sensor data. The real-time obstacle mapping system developed in this work automatically generates the Probabilistic Threat Exposure Map (PTEM) which previously had to be provided as an input to Zengin's control algorithms.*** There are several tasks that must be performed to achieve the this objective:

- The evaluation and integration of sensors used on the UGV
- The construction of the Probabilistic Threat Exposure Map

[2]Unmanned Ground Vehicles

- The integration of guidance module
- The simulation and real-world testing of the UGV system

1.3.1 Evaluation and Integration of Sensors

The accuracy, reliability, and repeatability of the sensors are very important since the dynamic map is constructed using live data from sensors. Evaluation and characterization of each individual sensor is performed. Each individual sensor is integrated into a common framework to construct the map.

1.3.2 Construction of a Probabilistic Threat Exposure Map (PTEM)

PTEM [3] [2],[4],[5],[6] is a common framework to represent all the obstacles in the locality. The framework provides the mathematical foundation for the guidance strategy to make intelligent decisions during the execution of the mission. The first task is to form a PTEM using sensor data. It is constructed in real time using an on-line approach. Once constructed, the PTEM provides a common map of the areas the vehicle needs to avoid and the areas of safe operation for the vehicle. The PTEM construction module facilitates the construction of map from continuous real-time updates from the physical environment.

1.3.3 Integration of a Guidance Module

The guidance module is based on the guidance algorithm developed by Zengin in [2]. The guidance module is designed in Matlab-Simulink environment and is physically implemented on an embedded micro-controller. The guidance algorithm interacts with the physical platform and deals with uncertainties while making complex decisions in real time. To better implement and test the guidance algorithms, a

[3]Probabilistic Threat Exposure Map

simulation model of the platform is developed. The simulation model is integrated with the guidance module.

1.3.3.1 Assumptions

The following assumptions are considered in the implementation and evaluation of the guidance module.

- The map of the area of operation represents a 2-D model of the environment.
- Motion is modeled in 3-DOF. Specifically, translation motion relative to X and Y axis and rotational about Z axis.
- Obstacles are static and cylindrical in nature

1.3.4 Simulation and Real-World Testing

The concluding task in the project is to develop the simulation model and perform rigorous real-time testing of the system. The simulation of the system helps in identification of various errors in the implementation. Real-world testing is the last step in the development of this system. This helps in identifying new sets of constraints and problems. It also helps in evaluating the success and robustness of the algorithms designed and implemented.

1.4 Applications

As stated earlier, the prominent assertion that this work's application to a UGV is simply an intermediate step towards its use on an UAV. Autonomous robotic systems capable of safe and efficient navigation in complex domains have a wide variety of potential applications. These applications include: surveying robots, search and rescue drones, and planetary rovers, autonomous crop harvesters, mining trucks, transport vehicles. In many cases, autonomous robots can also be used in place of

a manned system to execute tasks at a lower cost. This provides additional benefits like extended operating hours beyond the endurance of human beings and the ability to use sensing capabilities not possessed by humans. Another invaluable application is the ability for robotic systems to operate in domains that are not safe for humans, such as battlefields, toxic environments, and outer space.

1.5 Thesis Organization And Contributions

Chapter 2 provides an extensive literature review of previous work related to this research. This chapter focuses on different algorithm designs and physical implementations of systems capable of performing obstacle avoidance and navigation tasks.

Chapter 3 gives an overall view of both the hardware system architecture and software system architecture. This chapter provides a detailed description of how different hardware components interact with each other. The software architecture describes various programming languages used and the software model developed during the implementation of the project.

Chapter 4 introduces the hardware used in the project. It provides detailed description of all the hardware and software components used during the research project.

Chapter 5 gives a detailed description and the mathematical formulation used to construct the PTEM. It describes the algorithms developed to construct the PTEM. The construction of a PTEM example from sensor data is presented in this chapter. A clustering algorithm used to cluster obstacle points is also described in this chapter.

Chapter 6 will introduce the simulation and real-time navigation and control model developed. A module by module description of both the navigation and control models are provided in this chapter.

Chapter 7 discusses experimental setups and presents the results derived from these experiments. Results obtained from various simulations of the navigation and control model are presented in this chapter. A detailed analysis and conclusion of all the results obtained from the various experiments performed on the real-time navigation and control model is also given in this chapter. A detailed comparison of results obtained from simulation with those from real-time testing is provided in this chapter. This chapter also presents the current status of the research project. It describes in brief the successes and the shortcomings of this research project. It will also provide detailed explanations for why the anticipated results have not been obtained.

Chapter 8 concludes the document by stating the conclusions. It will also state the directions for suggested future work for this research. It also provides suggestions to overcome or debug the shortcomings of this research project.

The following list summarizes the contributions of this research.

- First embedded system implementation of the path planning and guidance algorithm originally developed by Zengin in [2].
- First evaluation of this algorithm within an autonomous vehicle.
- Developed and implemented a novel on-line algorithm to dynamically generate a map of the environment.
- Developed and implemented a sensor enabled common framework for the development and test of path planning and guidance algorithms.
- Developed and implemented a novel on-line vehicle localization algorithm using both global and local navigation frames.
- Developed run time algorithms capable of generating PTEM from a suite of sensors operating in an unstructured outdoor environment.

The path planning and guidance algorithm developed by Zengin [2] is evaluated on an autonomous vehicle for the first time. Prior to this implementation, this algorithm has only been successfully tested in a simulation environment. The successful implementation on an UGV will validate this algorithm for real-time implementation. For real-time implementation, the dynamic real-time map of the environment is constructed. This dynamic map is constructed from live sensor data. Once this dynamic local map is constructed, it is transformed into the PTEM. The PTEM is integrated with guidance algorithm to develop the navigation and guidance model. The unmanned ground vehicle is integrated with sensors for the implementation aspect of this project. All the sensors integrated are calibrated for accuracy, reliability, and repeatability.

CHAPTER 2

RELATED WORK

2.1 Introduction

Mobile robot navigation in an unknown and changing environment with uncertainties is one of the most challenging problems in robotics. There are multiple approaches one can follow to solve the problem of obstacle avoidance and path planning for a mobile robot. These approaches can be roughly divided into global or local approaches, offline or online approaches or a hybrid of any of the above. Each approach has its own pros and cons. For real-time autonomous navigation, the robot should be capable of sensing its environment, interpreting the sensed information to obtain the knowledge of its position and the environment. While performing this primary function, the robot also has to plan a real-time route from its current position to target positions while performing obstacle avoidance by controlling the robot's direction and velocity.

2.2 Autonomous Systems

Recent technological advances have enabled the development of unmanned vehicular systems and recent implementations have proven their benefits in both military and civilian applications. The full benefits of unmanned systems will be utilized when they can operate autonomously [2]. To achieve the objective of absolute autonomy both the hardware system and the control software algorithm must work in flawless conjunction and account for every or most of errors. So, while autonomous systems are gaining more and more popularity amongst researchers and military, the

controls dilemma is far from being solved. A lot of novel and innovative approaches to solve this problem have surfaced in the last five years; some of these solutions are identified in [7, 8]. The recent advances in autonomous self-piloting UAVs can be found in [9]. Path planning and environment mapping are a few of the biggest problems in autonomous systems. Neural networks and neural-fuzzy approaches to these problems are presented in [10].

2.3 Environment Model and Map Construction

For the autonomous vehicle to complete its missions successfully it is essential for it to fully interpret the environment in which it operates and functions. This is where building a robust model of the environment is highly essential for the success of an autonomous vehicle. Construction of a map is one of the prime objectives of an autonomous vehicle since all further operations and decisions reside on the accuracy and precision of the built map. There are multiple approaches of constructing a model of the environment. These approaches can be roughly divided into global or local approaches, offline or online approaches or a hybrid of any of the above. In our particular case, the obstacles might not be known prior to the start of mission. Thus, there is a growing interest in finding different approaches to deal with the uncertainties embedded in the information obtained during the mission. This leads naturally to considering probabilistic models to mathematically formulate the area of operation and incorporate the uncertainties. The most common method for building a probabilistic map of an uncertain area of operation is grid-based occupancy maps, which assign probabilities of adversary/target existence to the individual cells of the grid [11, 12]. Since the full knowledge of the environment is not always available to the planner, it is more realistic to use sets that describe the range of possible values. Most global approaches however is applied to a static environment where obstacles

are static. To deal with dynamic or moving obstacles mostly a local approach is opted for. Like in [13] due to the nature of dynamic obstacles the environment is modeled while the vehicle navigates through the environment. In such cases often zero or minimal information about the environment is known prior to the mission.

Classifications are also done based on the type of sensors used to model the environment. Sensors are mostly chosen on the basis of the amount of features essential to operation of the autonomous system. In [14] sonar is used to map the environment since the vehicle only needs to avoid the obstacles in the locality. While in [15] 2-DOF camera is used to create a map, since the respective projects require the information like color and height of the obstacle. A variety of sensors can be used to model the environment like in [16] a LRF [1] is used and in [17] an IR [2] sensor is used.

2.4 Path Planning

Most of the CPU cycles are utilized in computing the optimal path for autonomous navigation. The success of path planning depends largely on the precision with which the environment is modeled. The path planning algorithms are further classified as *online*[3] or *offline* [4] algorithms. This classification is exclusively based on how the algorithm calculates the path to achieve the goal while avoiding obstacles.

Path planning is either combined with obstacle avoidance or is added as a separate layer on top. Here one has to compensate between optimal and accurate path planning and the computational complexity and time involved in the process of path planning. A very popular approach is A* [18]; for its map representation A* utilizes a grid based search area divided into squares, it starts in an open point

[1] Laser Range Finder

[2] Infrared Sensor

[3] Path is calculated while the platform is on the move or in real-time

[4] Path is calculated before the mission starts i.e. not in real time

and searches are done in all direction representing a '*'; this is done for every point and a cost function is evaluated and the least cost path is selected. Many people use different graph theory approaches. Visibility Graph [19] is one such approach which considers obstacles as polygons and draws straight lines from start to target to achieve its goals. While a Voronoi Diagram [20] considers arcs instead of straight lines to find the shortest path towards the target, the obstacles are still represented by a hexagon.

Local approaches on the other hand are used for reactive obstacle avoidance. They consider only a small frame or model of the overall environment to produce control signals for robot. This comes with an obvious disadvantage that they cannot produce global solutions. However, their key advantage is that they come with less computational complexity and so they are one of the obvious choices when it comes to real-time implementation, where the local maps have to be updated with sensor data. The Vector Field Histogram (VFH) [21], employs a 2D histogram grid to represent the environment. The environment is reduced to a single dimension polar histogram which is built around the position of the robot at a certain moment in time. A decision is then taken to move into the sector with the least number of obstacles. VFH+ [22] is an extension to VFH and introduces some novelties by employing threshold hysteresis to improve the shape of the trajectory, and the use of a cost function. Local approaches are also implemented using a varying widow size called the Dynamic Window [23] approach. The method might be described by a search for commands computing the velocities of the vehicle which are then passed to the velocity space. The robot's trajectory consists of a sequence of circular arcs. It differs from VFH and VFH+ in the manner that it avoids dealing with the kinematics and dynamic constraints of the robot. These methods are extremely fast and typically consider only a small subset of the obstacles close to the robot.

2.5 Multi-sensor Integration

Sensors are used to sense the environment in which the autonomous vehicle operates as well as the status of the autonomous vehicle itself. Every autonomous system has to rely on the data from its sensors to carry out complex decisions. Every sensor produces erroneous data under some physical and environmental conditions, so no system should completely rely on the output of one sensor.

There are two main problems when dealing with multiple sensors on a platform. The first problem is data sampling or data rate. When the measurements are obtained from multiple sensors [24], there are two cases to be distinguished. In the first case, sensors operate synchronously, which means that the sampling times of all the sensors are the same. In the second, which is a more realistic case, sensors operate asynchronously [25] and provide data at different rates with different communication delays. When the measurements are obtained asynchronously in a centralized tracking algorithm, the possibility that the measurements are received in the wrong order due to time delays must also be considered. Different approaches for dealing with this problem have been presented in the literature [26]. This requires special consideration in carrying out the update. If the measurements originate from different sensors, then the sampling interval might be negative in general. This is called negative-time measurement update. The second problem is isolating the bad or erroneous sensor readings from the correct ones. This is an enormous problem when dealing with multiple sensors. Different approaches have been identified in [27] that deal with the problem. A lot of real-world practices involve sensor fusion and data estimation. These techniques are used to obtain repeatable and accurate data as shown in [28]. Multiple sensors are a necessity, but they have to be validated and used appropriately.

2.6 Simulation and Physical System Modeling

Simulation is a vital tool used in the development of autonomous systems. An autonomous system is a complex unit with many variables and unknowns. Simulating individual parts and units of the system can help identify flaws in its design or implementation. It is vital in the sense that simulation provides a controlled environment, where the number of parameters to be varied can be controlled so testing of each and every parameter can be done before testing the system in an actual environment. It also plays a pivotal role in the process since some of the systems developed are too expensive and uncertain to test directly in the real world.

To simulate different modules of the system, a simulation environment for a UAV is developed like in [29]. Here different control signals are passed to the UAV physical model through a simulation environment. The outputs and control signals are logged and recorded for analysis. A workstation is responsible for simulating the UAV physical model while the guidance algorithm is simulated on the implementation processor itself [30]. Such a simulation is known as a Processor In Loop (PIL) simulation. In this particular case, the simulation accurately predicts the timings and data synchronization issues for an actual implementation.

To simulate any system, the fundamental behavior of the system must be known and modeled into equations which describe the system. Unmanned Systems tend to be very complex and cannot always be modeled by simple equations. However a model consisting of several simple physical models representing small subsystems can sometimes be developed. Such equations are known as static and dynamic [31], [32] models of the system. The success of simulation largely depends on how well the system is modeled. The basic rules and fundamentals for modeling complex systems must be understood in order to model the system. Often, these models involve forming

and stating assumptions. In [33], the author describes how an autonomous vehicle is modeled and simulated from scratch.

2.7 Real-Time Implementation Techniques

Our project involves real-time implementation of the algorithms developed onto an unmanned system. Many projects/research ventures are halted as simulation being the last step and they rarely get implemented. Implementation of algorithms requires a different set of framework. Now, as there are rapid advances in processor technology rapid prototype modeling, such processes allow rapid test and implementation of algorithms developed.

Implementation on a physical platform is accompanied by a different set of constraints and requirements. Often these constraints are not considered while developing the algorithms. As a result, it might be necessary to modify or adapt the algorithms to meet different hardware requirements.

This can be seen in [34] where a test bed was created for unmanned systems design by designing a reconfigurable micro-avionics system that allows rapid-prototyping and testing of such aerobatics experiments across a wide range of vehicle types. The control algorithm was implemented on a MPC555 board, which has a higher level support of Matlab and Simulink, which makes implementation of controls algorithms developed really simple. In the above case a custom design based implementation was done; in many cases, to test the functionalities of a control algorithm it is implemented on a COTS [5] robot or test bed.

As indicated in [1], a real-time test bed labeled P3-DS1401 was constructed for real-time control verification. It was constructed on the reconstructive P3-DX robots

[5] Commercial Off The Shelf

of ActivMedia Inc. This particular implementation is done on a custom robot to produce a real-time test bed to identify the real-time constraints on controls algorithms.

CHAPTER 3

SYSTEM ARCHITECTURE

3.1 Introduction

This chapter describes the fundamental architecture of the system. The chapter describes how the different components of the system interact with each other rather than describing the function and description of each individual element. The system architecture has been divided into two major categories: the first one is how the hardware elements/components used during this research project interact with each other, while the other deals with the software aspect of the implementation and how different software/algorithmic components interact with each other, the hardware system, and the user.

3.2 Hardware System Architecture

This sections deals with how the hardware components interact with each other. It describes the hardware system architecture used in the project in detail. Figure 3.1 describes the hardware architecture developed during this research project. The originally proposed hardware architecture was of a single computing platform consisting of MPC555 as the sole controller interfacing with all the sensors and the physical platform. All the sensors we are using in the project requires either a RS-232, USB or TTL communications interface. Data string parsing using Matlab-Simulink toolchain was not successfully implemented. Since, only two RS-232 serial ports on the MPC555 are available, we had to change the architecture to the present model.

As Figure 3.1 shows, a distributed and modular architecture has been adopted for this particular project. There are plenty of advantages/reasons/motivation to adopt this particular scheme of architecture. The hardware architecture is built as a set of embedded micro-controllers connected over Controller Area Network (CAN), i.e. it is a purely distributed and therefore scalable. Even though this is a simple scheme it offers a number of benefits that motivates its selection in our application domain.

As shown in the Figure 3.1 the GPS receiver, LRF and Digital Compass are connected to PlugaPods^{TM}. The MPC555 micro-controller receives the parsed sensor data over the CAN network. As mentioned above, all the sensors use RS-232 serial communication protocol. The PlugaPod^{TM} has one RS-232 serial port. The PlugaPod^{TM} can reliably function with 4 soft UARTs [1]. Soft UARTs work on TTL level (+5V) serial communication. By using a RS-232 to TTL card all the RS-232 (+12V) serial data coming in from the sensors can be converted into TTL level. So potentially, we can use just one PlugaPod^{TM} to parse the serial data stings from all the three different sensors. But, the serial data string provided by different sensors is of different length . Each sensor parsing data string routine forms an individual thread in the IsoMax programming environment. In our case if we decide to connect three sensors to a single PlugaPod^{TM}, three independent threads will be needed in IsoMax to successfully run the application. In IsoMax different run times cannot be assigned for different threads. If we use a single PlugaPod^{TM}, the data from all the three sensors will be parsed at the rate of the slowest sensor. In our case, to parse one LRF scan serial data string it takes 2 to 3 seconds, but to parse a digital compass serial data string it only takes 0.1 to 0.2 seconds. So, if we use a single PlugaPod^{TM} the digital compass updates will be provided at 2 to 3 seconds. To avoid this delay

[1] Universal Asynchronous Receiver Transmitter

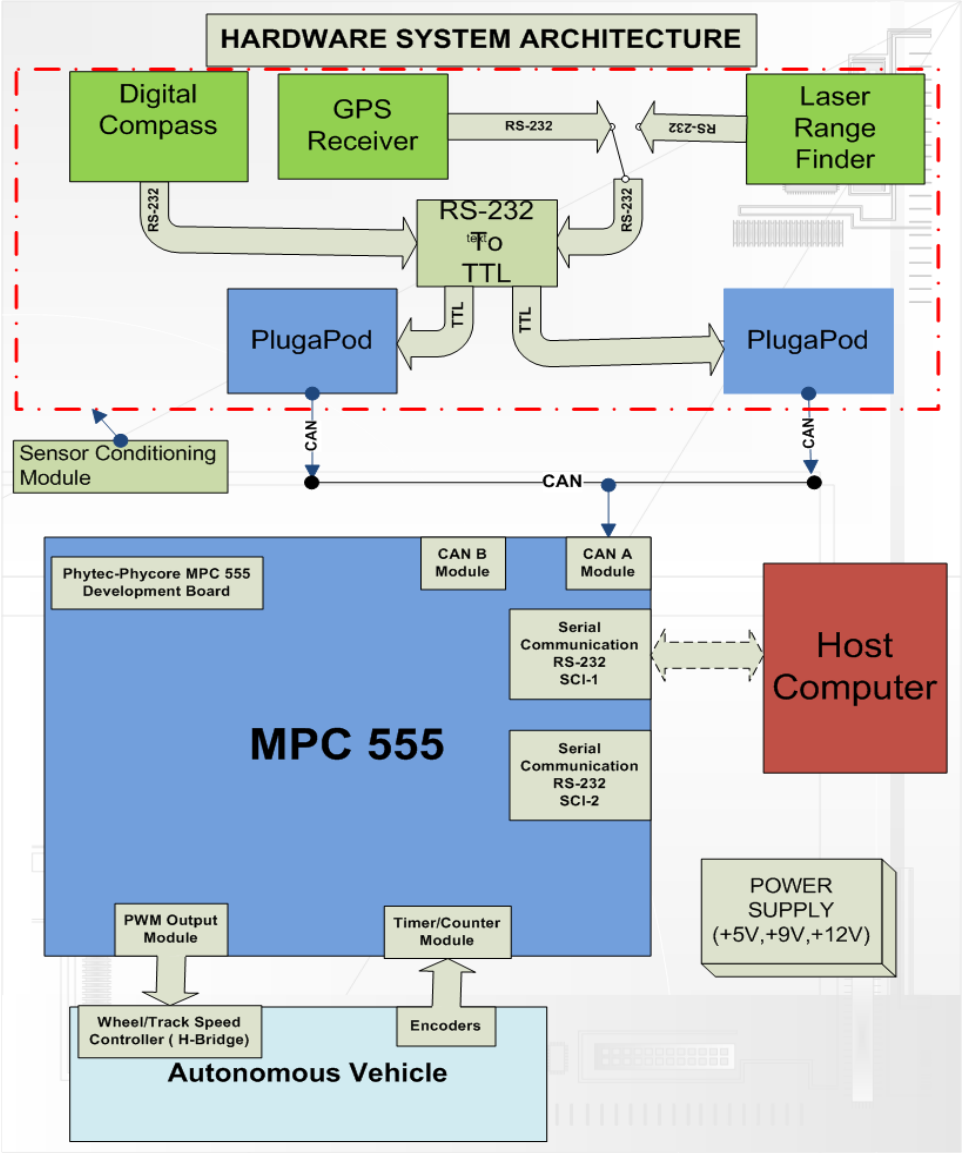

Figure 3.1. Hardware Architecture.

and utilize the faster update rates of different sensors, we have decided to dedicate one processor to each sensor. We are using 2 PlugaPod^{TM} for data string parsing. One PlugaPod^{TM} is dedicated to the digital compass for string parsing. The LRF and GPS receiver share a PlugaPod^{TM} for data string parsing. Since, we do not use LRF and GPS receiver together in the current setup we share a PlugaPod^{TM} for string

parsing. The switch between the GPS receiver and LRF is done by connecting the respective outputs of serial communication port to the TTL card input port. If the need arises to use a GPS receiver and LRF together, an additional PlugaPod^{TM} can easily added to the current setup without any code or hardware modifications. We also require one serial port on each processor to interact with the application, so we are using soft UARTs. Soft UARTs operate at TTL voltage level and thus a RS-232 to TTL converter is required.

The MPC555 receives data in the form of CAN messages and makes control decisions based on these data. The output from the MPC555 processor is in the from of PWM signals applied to the H-bridge which controls the wheels of tank. The power supply for the whole system is generated from a common battery. The battery supply is given to different voltage regulators to generate different voltages. Each voltage regulator is also protected by a separate individual fast blow fuse. Serial communication is used to communicate between the MPC555 micro-controller and the development workstation.

The high level of modularity of a CAN architecture offers extreme flexibility to select the actual type of processor to be used in each sub-module. Different processors can be used according to functional requirements, and they can be scaled according to computational needs of the application. Module interconnection is an additional extra benefit because the complex interconnection schemes needed by parallel buses do not fit properly with the space and weight limitations in a mini/micro UAV, which is our ultimate application target.

Finally, development simplicity is the main advantage of this architecture. By using techniques such as distributed computing, the computational requirements of a single do-all computer can be reduced/eliminated. These modules are depicted in Figure 3.1.

3.3 Distributed Computing

In our hardware architecture organization, the computational requirements have been distributed amongst several computers/micro-controllers. In our particular case we are using 3 micro-controllers to distribute the computational requirements.

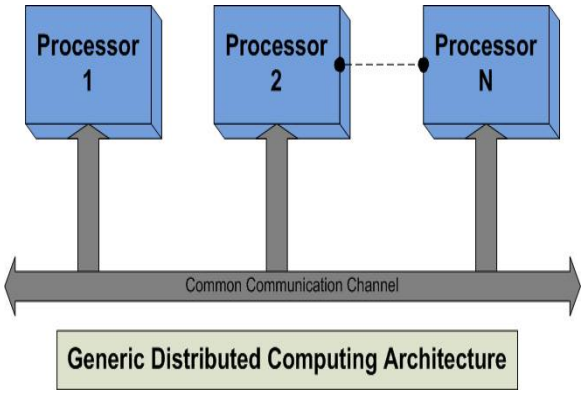

Figure 3.2. Generic Distributed Computing Architecture.

Figure 3.2 gives a generic model of a distributed computing architecture. Here up to N different processors can communicate over a common communication channel. Some examples of this common communication channel are local area network (LAN), controller area network (CAN), serial, I2C, serial peripheral interface (SPI). The common communication channels between the 3 processors available were CAN, SPI and serial. Serial communication port (RS-232) is required to communicate with the sensors but SPI is not supported by Matlab target support toolbox. Hence, in our particular case we have adopted CAN as the common communication channel for our distributed computing.

3.4 Controller Area Network (CAN)

Controller area network (CAN) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. Bit rates up to 1 Mbit/s are possible at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g. 125 kbit/s at 500 m). CAN was developed originally in 1983 at **Robert Bosch**. More information about this protocol and its related literature can be found in [35].

In this research project implementation, we are using a 3-wire CAN network, with CAN-H, CAN-L and GND being high, low and ground wires respectively. The data rate of the CAN bus selected must be greater than the maximum amount of data transfered over the network per second. In our project, the maximum updates possible from the GPS sensor is at 20 Hz, the laser range finder 5 Hz and the digital compass at 400 Hz. We require four CAN messages for each GPS reading, a maximum of 769 CAN messages for each LRF scan and one CAN message for each digital compass reading. So, the maximum number of messages that need to be passed over the CAN network each second is 4325 and the maximum size of each message is 8 bytes long. So, the maximum bit rate required to achieve this communication is 276.8 Kbits/sec. As a result, we have adopted CAN-bus CAN 2.0 at 500 Kbits/sec, since it is more than enough for our sensors in our current implementation. If needed, we have the functionality to operate the CAN-bus at 1 Mbits/sec.

3.5 Software System Design/Architecture

This section describes the software strategy adopted for this project. It will also provide a brief overview of the software tools used in the project. As shown in Figure 3.3 and also explained in Section 3.3, we have adopted a distributed com-

puting architecture approach. The software design/architecture is also influenced by this distributed computing strategy. The software design is partitioned into different modules, each module has specific inputs and/or outputs and is responsible for performing a certain task. Partitioning the software into different modules provides easier debugging and organization control over the software.

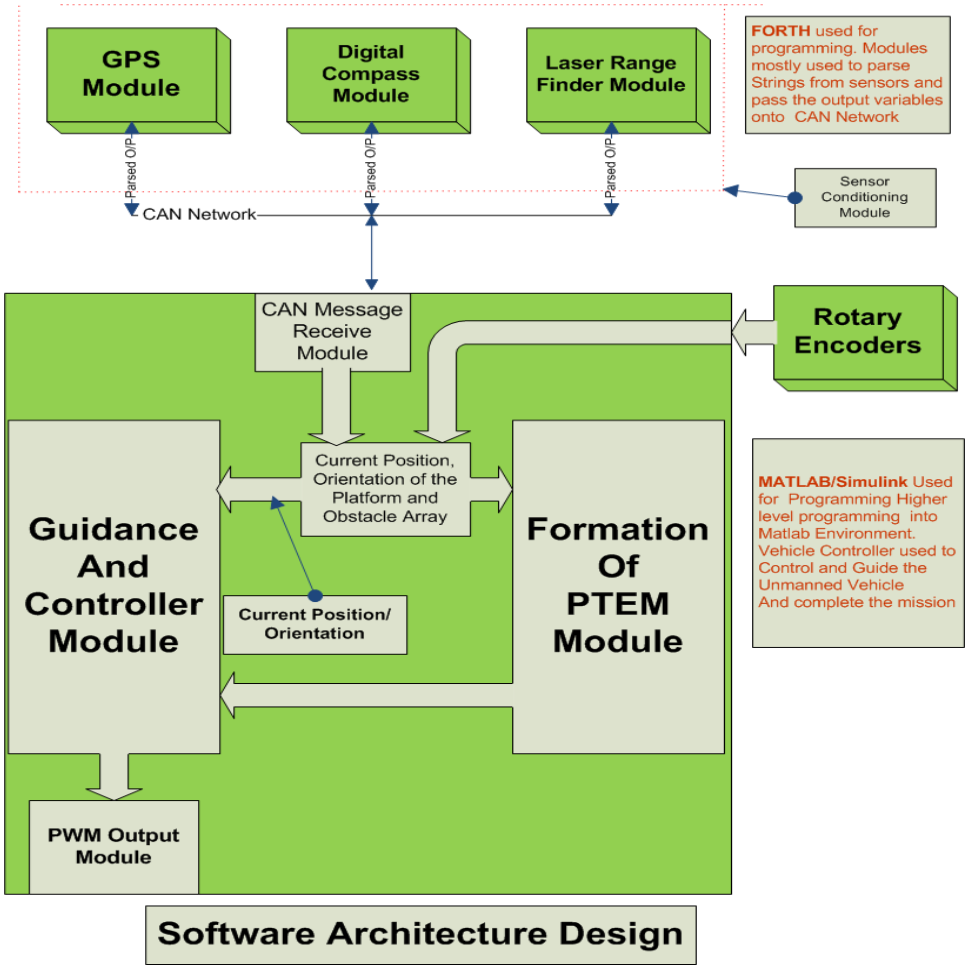

Figure 3.3. Software System Design/Architecture.

3.5.1 Sensor Conditioning Module

The GPS, LRF and digital compass modules are combined to make up the sensor conditioning module. The input to this module are the raw sensor data strings from the sensors with their own proprietary or standard formats. The output from this sensor modules are often the parsed sensor variables required by the control algorithm. The software implementation for the sensor modules is done in Forth programming language. It is a stack-based language and has a very small code signature making it ideal for processes which have limited resources or want optimized code. More information about the Forth programming language can be found in [36].

The NovAtel GPS receiver generates multiple output logs. These logs contain on array of data elements like: latitude, longitude, time stamp, heading, velocity, etc. Each log type contains a specific set of data elements. GPS outputs are also generated in various data formats. Examples include: NMEA [2] messages, NovAtel proprietary logs, and RTK [3] logs. In our project implementation, we have selected the NMEA GPGGA [4] [37] log. We are logging this data string at 2 Hz (the maximum rate possible from the NovAtel GPS receiver is 20Hz). This string is then parsed by a PlugaPod^{TM} where variables such as latitude and longitude are extracted from the data log. These variables are then packaged in to CAN messages (two per variable) and send over the CAN network to the MPC555 micro-controller.

The GS Revolution digital compass is connected to PlugaPod^{TM}. This digital compass can provide multiple data strings/logs including NMEA data logs. Since, our navigation frame deals with true north [5] instead of magnetic north [6] we have selected

[2] National Marine Electronic Association

[3] Real Time Kinematics

[4] Global Positioning System Fix Data

[5] It is the direction along the earth's surface towards the geographic North Pole [38]

[6] North direction indicated by Earth's Magnetic field

the NMEA HDT [7] data log. The true heading information is parsed by an IsoPod^{TM}, packed into CAN messages that are sent out on the CAN network to the MPC555 processor. The GS Revolution is capable of delivering output at 400 logs/sec but, we are using it at 5 logs/sec. Each digital compass message to be transmitted over the CAN network requires one CAN message.

PlugaPod^{TM} also parses data coming from the Hokuyo LRF. The LRF has its own proprietary data format. The data strings are first converted into normal integer values. The integer data is then packed into a series of CAN messages and sent over the network. The number of CAN messages sent depends upon the number of obstacle points detected in one scan. For each obstacle point detected one CAN message is generated consisting of the radial distance and the angle of the radial line.

3.5.2 Vehicle Control Module

The entire implementation of this module is done in Matlab/Simulink environment and physically implemented on a MPC555 micro-controller. This module comprises of the control and guidance algorithms developed. The inputs to this module are the parsed data from the sensor conditioning modules. The commanded speed and heading control signals are the outputs of this module and are sent to the vehicle in the form of PWM signals after being mapped into right and left wheel speeds. All the algorithms are implemented in Matlab and Simulink. The Real-Time Workshop is used to generate C code form the simulink model. We are using the Code Warrior V8.7 Development Studio [39] as the C compiler along with Real-Time Workshop. The C code generated by the Real-Time Workshop is converted in to a binary image or hex file by Code Warrior. This hex file is then downloaded on to the MPC555 micro-controller over a serial communications link.

[7] True Vessel Heading

The Vehicle Control module has five sub-modules:-

1. CAN Message Receive Module
2. Current Position/Orientation Module
3. PTEM Construction Module
4. Guidance and Control Module
5. PWM Output Module

3.5.2.1 CAN Message Receive Module

This module is responsible for receiving all the CAN messages sent over the network from the IsoPod^{TM} and PlugaPod^{TM} micro-controllers. This module is responsible for unpacking all CAN messages containing the sensor input data and converting the sensor values into Matlab data types which can be used by the rest of the Simulink model. The data from rotary encoders is also received in this module.

3.5.2.2 Current Position/Orientation Module

This module receives inputs in the form of sensor status variables from the CAN message receive module. It is responsible for combining this CAN messages to represent the current position, current orientation and current array of obstacles detected. This information is then passed onto the next module for processing. The data from the rotary encoders and digital compass are utilized to obtain current position estimate. This is done through a process called *dead reckoning* that is described in detail in Chapter 6, Section 6.2.2.

3.5.2.3 PTEM Construction Module

This module is basically responsible for constructing an online map of the environment in which the vehicle is operating. It takes inputs from the GPS receiver and

the digital compass to localize the platform in the navigation frame. It uses the data from the LRF to identify the obstacles and map them into the navigation frame (NF). The NF will be described in detail in Chapter 5, Section 5.2.1. The map constructed is used as a framework to make further guidance and control decisions. It is therefore essential to have as accurate model of the environment as possible. The map is then passed on to the guidance and control module.

3.5.2.4 Guidance and Control Module

This module fundamentally defines how the guidance and control algorithm is implemented. The primary inputs to this module are the PTEM and the current orientation/position of the platform. It provides outputs in the form of control signals which are given to the platform/vehicle. These control signals control the orientation and velocity of the platform/vehicle.

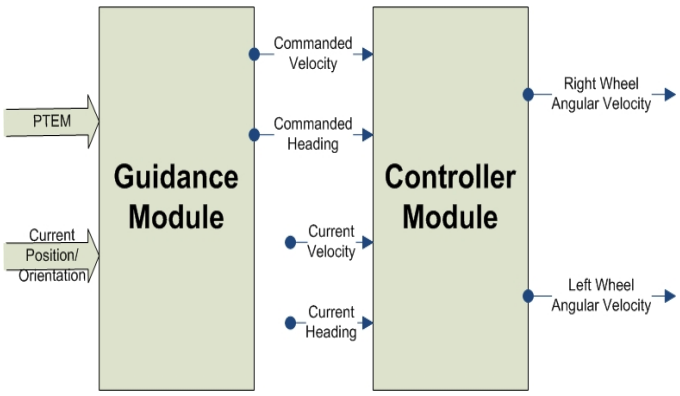

Figure 3.4. Guidance And Control Module.

As shown in Figure 3.4, the module contains two sub-modules: the Guidance module and Controller module. The Guidance module is responsible for taking current position/orientation data and the PTEM for generating commanded heading and

commanded velocity control signals. These control signals are taken as input by the Controller module along with current velocity and current heading. The Controller module uses this data to generate the angular velocity control signals for left and right vehicle tracks. These angular velocity control signals are used as inputs by the PWM output module.

3.5.2.5 PWM Output Module

The angular velocity signals generated by the Controller module are the inputs to the PWM module. This module converts these signals into four PWM outputs representing: right forward, right backward, left forward and left backward. These outputs are sent to the vehicle's motion control subsystem that is responsible for moving the platform.

This was a very brief description of all the modules that form the Vehicle control module. The mathematical formulation and construction of the PTEM are described in Chapter 5. The software implementation of the vehicle control module as a Simulink model is described in detail in Chapter 6.

CHAPTER 4

TOOLCHAIN DESCRIPTION AND IMPLEMENTATION HARDWARE

4.1 Introduction

The implementation aspect of the research deals with hardware and associated software components. This chapter provides a detailed description of all the hardware components including all the sensors and micro-controllers used in the project. It also provides a detailed description of the toolchain used for programming and simulation.

4.2 Hardware Components

Most of the hardware used, is Commercial Off The Shelf (COTS). We have not custom designed any hardware component.

4.2.1 Sensors

This section gives a detailed description of the sensors used in the project. It also describes the function of each individual sensor in detail.

4.2.1.1 Laser Range Finder

One of the key challenges of autonomous navigation is obstacle avoidance. To perform obstacle avoidance, the obstacles have to be identified or sensed. In this research project, a scanning laser range finder is used to detect the obstacles. Other potential obstacle detection sensors include: IR, ultra-sonic, and visual sensors.

During this research project, the Hokuyo URG-04LX scanning laser range finder (LRF) is used to detect obstacles. Figure 4.1 shows the Hokuyo LRF and Figure 4.2 shows two obstacles detected by the laser range finder (LRF).

Figure 4.1. Hokuyo URG-04LX Scanning Laser Range Finder.

The Hokuyo URG-04LX has a 270-degree scanning range and 240-degree sense range with 0.36 degree angular resolution. It has USB and Serial (RS-232) interfaces for maximum flexibility. Its accuracy is +/- 20 millimeters. The scan time is 100 milliseconds. The maximum range of the Hokuyo is 4 meters and the minimum sense range is 20 millimeters. More information on the Hokuyo LRF can be found in [40].

The Hokuyo is connected to an IsoPod^{TM} through a serial cable which parses the output, converts it from the propriety format into angle and distance readings. The IsoPod^{TM} issues the command to initiate the scan. One scan takes 100 milliseconds, once the scan is complete, the data from a complete scan is transmitted to the IsoPod^{TM} via a serial cable. Every scan point is represented by two bytes of data in the scan (a maximum of 769x2 bytes per scan). The data is encoded using a Hokuyo proprietary data structure and encoding scheme. The IsoPod^{TM} parses the scanned data into two fields: the angle of an obstacle point and the radius or distance to the

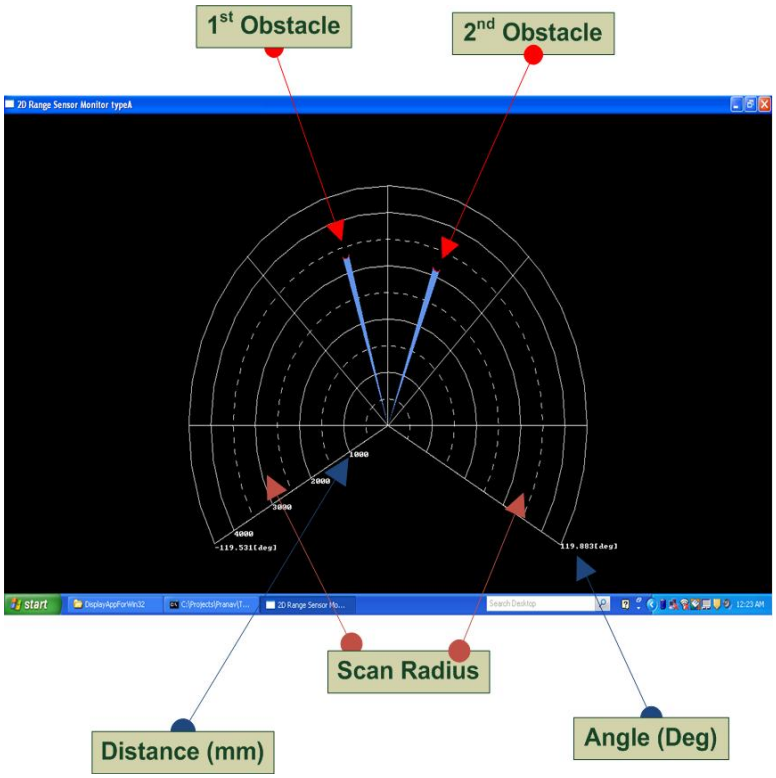

Figure 4.2. Obstacle Detection By Laser Range Finder.

detected obstacle point. Both variables are packed into one CAN message and sent over to the CAN network. A special CAN message is sent over the network before initiating a new scan. This data is passed over the CAN network to the MPC555 for further processing.

4.2.1.2 Global Positioning System (GPS)

Localization is a crucial aspect of autonomy. Localization can be global, local or both. To achieve effective localization the platform must know its precise location with respect to a fixed frame. One can define its own frame, or use standard frames already developed. Localization can be done effectively with combining inputs from multiple sensors; it can also be performed by using an accurate single sensor.

The implementation demands local localization, i.e. we have to create a local map of the environment. For this we are using a GPS receiver as our primary localization sensor. We are using a Differential GPS (DGPS) system implemented using Real-Time Kinematics (RTK). Differential GPS implementation requires a base station and a mobile or rover station. The base station can be fixed or mobile, a fixed base station provides more accuracy. The base station is a NovAtel Propak using a OEM4-L GPS card shown in Figure 4.3 and uses the antenna shown in Figure 4.4.

Figure 4.3. NovAtel ProPak Enclosure.

Figure 4.4. NovAtel L1/L2 Antenna.

Figure 4.5. NovAtel OEM-G2L GPS Card.

Figure 4.6. NovAtel L1/L2 Antenna.

The mobile/rover station is more compact and has lesser functionality compared to the base station. The mobile station contains a NovAtel OEM4 GPS-2L card as the GPS receiver shown in Figure 4.5 and uses the mobile antenna shown in Figure 4.6. Figure 4.7 shows a custom housing and interface board created for the rover GPS unit. The RTCM-V3 format is used to implement the RTK functionality. By implementing DGPS we aim to getting an accuracy of 2 cm CEP [1] while we are static and we get an dynamic accuracy of 10-15 cm CEP.

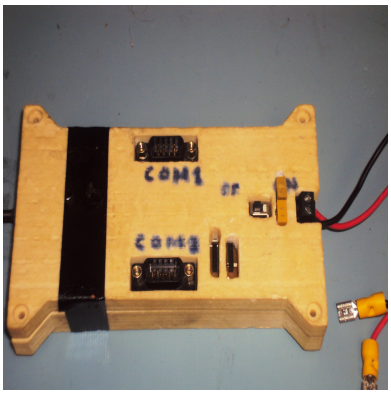

Figure 4.7. OEM G2L GPS Receiver Enclosure.

Figure 4.8. XTend RF Serial Modem.

[1] Circular Error Probable

To implement DGPS a communication link between the two GPS receivers must be established. This is achieved by connecting an Xtend RF Modem to an auxiliary serial port on each of the GPS receivers. Figure 4.8 shows the Xtend module used. This unit has a range of about 4 miles, which is more than enough for our particular implementation. More information about the NovAtel Systems and the DGPS implementation can be found in [41].

The NovAtel Propak G2 plus is the fixed base station while the custom designed OEM G2-L card is the rover station or mobile station mounted on the platform itself. RTK can be implemented using various standards such as RTCA, RTCM, RTCMv3, CMR, CMR+. We have decided to use RTCMv3 [2] [42] since it is the most efficient amongst all the standards and is also supported by both the NovAtel receivers. The RTCM format (RTCM, 2001) requires about 4800 bits per second (bps) to broadcast dual-frequency code and carrier-phase observations or observation corrections of 12 satellites. The same information content is sent with about 1800 bps in the newer RTCM v3 format (RTCM, 2006) [42]. This means less latency time, less bandwidth requirements and faster updates.

We are using RTCM1006 and RTCM1003 corrections to implement RTK. Once we log this corrections out of the base station, the rover station has to be initialized to receive them. We are using the GPGGARTK log [41] at the base station which is similar to GPGGA but has extra precision bits of RTK. We are outputting this log at 2 Hz (Maximum 20 Hz possible).

4.2.1.3 Rotary Encoders

In addition to GPS sensors, encoders are used to provide displacement data used in Dead Reckoning algorithm which estimates the position, speed and orientation of

[2] Radio Technical Commission for Maritime Services

the platform. A rotary encoder is an electro-mechanical device that converts the angular shaft position into digital pulses. The rotary encoder is mechanically connected to the shaft of the wheel of the platform. One rotation of the wheel corresponds to one rotation of the shaft. Each rotation of shaft generates a specific number of digital pulses from the encoder. Each rotary encoder has two output channels. In addition, each encoder requires a power supply (+5V and ground). The output channels of the rotary encoder are connected to the timer processing unit (TPU) of the MPC555 micro-controller. The TPU is accompanied by a digital counter. Every shaft rotation either increments or decrements the counter by a specific number of counts depending upon the direction of rotation. The counter reading corresponds to the current digital reading of the rotary encoder.

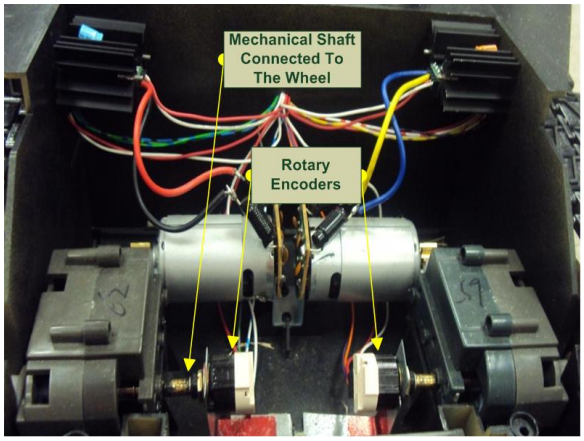

Figure 4.9. Rotary Encoders.

The Figure 4.9 shows the rotary encoders connected to the output shaft of the motor gear box. Since the encoder is attached to the output shaft of the gearbox it directly measures the angular rotation to the drive sprocket of the vehicle's track. Linear displacement can be calculated using the radius of the sprocket and angular rotation of the sprocket. The vehicle used in this research is a tracked differential

drive platform. Each motor gearbox has its own rotary encoder. The encoders are supported by an L-shaped bracket. The position of encoders are fixed mechanically.

4.2.1.4 Digital Compass

We need to know the current orientation of the platform in our project. Vehicle orientation can be measured by different sensors like an IMU [3], a digital compass, or can be estimated from GPS readings under the assumption that the vehicle heading and orientation are the same. In our project, a digital compass is used to identify the orientation of the platform. We are using the GS Revolution digital compass.

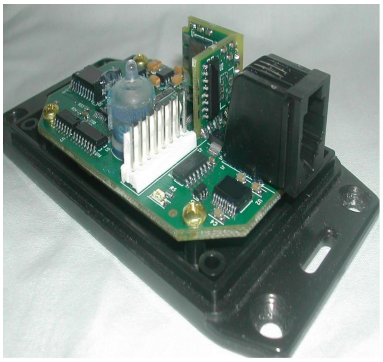

Figure 4.10. GS Revolution Digital Compass.

Figure 4.10 and 4.11 shows the GS Revolution compass. It can provide heading, pitch, roll, and accelerations along the x,y and z axes in a variety of formats ranging from NMEA to binary or proprietary ASCII outputs. It also has two independent full duplex serial communication channels. The proprietary GS Revolution software allows hard iron and soft iron calibration routines to be performed. The sensor provides a lot of output data strings like HCHDG, HCHDT, binary and NMEA 0183

[3]Inertial Measurement Unit

Figure 4.11. GS Revolution Digital Compass Enclosure.

strings. The dynamic local map is constructed in navigational frame (NF). NF uses true north as reference. HCHDT data string is used as it provides heading with respect to true north. The GS revolution data string is outputted at 5 Hz.

The orientation values from the digital compass are also used to construct the PTEM. The obstacles detected in the body fixed frame (BFF) by the LRF are mapped into the navigational frame (NF) using the orientation data from the compass and the position data. The position data is provided by the GPS sensors or is calculated using the dead reckoning algorithms.

4.2.2 Computing Resources

This section discusses the processors used in the project. An modular and distributed computing approach was elected which requires multiple processors. The processors used can be divided into two categories. The first category of processors are responsible for parsing serial data strings coming from the sensors and generating message packets containing sensor data. The second category of processor implements all of the control algorithms. This processor will be referred to as the 'Master Control Processor' in this document.

4.2.2.1 String Parsing Processors

In our design we have a processor dedicated to each of the sensors. The primary responsibility of these processors is to parse the data strings coming from the sensors and pass the data over the CAN network to the Master Control Processor.

Figure 4.12. IsoPod^{TM}.

Figure 4.13. PlugaPod^{TM}.

Figures 4.12 and 4.13 represent the two types of processors that can be used for data string parsing. Since we are devoting one processor to each sensor. We are using two string parsing processors. As explained earlier, one PlugaPodTM is dedicated to digital compass while the LRF and GPS reciever share one PlugaPodTM. Both the processors, IsoPod^{TM} and PlugaPodTM are from New Micros Inc. and use the same operating system and user interface. Each of these processors are provided with a factory installed application development language and environment called IsoMaxTM. IsoMax^{TM} is a real-time operating system and language. The IsoMax^{TM} environment provides an interpreted software development environment that allows a user to communicate with the processor in the foreground while simultaneously running an application in the background. This capability facilitates dynamic on-line debugging and application in the background [43].

The IsoMax^{TM} language is built upon the Forth programming language. The Forth language is a stack-based, interpreted, interactive operating system and application development language [36]. The Forth operating system was designed to develop fast and efficient real-time software applications using limited computing resources. IsoMax^{TM} is built on the top of the Forth environment and retains all the features and capabilities of Forth. In addition, IsoMax^{TM} provides a Virtually Parallel Machine Architecture that supports the development of multitasking, real-time, embedded control applications using very lean computing resources.

IsoMax^{TM} facilitates the creation of these multi-tasking embedded applications by providing the mechanisms within the language to define multiple Finite state machines that execute concurrently within the same application. The finite state machine structure provided by the IsoMax^{TM} environment were used extensively in the embedded, real-time sensor data from the system sensors to the Master Control processor via the CAN network.

The application originally developed by Randy Dumse at New Micros Inc., to parse the GPS receiver serial data feeds provided the best example of this functionality. This application was designed to provide a fault-tolerant system capable of processing a complex multi-field serial data string in an environment where data string corruption was likely. The application must also permit other real-time tasks to run in parallel with such as motor control functions or the handling of other external sensor driven events. The original GPS serial string parsing application was designed to extract relevant data fields from the NMEA GPGGA and the GPRMC serial data strings. The application was designed to ensure the integrity of the entire string before attempting to remove the values of interest. Foremost in the application's design was the requirement that the processor could not be 'captured' by the

parsing routine in the event of a serial communications failure or the corruption of the data string.

The parsing application utilizes five parallel finite state machines. All the defined state machines with an IsoMax^{TM} system have access to global memory and read/write access to all system variables. This capability enables inter-process control and communication between the various Finite-State-Machines defined within the IsoMax^{TM} system. This feature enables a level of concurrency and control not typically available on an embedded micro-controllers within the 16-bit Motorola DSP56G800 class. In fact, the IsoPod^{TM} and PlugaPod^{TM} controllers were included in this project's hardware architecture because the 32-bit Motorola MPC555 microprocessor and its associated Matlab/Simulink based Development environment, presented in Section 4.2.2.2, could not provide adequate real-time string parsing capabilities.

IsoPod has two RS232 ports while PlugaPod only has one RS-232 port. They both have CAN 2.0 capability and floating point math capability. PlugaPod uses a Motorola DSP56F803 while an IsoPod uses a Motorola DSP56F805 processor. We have implemented bootup autostart on both the processors, so that they can act in stand-alone mode. More information regarding the processors and programming language can be found in [43].

4.2.2.2 Master Control Processor

The Master Control Processor is the heart of the processing unit. The primary responsibilities of the Master Control Processor are:-

- Accumulating various sensor values over the CAN network
- Construction of the PTEM based on current and past data
- Giving Control signals to the platform to navigate through the mission

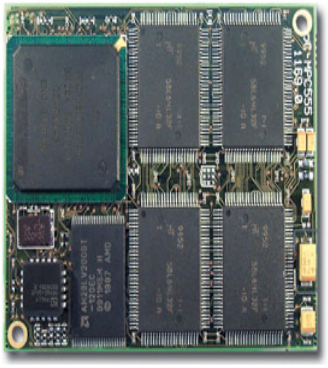

Figure 4.14. Motorola MPC555 Module With Flash/RAM.

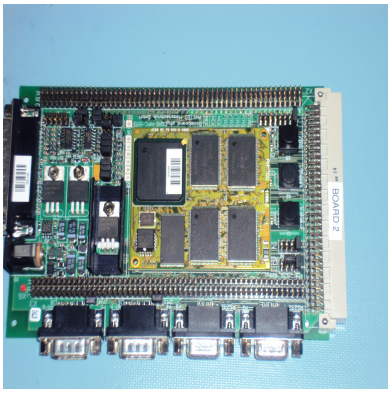

Figure 4.15. Phytec Phycore MPC555.

The Master Control Processor is shown in Figure 4.14. We are using a Motorola MPC555 as the Master Control Processor, the board shown in Figure 4.15 is a rapid prototype development board developed by Phytec called Phytec-Phycore MPC555. We chose this particular board since it is supported by the Mathworks toolchain, i.e the lower level drivers have been implemented in C and are available as a blocks in the FM5 Target for the MPC555 toolbox. The whole toolchain is described in Section 4.3. The MPC555 is a high-speed 32-bit Central Processing Unit that contains

a floating point unit designed to accelerate the advanced algorithms necessary to support complex controls applications.

It is based on Power PC Architecture and supports a wide range of on-board peripherals. It has two fully functional RS-232 ports and two CAN 2.0 ports. It also has multiple PWM signal generation blocks and I/O modules. More information about the MPC55 processor and Phytec-Phycore MPC555 development board can be found in [44, 45].

4.3 Software Development and Simulation Toolchain

The choice of a software development toolchain is a very important decision for a project like ours because that selection will determine how the algorithms will be implemented, how much time it will take to implement those algorithms, and also how easy will it be to simulate and test. The choice of our toolchain was based on two factors:-

- The toolchain must be related to Matlab/Simulink since all our algorithms were developed and simulated in Matlab/Simulink.
- We also wanted a toolchain that provided an automated low level code generation process.

On this basis, we selected the Mathworks toolchain comprising of Matlab, Simulink, Real-Time Workshop, and Real-Time Embedded Coder. Based on the processor, we have selected FM5 target support for the MPC555 toolbox. Figure 4.16 gives an overview of the structure and flow diagram for the Matlab-Simulink toolchain[46].

The Matlab-Simulink toolchain has the following features:-

- The Real-Time Workshop Embedded Coder product generates production code for use on the target MPC555 micro-controller.

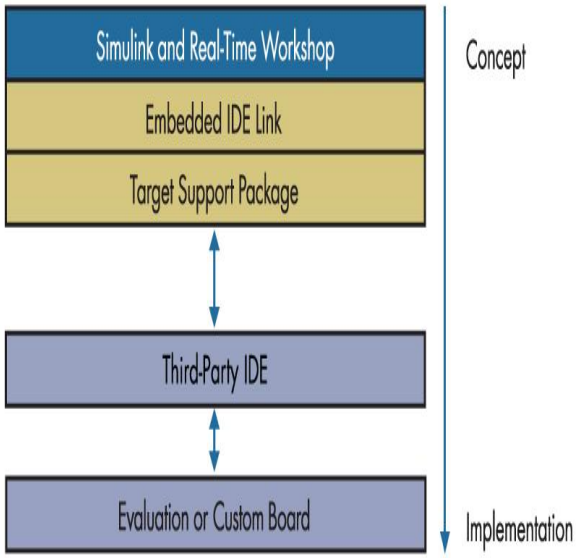

Figure 4.16. Software Development Toolchain.

- The Target Support Package Library provides device driver blocks so we can access on-chip resources.
- Generation of real-time, stand-alone code for MPC555.
- Software-in-the-loop (SIL) simulation and Processor-in-the-loop (PIL) both supported by the toolchain.
- Real Time Debugging Using BDM communications.

By using this toolchain, we can automate the lower level code generation process from higher level matlab/simulink structure. More information is available in [46]

4.4 Implementation Platform

This section describes the vehicle platform used for the implementation of the thesis project. The platform is a small plastic tank, as shown in Figures 4.17 and 4.18. It is a differential drive or skid steer platform. This class of vehicle has two drive motors. The vehicle turns by creating a speed difference between the two tracks.

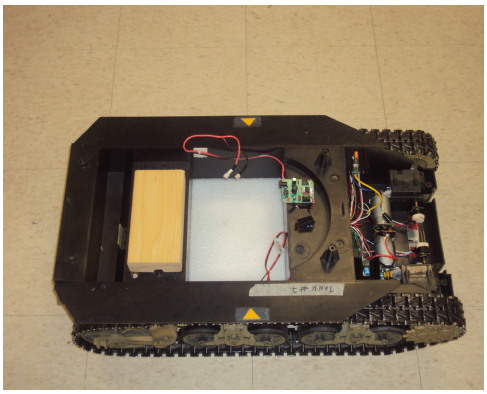

Figure 4.17. Hardware Implementation Platform (Tank).

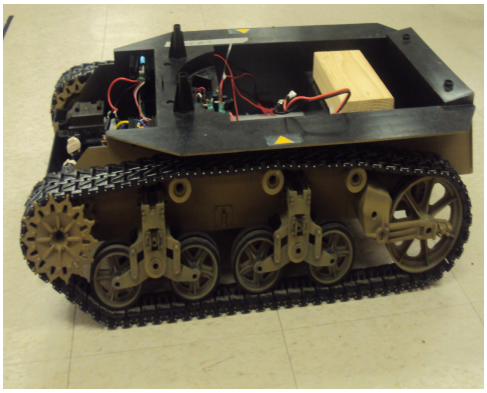

Figure 4.18. Side View of the Tank.

Though the mechanical body of the tank was purchased, all the electronics were added to make it functional. Figure 4.19 refers to the electronics added in the tank to make it functional. The basic electronics added are H-bridges to drive the DC motors, and encoders to provide feedback related to angular displacement of the gearbox output shaft used to drive each of the vehicles tracks.

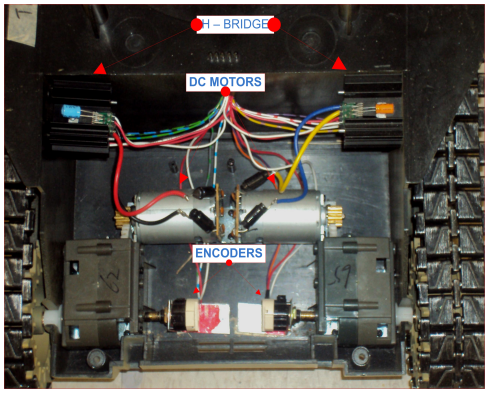

Figure 4.19. Electronics added to the Tank for Basic Motion.

Figure 4.20 shows the top view of the tank after all the electronics were mounted.

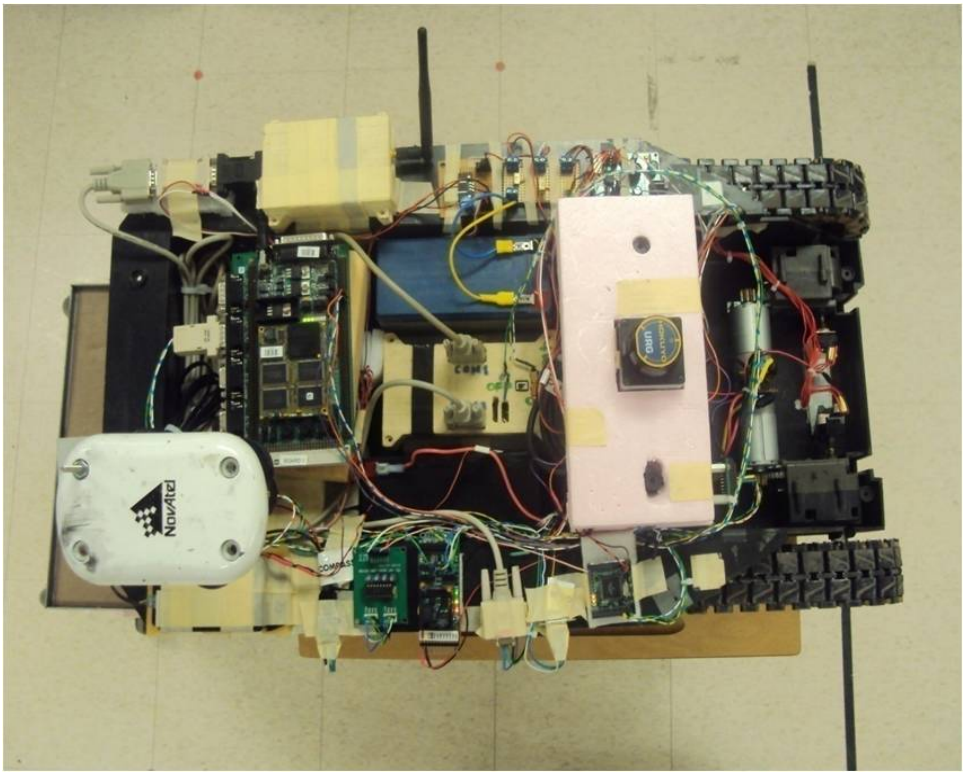

Figure 4.20. Top View after Mounting All the Sensors.

The tank shown above has also been modeled in the Matlab-Simulink environment through the construction of a kinematics-based model of the tank and its sensors. The model of the tank is described in Chapter 5.

CHAPTER 5

CONSTRUCTION OF THE ENVIRONMENT MAP AND COLLISION AVOIDANCE ALGORITHM

5.1 Introduction

This chapter describes the formulation and construction of a map of the area in which the vehicle operates. It also briefly describes the collision avoidance algorithm developed by Zengin [2]. This chapter describes in detail the algorithms used to construct the map, the problems encountered while constructing the map, and the approach used to construct a sample map from the sensor data. The map is constructed in 2-D space. The map, once constructed, is transformed into Probabilistic Threat Exposure Map (PTEM). The obstacles encountered during the mission are referred to and modeled as *threats*. The threats are represented by Gaussian distributions. The PTEM is defined by the of means and variances of all the threats identified. Chapter 7 will describe the experiments used to test the accuracy of the map construction and will also provide the results of the experiment.

5.2 Formulation of the Local Map and the PTEM

This section provides the mathematical formulation for construction of a local map and the transformation of this local map into the PTEM. The local map is constructed using live input data from the sensors in 2-D space. The position, orientation, and obstacle detection sensors are required to construct the local map.

5.2.1 Formulation of the Local Map

To construct a map, a fixed reference frame is needed in which the whole map is constructed.

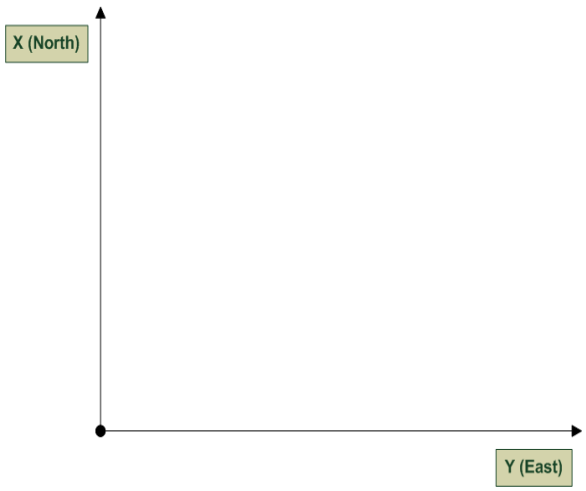

Figure 5.1. Coordinate System Used During the Project.

Figure 5.1 shows the coordinate system used during this research project. The reference frame presented here is a standard navigational frame used for unmanned systems. It defines X as the true North direction, Y as the East direction and the Z axis is projected into the ground. Angles are measured with respect to X-axis. The positive rotation around Z is in the clockwise direction, i.e from X towards Y. This frame is important as all the mathematical formulations and calculations are developed with respect to this navigational frame.

A local map, once constructed, provides a common framework for all obstacles detected in a local frame. In order to construct the local map three elements are required:

- Local Position of the platform/vehicle
- Orientation of the platform/vehicle

- Obstacles identified and mapped into the navigational frame

In our project, we refer to the local frame as the **navigation frame (NF)**, since all the navigation/motion is going to be defined with respect to this frame.

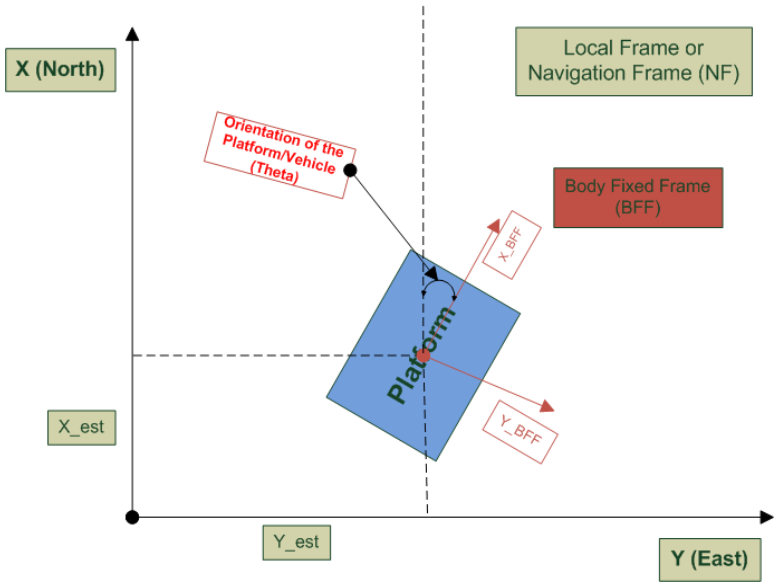

Figure 5.2. Navigation Frame (NF) and Body Fixed Frame (BFF).

Figure 5.2 shows the **Navigation Frame (NF)**. NF is a 2-D frame with 3-DOF of motion allowed; rotation is performed along Z axis as described earlier. The whole local map is constructed using the NF as a reference frame. The second reference frame is the **Body Fixed Frame (BFF)**, which is fixed on the vehicle platform and thus moves and rotates with the vehicle relative to NF.

The position of the platform is sensed by the position sensor and is directly mapped into NF. The orientation of the platform (θ) is shown in the Figure 5.2. The orientation of the platform is sensed by the orientation sensor (digital compass). The obstacle detection sensor is mounted in a fixed position and orientation on the platform. Since the platform rotates relative to NF, the obstacle detection sensor also

rotates along with the platform. The obstacles are identified in BFF. The orientation θ information of the platform is used in mapping the obstacles detected in BFF to NF.

Construction of the PTEM requires the current position of the platform. The current position of the platform can be obtained in two ways: directly, through the use of the GPS sensor, or by performing dead reckoning calculations with the inputs obtained from the Digital Compass and the rotary encoders. The current position of the vehicle is calculated using dead reckoning. This process is explained in detail in Chapter 6, Section 6.2.2. The other position sensor i.e. GPS will provide the current latitude and longitude values. This latitude and longitude values are then converted into UTM [1] co-ordinates and the X_{est} and Y_{est} values are calculated. This process will be explained in detail in Chapter 6, Section 6.2.2.

Figure 5.3 shows how a point on an obstacle is detected by the LRF and is mapped into NF. Here, X_{est} and Y_{est} are current positions of the platform calculated from the GPS or encoder readings. P is the obstacle point detected by the LRF, radius R and angle α represent the point P in BFF . The orientation of the platform is represented by θ and is calculated by the digital compass. To construct the local map, the obstacle has to be transformed from BFF in to NF. For the formulation of local map, we assume that digital compass is placed in line or parallel to the LRF, GPS antenna and LRF are at the same point. The following formulas will help better explain the rotation and transformation process of obstacles from BFF to NF.

$$X_p = R \times \cos(\alpha) \tag{5.1}$$
$$Y_p = R \times \sin(\alpha) \tag{5.2}$$

[1] Universal Transverse Mercator

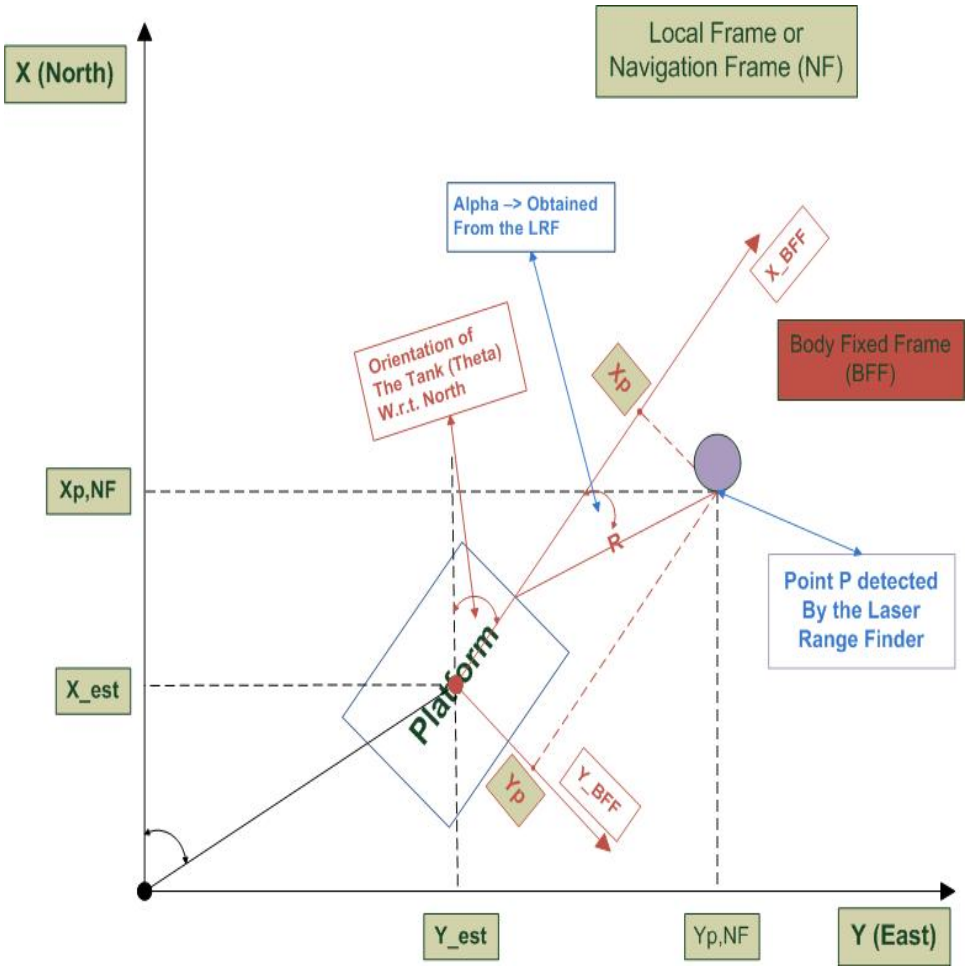

Figure 5.3. Obstacle Detection by LRF .

The position of the obstacle point in NF is calculated by rotating the coordinates (X_p,Y_p) to NF using the 2-D rotation matrix, parameterized by θ, and then translating them to NF by the position of the vehicle platform (X_{est}, Y_{est}).

$$\begin{bmatrix} X_{p,NF} \\ Y_{p,NF} \end{bmatrix} = \begin{bmatrix} X_{est} \\ Y_{est} \end{bmatrix} + \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} R\cos(\alpha) \\ R\sin(\alpha) \end{bmatrix}$$

This matrix equation is written in the scalar form as follows.

$$X_{p,NF} = X_{est} + R\left[\cos(\theta)\cos(\alpha) + \sin(\theta)\sin(\alpha)\right]$$

$$Y_{p,NF} = Y_{est} + R\left[-\sin(\theta)\cos(\alpha) + \cos(\theta)\sin(\alpha)\right]$$

This implies,

$$X_{p,NF} = X_{est} + R\cos(\theta - \alpha) \tag{5.3}$$

$$Y_{p,NF} = Y_{est} - R\sin(\theta - \alpha) \tag{5.4}$$

5.2.2 Formulation of the PTEM

As discussed earlier, there are multiple approaches to path planning. One particular approach to path planning is a probabilistic approach [47, 48]. In this particular approach, the environment is modeled as a continuous probabilistic map. In particular, the map models the obstacles by the sum of Gaussian probability density functions. Different types of obstacles and restricted areas are characterized using the same probabilistic framework. This particular probabilistic map or framework where obstacles are represented by a single or multiple probability density functions is referred to as **PTEM**.

This probabilistic map is not meant to provide the actual map of the area of operation, but to provide a means to plan the trajectory of the UGV to avoid obstacles and accomplish the given mission. A threat/obstacle may be characterized by a single Gaussian PDF[2] or multiple Gaussian PDFs might be required to represent an obstacle. Also, multiple obstacles can be represented by a single Gaussian PDF. There are two parameters needed to fully specify a Gaussian PDF; the mean value specifies the concentration point (location) and the variance specifies the area of influence of the PDF. There is not necessarily one-to-one correspondence between the actual number of obstacles and the Gaussian PDFs used to construct the PTEM.

Once the PTEM is constructed, there is no need to distinguish between different types, sizes of obstacles and their location since the use of the probabilistic map is sufficient for decision making. A position in the area of operation is defined by its

[2]Probability Density Function

vector, $\underline{r}$, relative to the origin of NF. Let r be the representation of vector r, i.e. r = $[xy]^T$, where x and y are the components of vector $\underline{r}$ along the x-axes and y- axes of the reference frame. By using this notation, PTEM equation of an area of operation, modeled by Gaussian distributions, can be written as:

$$f(r) = \sum_{i=1}^{N} \frac{1}{2\pi\sqrt{det(K_i)}} exp[-\frac{1}{2}(r-\mu_i)^T K_i^{-1}(r-\mu_i)] \tag{5.5}$$

The above equation represents well known Multidimensional Gaussian law [2], where μ_i and K_i are the mean vector and the covariance matrix of the ith threat, respectively and defined as:

$$\mu_i = \begin{bmatrix} \mu_{x,i} \\ \mu_{y,i} \end{bmatrix}$$

$$\mathrm{K}_i = \begin{bmatrix} \sigma_{x,i}^2 & 0 \\ 0 & \sigma_{y,i}^2 \end{bmatrix}$$

Since in our case the obstacles are of circular shape we have $\sigma_{x,i}^2 = \sigma_{y,i}^2$. Eq. (5.5) calculates the value of the PTEM at a given position in the area of operation. To identify the location and size of the obstacles within the area of operation, a threshold value, f_r, is defined. Then, the areas where the obstacles reside or the restricted areas are quantified as

$$A_r \quad = \quad \{\underline{r} = [xy]^T | f(r) \geq f_r\} \tag{5.6}$$

which means, the set of positions at which the value of PTEM is greater than or equal to the threshold, f_r. In this research, the threshold value, f_r, is kept constant once it is calculated. This does not mean that the restricted areas are fixed. As the map is updated, i.e. the means and variances are changed, the restricted areas, i.e., the locations and sizes of the obstacles will be updated as well.

5.3 Construction of the Local Map and the PTEM

The previous section provided the mathematical formulation for construction of local map and the PTEM. This section concentrates on how to construct physical map from data provided by the sensors on the platform. It deals with the actual implementation of the algorithms and formulae presented in the previous section. To construct a local map one has to deal with issues such as actual data conditioning, processing issues and timing issues.

5.3.1 Construction of the Local Map

The construction of local map requires the following pieces of information:

1. Current position of the platform in the navigational frame (NF)
2. Current orientation of the platform relative to navigational frame (NF)
3. Location of obstacles with respect to the navigational frame (NF)

The current position can be obtained either by rotary encoders or GPS. Dead reckoning is used to obtain current position from rotary encoders. Once the GPS values are converted into UTM co-ordinates, this UTM co-ordinates provide X and Y location of a particular GPS location. The process of converting the GPS position into UTM is explained in Chapter 6, Section 6.2.2. This position then can be mapped into navigational frame (NF). This provides the first bit of information to construct the PTEM.

The current orientation of the platform/vehicle is identified by the digital compass. Since, we are using HDT data log as mentioned earlier we get the true heading/orientation relative to navigation frame (NF). This provides the second information to construct the PTEM.

Now, to obtain the obstacle data, we use a Laser Range Finder (LRF) as the obstacle detection sensor. Since the LRF is mounted fixed on the body of the platform,

it provides data in BFF. This data has to be transformed into navigation frame (NF) in order to map the obstacles in the local map. This transformation is shown in Section 5.2.1. The equations used for transformation also require the current orientation of the platform for transformation of BFF data into NF.

Since all the parameters are referenced in NF or converted into NF, we can construct the local map. So, by using GPS receiver, LRF, rotary encoders, and digital compass the local map of the surroundings can be constructed.

5.3.2 Construction of the PTEM

The previous section describes how the data are obtained for the construction of the PTEM. The algorithm developed by Zengin [2] operates on PTEM and not the local map. So, we have to transform this local map into PTEM. As elucidated in Section 5.2.2, PTEM is nothing but an array of Gaussian PDF functions. To specify a Gaussian function we require two parameters:-

1. Mean of the function, in our case mean is an [X,Y] vector defined in Section 5.2.2
2. The variance of the function, indicating the area it covers, refer Section 5.2.2

The third parameter required to construct PTEM is the threshold, as described in Eq. (5.6). As mentioned earlier, fixed threshold is employed during the research project. Figure 5.4 illustrates a sample PTEM constructed in simulation environment without applying threshold. Figure 5.5 shows the physical change in shape of PTEM after applying the threshold. So, depending upon the threshold applied, we can construct different shapes representing the restricted regions of PTEM.

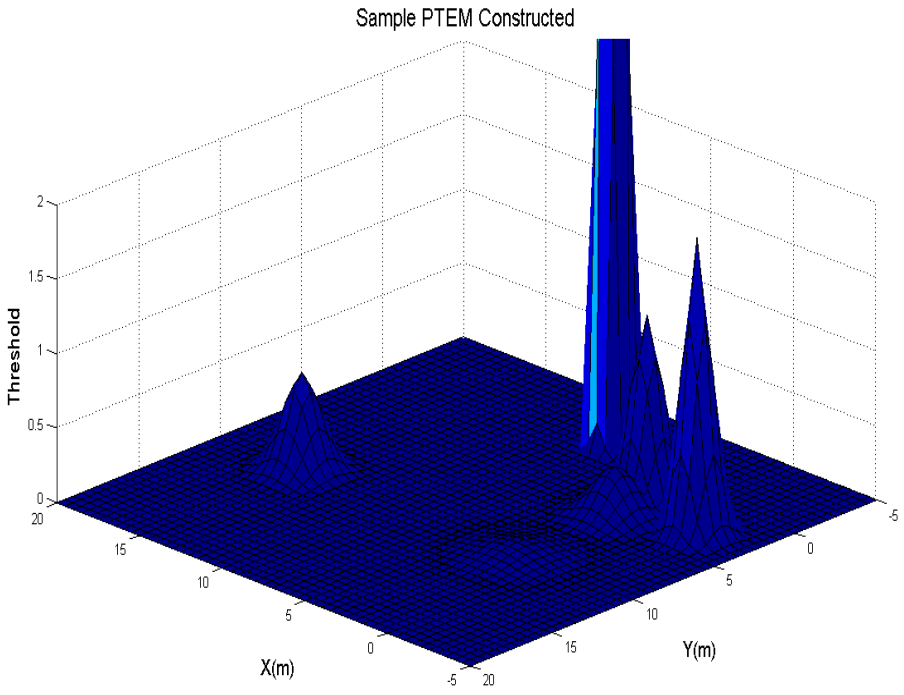

Figure 5.4. Sample PTEM without Applying the Threshold.

5.3.2.1 Clustering of LRF Data

Clustering data is important as the data from the LRF contains multiple points per obstacle detected. If we do not cluster the data points, all the points are needed be individually considered in constructing PTEM and the resulted PTEM will have multiple Gaussian functions representing a single obstacle. Further, clustering also reduces significantly the size of the PTEM and makes the implementation computationally more efficient. A cluster of LRF data points is represented by a circle with the minimum radius that encloses the previous form of the cluster circle and the new data point. The decision to include a single data point into a cluster depends upon the following rules:-

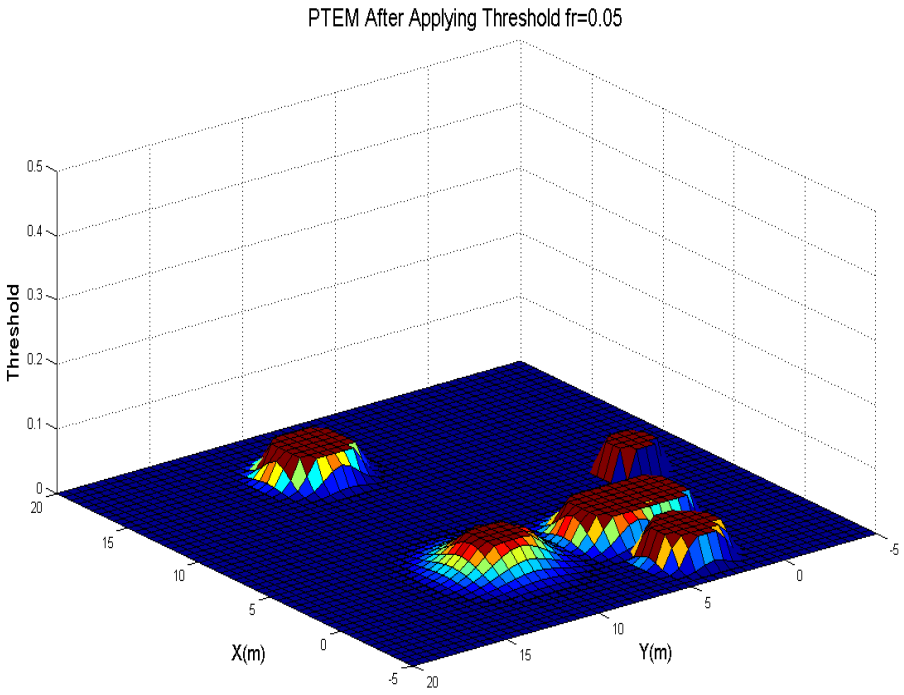

Figure 5.5. PTEM after Applying Threshold.

- The absolute spatial distance between the point (to be included in the cluster) and the boundary of the cluster should be less than a fixed threshold (C_r).
- All the points in a given cluster, do not have to be at a fixed distance from each other.
- Two clusters have to be separated by a distance of more than a fixed threshold. Two clusters cannot overlap each other.
- A map with obstacles at least has one cluster.

The above rules have to be considered while clustering data. The fixed threshold (C_r) in our project is equal to the width of the platform, as the vehicle cannot navigate through two obstacles if the distance between them is less than the width of the

platform. Note that this fixed threshold (C_r) is different from the fixed threshold (f_r) of the PTEM, referred to in Eq. (5.6).

For clustering of data, a hierarchical algorithm is used. Such algorithms can be either agglomerative (bottom-up) or divisive (top-down). Agglomerative algorithms begin with each element as a separate cluster and merge them into successively larger clusters. Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters [49]. A simple form of agglomerative clustering algorithm was developed.

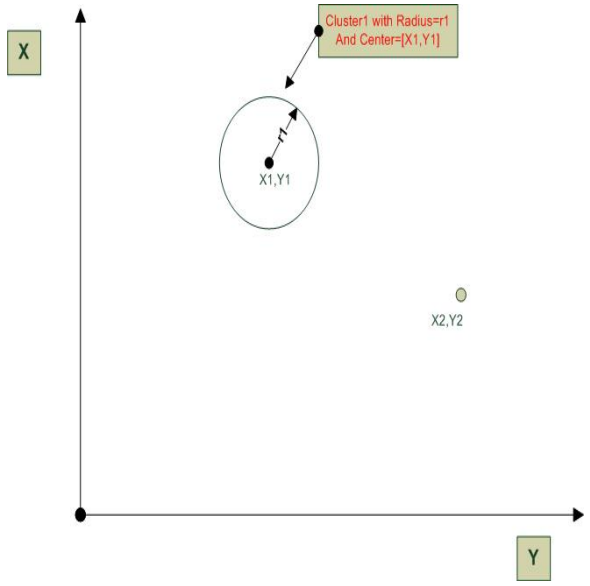

Figure 5.6. Single Cluster and a Single Point.

Figure 5.6 shows a single cluster with center [X1,Y1], radius r1 and a single point P [X2,Y2]. The Figure 5.7 shows how a point P2 [X2,Y2] is added to the cluster. The decision on whether a new point should be added to an existing cluster or a new cluster should be created is made on its distance to the center of the cluster. This distance (d) is given by:

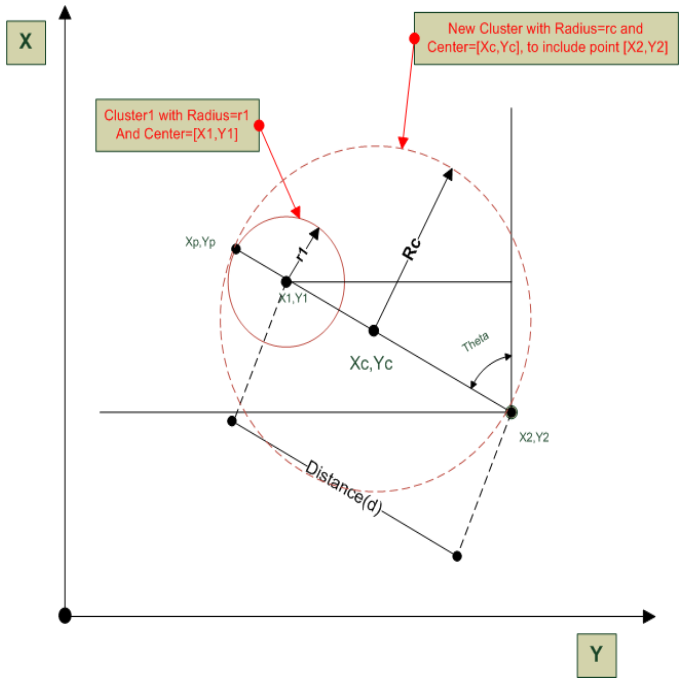

Figure 5.7. Expanding Single Cluster to Include a Single point.

d=$\sqrt{(X2-X1)^2+(Y2-Y1)^2}$

If d $\leq$C$_r$ then, P is included in the cluster. Otherwise, a new cluster is created.

Once the decision regarding the inclusion of the point into a cluster is made, the center and radius of the new cluster circle has to be computed to enclose the new point along with the previous cluster circle.

The geometry depicted in Figure 5.7 implies

$\theta = \arctan \frac{(Y2-Y1)}{(X2-X1)}$

$\alpha = \pi + \theta$

$X_p = r1 \times \cos(\alpha)$

$Y_p = r1 \times \sin(\alpha)$

Thus, the new center [C] is computed as

$$X_c = \frac{(X_p + Y2)}{2}$$
$$Y_c = \frac{(Y_p + Y2)}{2}$$

The new Radius Rc is then calculated as

$$R_c = \frac{\sqrt{(X2 - X_p)^2 + (Y2 - Y_2)^2}}{2}$$

Therefore, we have a growing cluster as new points are added. If a point does not fit into any cluster then, it will be assigned a new individual cluster of its own having the center as the point itself and radius equal to a default threshold ($r_t < C_r$), r_t is default radius assigned to the newly formed clusters, i.e. clusters consisting of only one obstacle point.

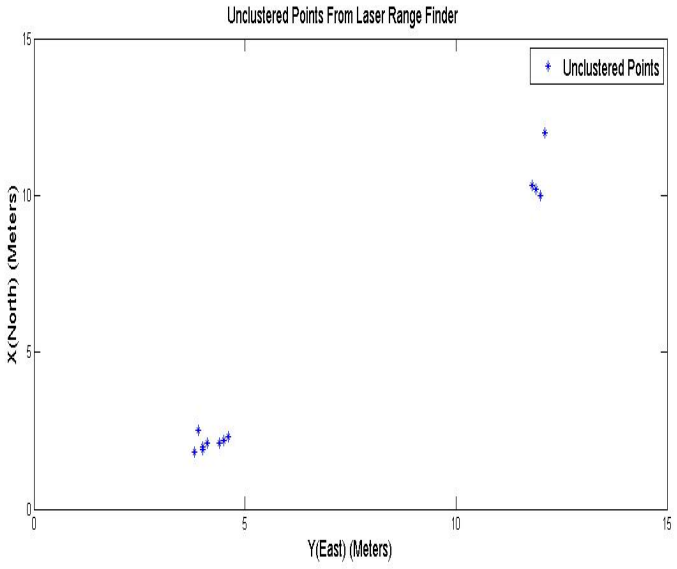

Figure 5.8. Points identified by the laser range finder.

Figure 5.8 shows an example of unclustered points coming from the Laser Range Finder, while Figure 5.9 shows the cluster circles constructed from the unclustered points. Each circle has a center and a radius. From this information, we can find the

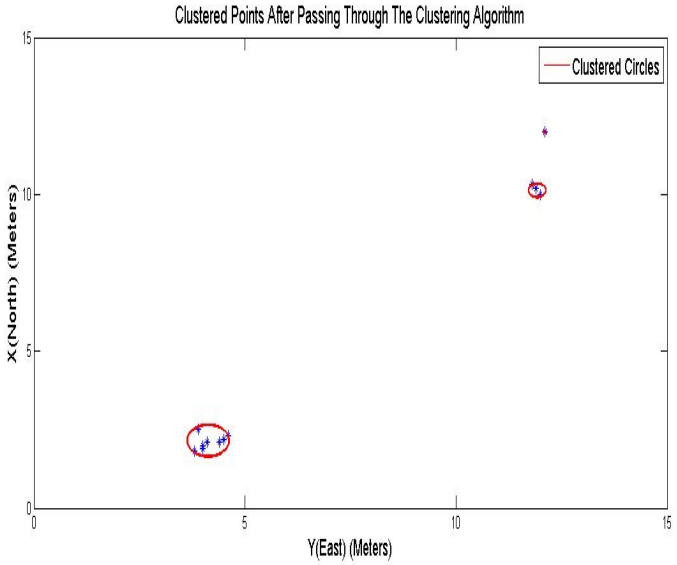

Figure 5.9. Clusters constructed from laser range finder points.

mean and variance of the Gaussian PDF representing each cluster circle as detailed below.

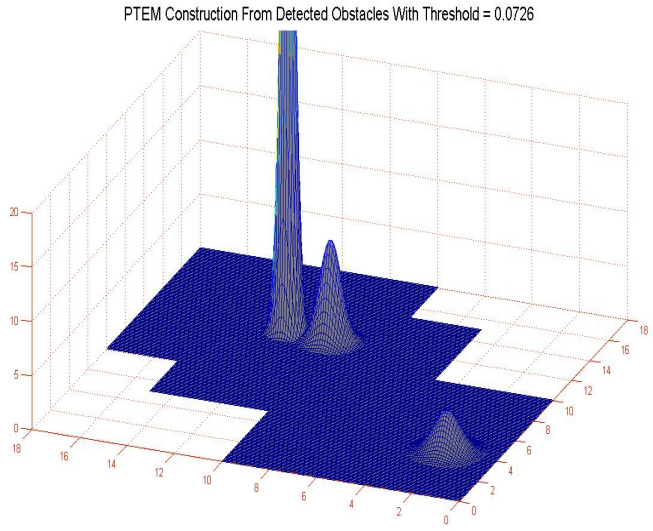

Figure 5.10. Construction of PTEM based on the radius and mean of the clusters.

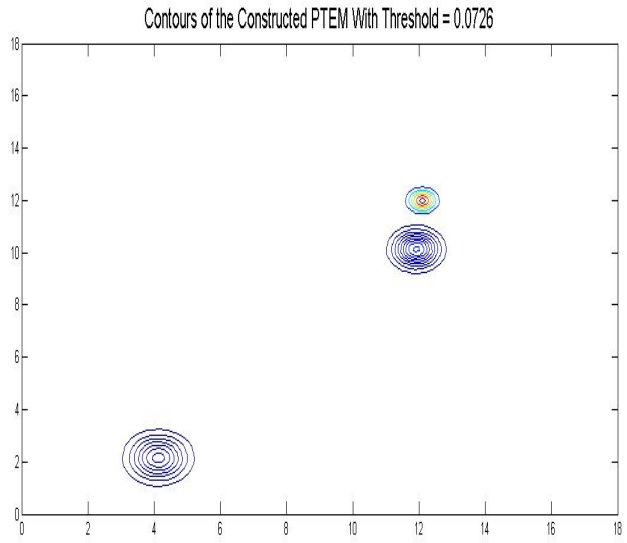

Figure 5.11. Contour of the constructed PTEM.

Figure 5.10 shows the PTEM constructed from the clusters shown in Figure 5.9. The center of each cluster circle is assigned as the mean value of the corresponding Gaussian PDF. The variance of a Gaussian PDF is calculated based on the radius of the cluster circle. The details of this process is given in the next section. Figure 5.11 shows the actual contours of the PTEM. The outer contour of each cluster represents the boundary of the actual restricted area of the PTEM

5.3.2.2 Calculation of the Threshold and Variance

As mentioned earlier, PTEM is formulated as an array of mean and variances. Once clustering is performed, the center and radius of each individual cluster is identified as well as the threshold value of the PTEM. The identification of the restricted areas within the PTEM, a threshold value, f_r, should be set. For this, the first cluster circle is used in Eq. (5.7).

$$f_r = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{r_{ca}^2}{\sigma^2}] \tag{5.7}$$

where r_{ca} is the radius of the first cluster circle and σ is the variance, which is pre-specified for the first cluster. The value of the threshold, f_r, calculated from the first cluster, is fixed and used for calculating the variances of the PDF's corresponding to all the other clusters. For the calculation of the variance, the formula in Eq. (5.7) is rearranged to obtain Eq. (5.8).

$$2\sigma^2 ln(fr\sqrt{2\Pi}) + r_{ca}^2 \quad = \quad 0 \tag{5.8}$$

For the fixed threshold value and the radius of each cluster after the first one, this equation is solved for the corresponding variance. With the threshold, the mean and the variance values, the PTEM is defined.

5.3.3 Data and Logic Flow Diagram for Construction of the PTEM from LRF data

The flow diagrams describes the construction of PTEM from the obstacles points detected by LRF. Figure 5.12 shows a data flow diagram describing the construction of the obstacle array in NF from the obstacle points detected by LRF in BFF. The current position (Xcurr and Ycurr) and orientation (θ) are calculated from the position sensor (GPS or rotary encoders) and orientation sensor (digital compass), respectively. Each scan of the LRF provides obstacle points, if an obstacle is detected. This scan is converted into CAN messages and sent over the CAN network. Each CAN message contains a set of angles (α) and radii (R) representing each obstacle point. The obstacle points are converted into an obstacle array in body fixed frame by 'Convert In To BFF Obstacle Array' function. The outputs of this block

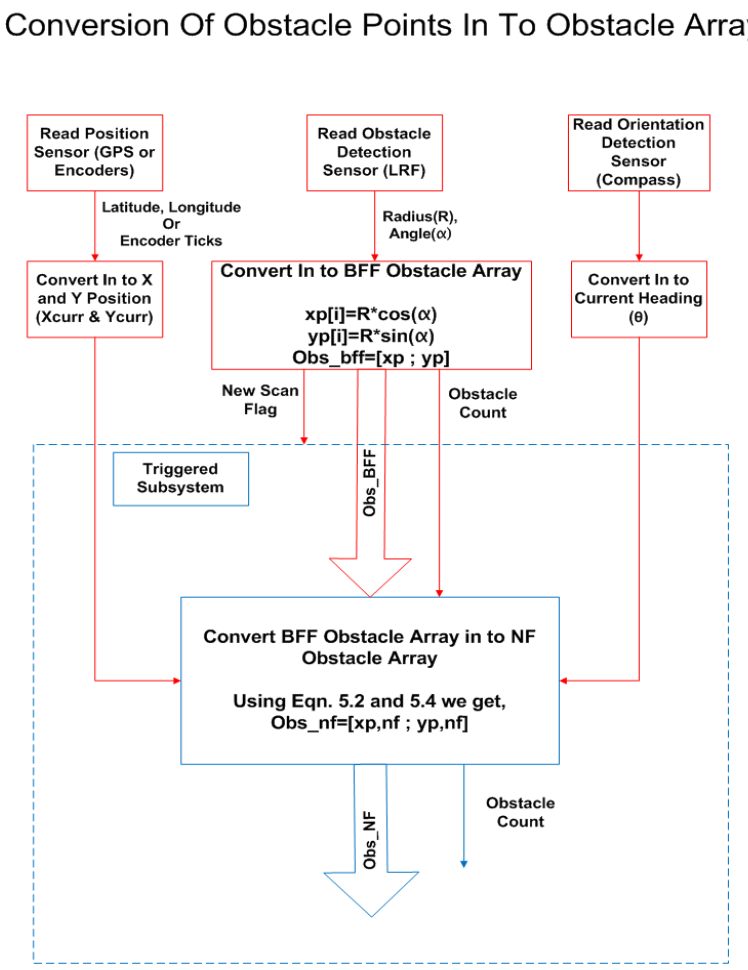

Figure 5.12. Construction Of Obstacle Array In NF From LRF Data.

are obstacle array in BFF, obstacle count and new scan flag indicating the start of a new LRF scan. The triggered subsystem module operates only when it is triggered by the new scan flag. So, every time a new scan is performed the PTEM is constructed. The BFF obstacle array is then converted into NF obstacle array. The mathematical formulae are used in the flow diagram are given in Section 5.2.1.

Figures 5.13 and 5.14 show the data and logic flow diagrams. The inputs to the triggered subsystem are array of obstacle points in NF and the total number of obstacle points. The output is the PTEM consisting of mean and variance. The

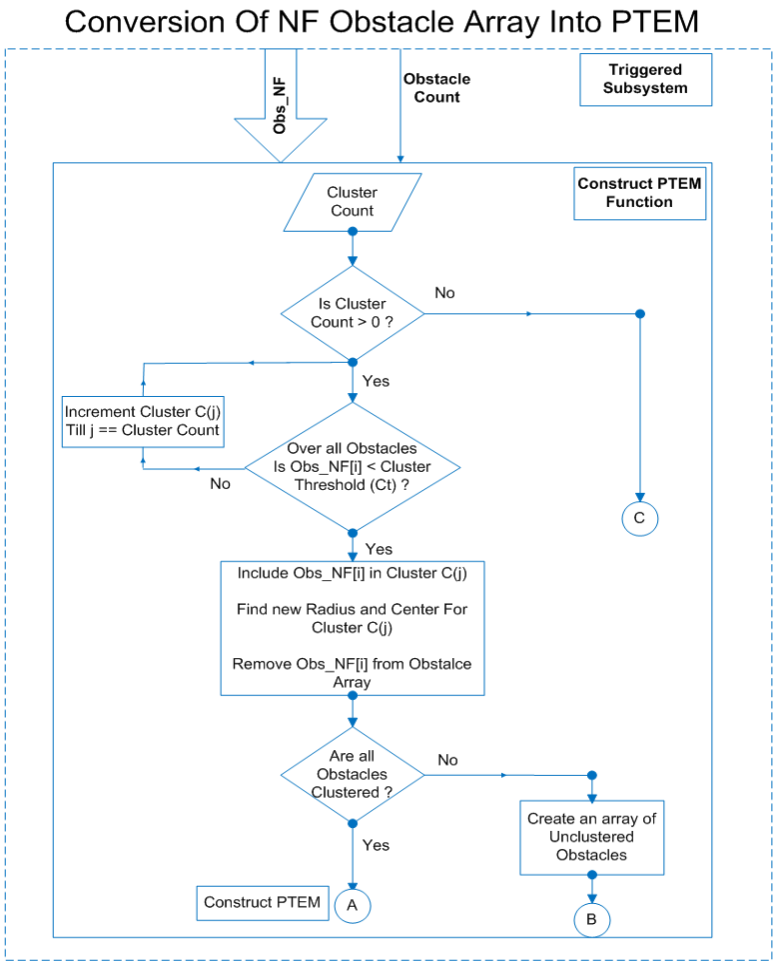

Figure 5.13. Clustering Of Data From NF Obstacle Array.

obstacle points are clustered into different clusters as shown in Section 5.3.2.1. Once the obstacles are clustered, the variance of the each cluster is calculated using the Eq. (5.8). The mean is the center of the cluster. The means and the variances along with the threshold form the PTEM.

5.4 Collision Avoidance Algorithm

Once the PTEM is constructed, the subsequent step is collision avoidance algorithm. Many collision avoidance algorithms were discussed in detail in Chapter 2, all

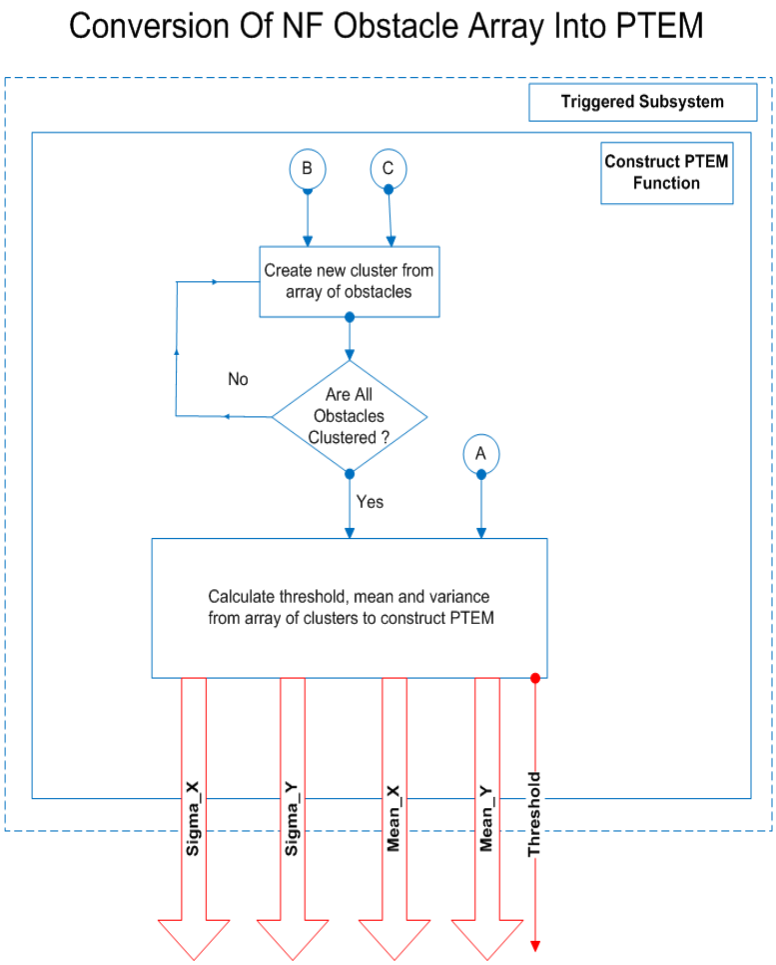

Figure 5.14. Construction Of PTEM From NF Obstacle Array.

have their pros and cons. This part of the implementation is adopted from Zengin's dissertation [2]. Since the PTEM is developed using Gaussian PDF functions, it is differentiable. So, we have adopted Gradient search approach [50, 51] as the collision avoidance algorithm. This determines the direction of minimum increase or steepest descent of the PTEM. In other words, it can be easily determined in which direction the UGV should move to minimize the threat exposure level or maximize the likelihood of avoiding collision with an obstacle.

In the Gradient search approach, we calculate the partial derivative of the PTEM along the X and Y directions to find the smallest descent vector and then the decision is made to move the UGV in that direction or not. This decision is based on a Rule Based expert system developed by Zengin. For more information on how collision avoidance is formulated, refer to Ref. [2].

CHAPTER 6

REAL-TIME SIMULINK MODEL AND SIMULATION MODEL DESCRIPTION

6.1 Introduction

As discussed in Chapters 3 and 4, the Matlab-Simulink toolchain along with the real-time workshop has been employed for the implementation of this research project. A simulation model of the physical tank and sensors is developed to test the algorithms in simulation environment. Further, a real-time navigation and control simulink model of the system is developed to test the system in real world. This model is converted into the executable code and downloaded on the MPC555 for implementation purposes. This chapter provides a detailed description of both the simulink models developed during the research project.

6.2 Real-Time Navigation and Control Simulink Model

This section describes the real-time navigation and control model. It describes the complete procedure, from getting the sensor inputs into the system to providing the output control signals to the wheels of the tank.

The block diagram in Fig. 6.1 shows the overall picture of the real-time navigation and control model developed in Simulink. As observed in this navigation and control model, the flow of data/information is from top to bottom. The different colors in the model indicate the different sample times or update rates of different modules or blocks. The data from the sensors come in the form of encoder readings and CAN messages from IsoPod^{TM}/PlugaPod^{TM}. The encoder data and CAN messages are received in the module 'CAN receive'. This information is then passed on

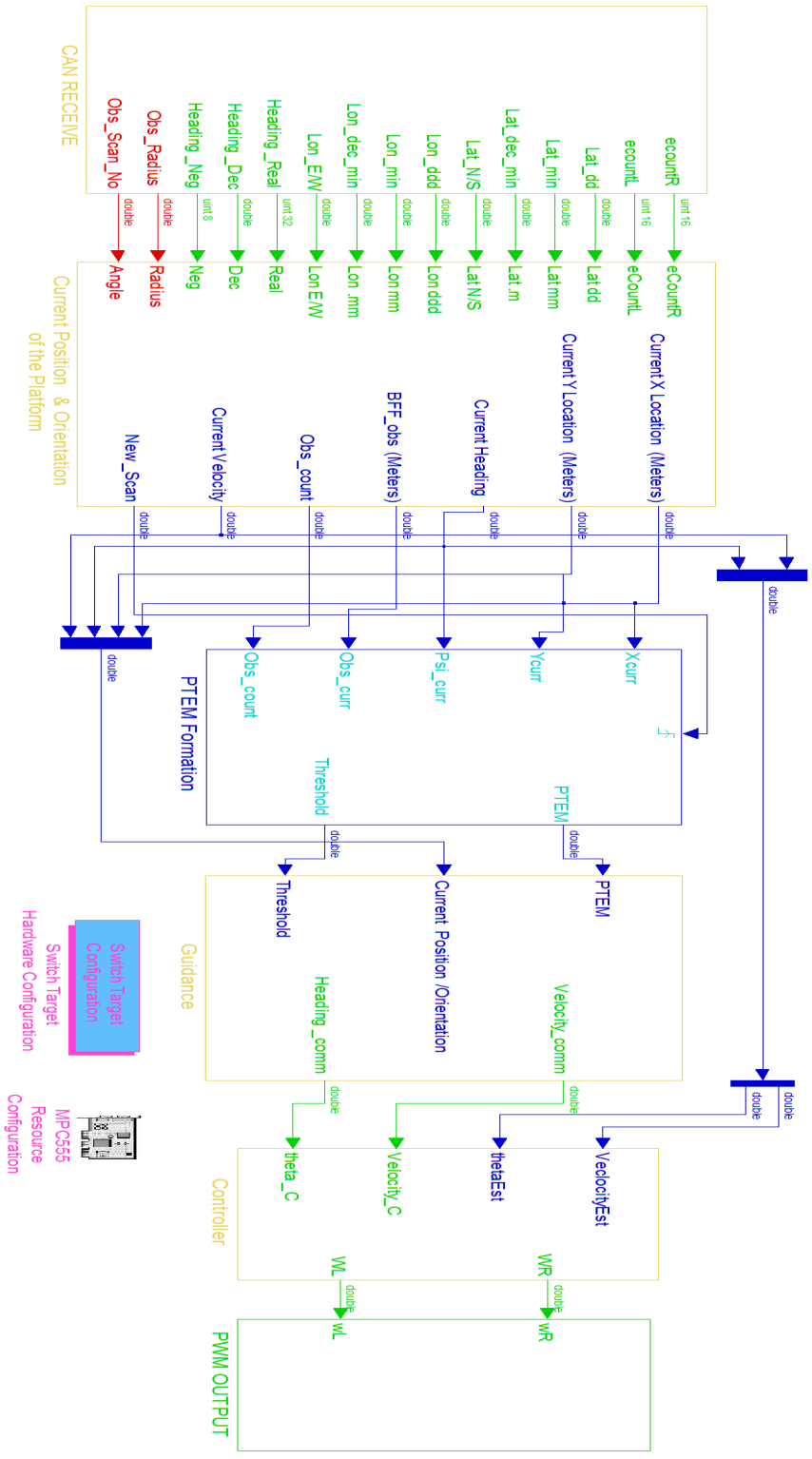

Figure 6.1. Real-time Navigation and Control Simulink Model.

to the module 'Current Position and Orientation of the Platform'. The basic function of this module is to convert the CAN messages and encoder data to information like position, orientation, velocity and obstacle array. This information is then used to localize the tank platform to construct PTEM and to calculate guidance control commands. The 'PTEM formation' module is used to construct PTEM as described in Chapter 5. It outputs the constructed PTEM every time a new obstacle array is detected. The position, orientation data and the constructed PTEM is provided to the 'Guidance' module. It calculates the commanded speed and commanded heading based on the inputs, waypoints and PTEM values. The commanded values are then passed onto the 'Controller' module. The controller block, as the name suggest, controls how the commanded values are achieved. The outputs of the controller block are the left and right wheel angular speeds, which are given to 'PWM output' module. This block is responsible for converting the wheel velocities into actual PWM signals and outputting the PWM values to the MPC555 pins.

This was a brief overall description of the real-time navigation and control model. The sections below describe the responsibilities and functions of each individual module in detail.

6.2.1 CAN Receive Module

The primary function of the CAN Receive module is to receive CAN messages over the CAN network in the right order, strip the data out of the messages and pass useful data over to the next module for processing. It also reads the encoder values from the encoder pins.

Figure 6.2 shows the internal block diagram of the CAN receive module. As shown, one CAN receive message block is required to receive a unique CAN message as each CAN message is identified by its unique CAN identifier. The CAN receive block

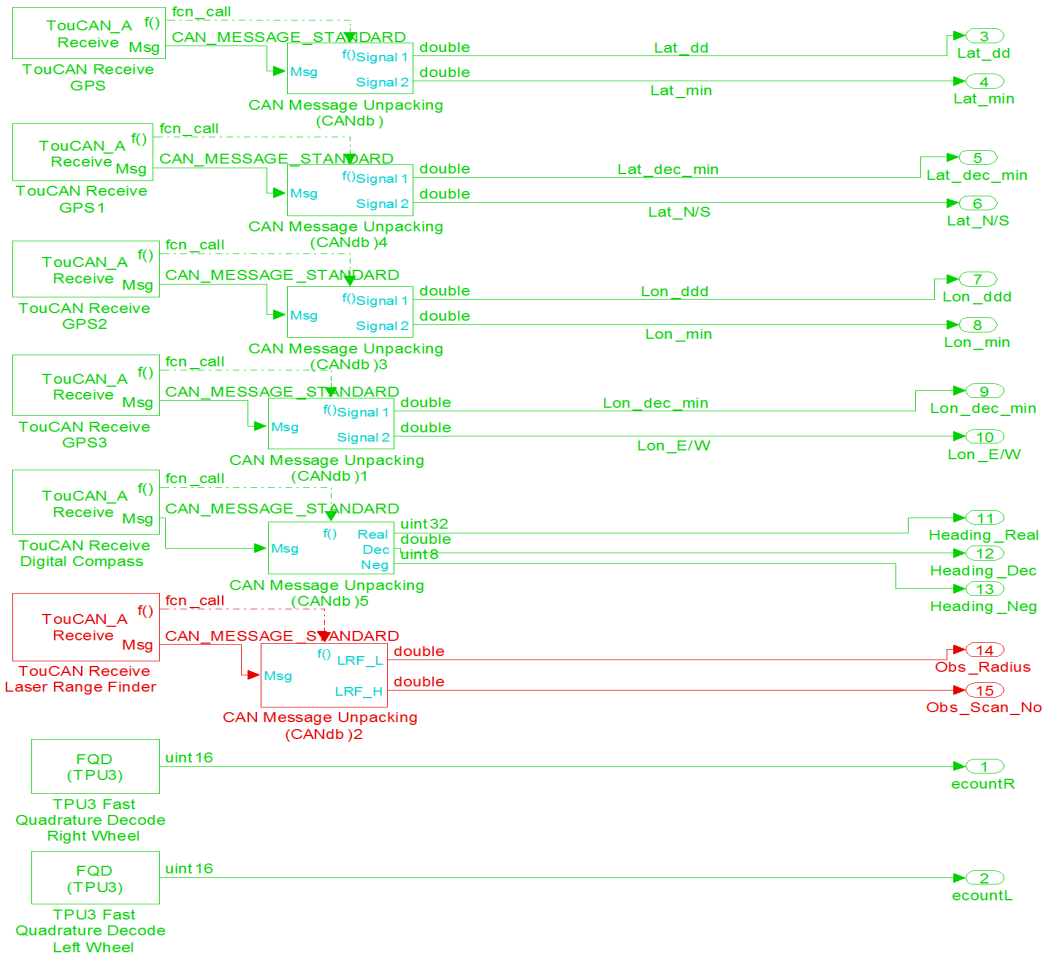

Figure 6.2. CAN Receive Module.

consists of information like the port module name (i.e. A or B), message identifier, CAN message type (i.e. 11 bit or 29 bit) and sample time. The output of the CAN receive block is passed to the CAN unpacking block. This block strips the CAN message and passes on the parsed data for further processing. This block can be a standard data parsing block or a custom data parsing block. A custom data parsing block is used to receive CAN messages from digital compass to obtain maximum efficiency during the data transfers. Parsing of the custom data type is done using an embedded Matlab 6.4 block. The IsoPod^{TM} cannot transmit float messages over

the CAN network. The heading data from the digital compass is in the form of float values, hence the need to custom define a data type to carry float characters. This block provides with output signals containing the parsed data signals. Standard CAN message unpacking blocks are employed to receive GPS and LRF messages.

Since the CAN receive block receives sensor input values, it plays a vital role. So, if we have any delay or we skip CAN messages, we will not be able to process new sensor data. The sample time of the CAN message block determines the rate at which data will be received. The CAN module on the MPC555 has a 16 message CAN buffer. Each CAN receive block can only store one message in the buffer. GPS and digital compass CAN receive blocks employ this method of buffering single CAN message values. Since we only require the most recent value, so the buffer can be allowed to be over written by a new message. However, the LRF requires multiple CAN messages to send one LRF scan data. Also, if the LRF scan initialization message is missed or skipped, erroneous obstacles are mapped into NF. A software buffer along with a CAN receive interrupt routine is used to receive messages from LRF. Depending on the size of the buffer and ISR [1], this ensures that all or most of the CAN messages over the network are received.

The rotary encoder data is read through the TPU module of MPC555. To read one rotary encoder value both channels have to be read. This is done using the TPU module of MPC555.

6.2.2 Position and Orientation Module

This module is responsible for receiving the data from CAN receive module and transforming that data into current position of the platform (current X and Y

[1] Interrupt Service Routine

position), current orientation (heading), current velocity and also transform the laser range finder data into obstacle array in BFF.

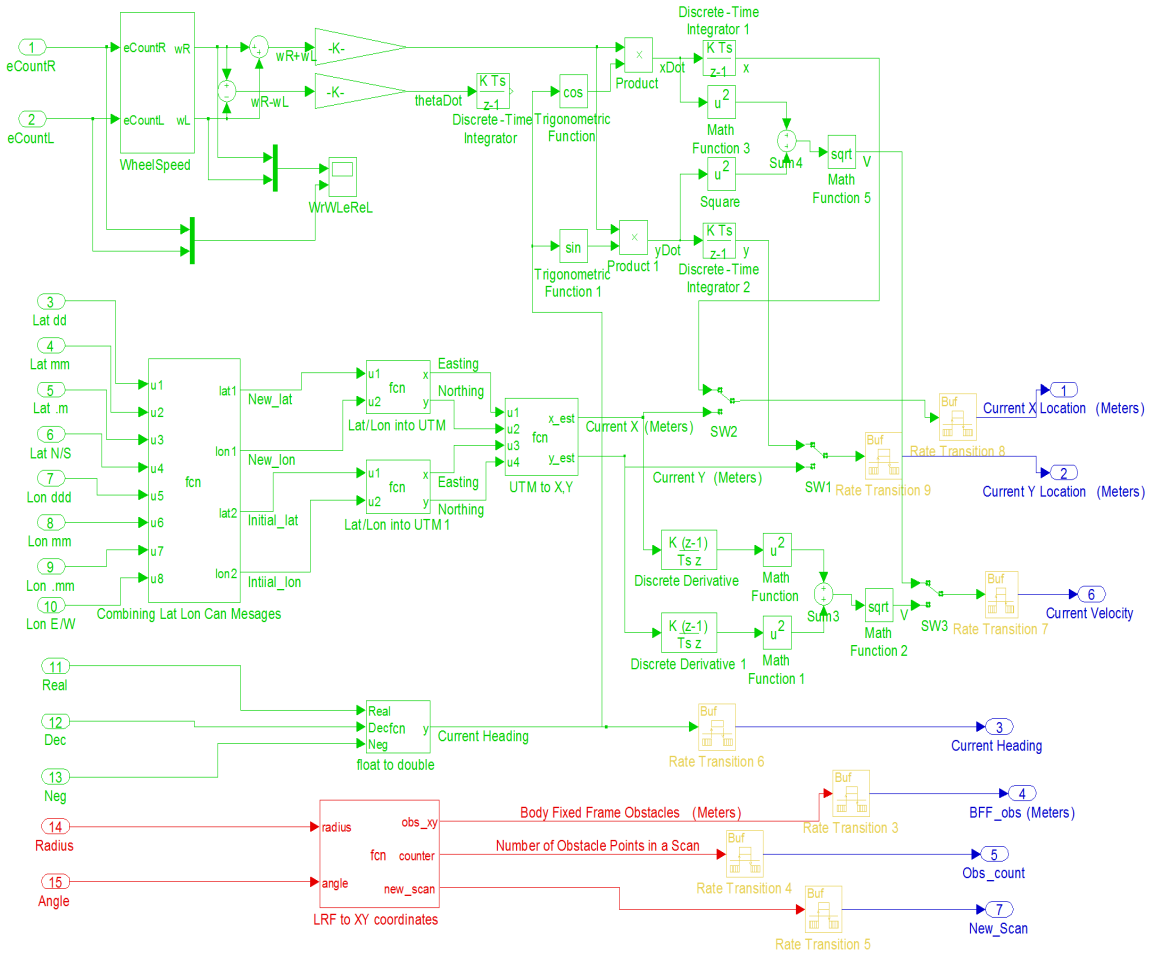

Figure 6.3. Position and Orientation Module Internal Diagram.

As shown in Figure 6.3, the different CAN messages are received through the input ports shown on the extreme left hand side of the block diagram. The GPS messages are combined from different messages into actual latitude and longitude values. This sub-module is implemented using embedded matlab function block. This block has been employed multiple times during this project. More information about

it will be provided later on in this section. This latitude and longitude values are then converted into UTM [2] co-ordinates by using the algorithm developed by Alexandre Schimel [52]. The algorithm was obtained in the form of matlab script from matlab user community under BSD free license agreement. The formulae used to convert GPS readings in to UTM are developed and explained in [53]. The latitude and longitude values are converted from degrees format (ddmm.mm)[3] into radians. WGS84[4] world model is used which models world as an ellipsoid. The distance between the current point and polar axis, central meridian is calculated. Based on this distances and WGS84 model the UTM coordinates are calculated based on the formulae given in [53]. The UTM is a grid-based method of specifying locations on the surface of the earth. The entire earth is divided in to 60 zones and each zone is represented in two dimensional space (Northing (X) and Easting (Y)). Current latitude and longitude values are converted into UTM and subtracted from the initial UTM values based on the starting location of the platform. This provides us the local X and Y values in meters.

Since we use our custom format to transmit heading over the CAN network, the heading messages are converted into heading by the block called 'float to double' shown in Fig. 6.3. The laser range finder messages consist of angle and radius of each obstacle point detected, which are converted into BFF [XY] array. To perform this function the blocks shown in the Fig. 6.4 is employed.

Figures 6.4 and 6.5 refers to the Embedded Matlab Function block and the internal script of the block respectively. Embedded matlab allows us to include Matlab

[2]Universal Transverse Mercator

[3]dd- degrees, mm- minutes and .mm- decimal minutes

[4]World Geodetic System 1984 model

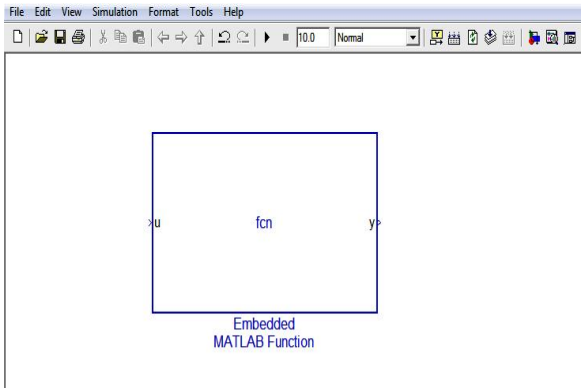

Figure 6.4. Embedded Matlab Function Block.

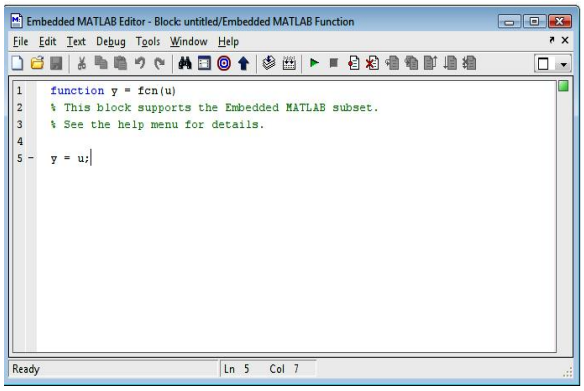

Figure 6.5. Embedded Matlab Block Internal Script.

script in a simulink model that can be compiled by Real-Time Workshop. Embedded Matlab uses a subset of matlab functions; not all functions are supported.

The current positions (X and Y) are also calculated using data from rotary encoders. These calculations are done using a process called *dead reckoning*. Dead reckoning is a process of estimating the current position of the platform/vehicle using previously determined position and advancing position based upon estimated or measured speed over fixed time intervals. The main disadvantage is that a small error that keeps on accumulating may become significant as time progresses.

6.2.3 PTEM Formation Module

The primary function of this module is to convert the BFF array of obstacles into PTEM. Figure 6.6 shows the PTEM module. Here, the inputs are array of

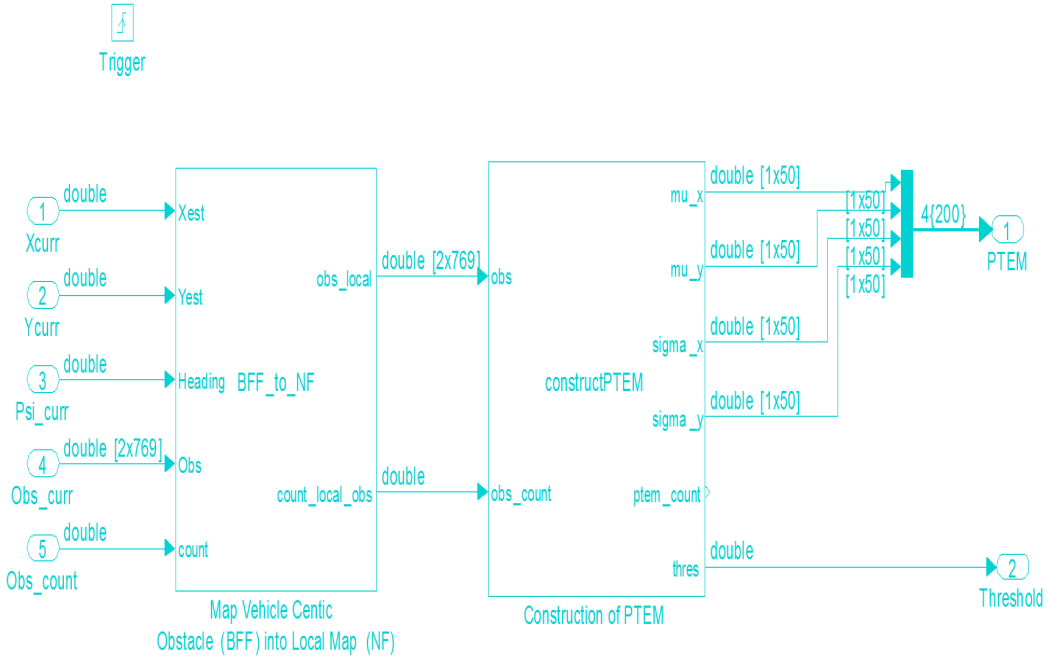

Figure 6.6. PTEM Module.

obstacles from LRF, current position and orientation. BFF obstacle array is first converted into navigational frame (NF) obstacle array by the equations described in Chapter 5. This array of obstacles is given to the block that constructs the PTEM. The obstacle points are clustered into different clusters or combined into existing clusters. Once clustered, an array of mean and variance is calculated from these clusters. This arrays represent the PTEM, which is then passed on as an output array to different modules.

6.2.4 Guidance and Waypoint Navigation Module

This module calculates the heading and speed commands to steer the platform away from obstacles and towards the next waypoint successfully. The algorithms used for guidance are adopted from Zengin's dissertation [2]. The guidance algorithm used was obtained in the form of Matlab script and S- functions from Zengin's earlier work. Since we have developed our model in simulink, the algorithms were divided into different embedded matlab blocks.

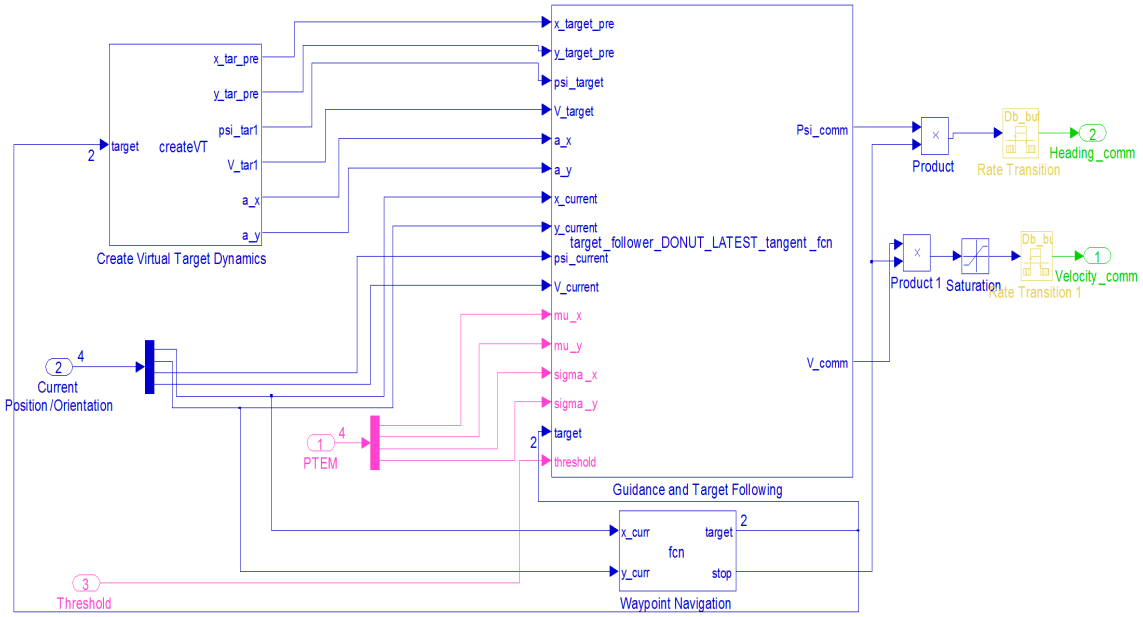

Figure 6.7. Guidance and Waypoint Navigation Module.

The current waypoint to be pursued is given by the Waypoint Navigation block. Depending upon the current position of the platform the 'Waypoint Navigation' block will output the current waypoint for the vehicle to pursue. This set of waypoints is not directly pursued by the platform/vehicle. Instead, it is given to the virtual target which pursues the waypoint. Once the platform has reached all the waypoints and

is within the proximity distance of the last waypoint, the commanded velocity is outputted as zero and the vehicle comes to a stop. This function is also performed by the waypoint navigation block by outputting a stop flag.

The guidance algorithm developed by Zengin is to pursue or track moving targets/vehicle. In our case we do not have a moving target to pursue/track. So, a virtual target is created by the block 'Virtual Target Dynamics' . This block creates the dynamics of a virtual target. This virtual target dynamics block along with waypoint navigation block ensures that the virtual target navigates through all the waypoints. The guidance algorithm pursues and tracks this virtual target. The virtual target dynamics block takes current waypoint and initial position as the inputs and generates dynamics of the virtual target as shown in Fig. 6.7.

The 'Target Follower Function' block is responsible for tracking the virtual target and avoiding all the obstacles in the way. While pursuing the virtual target the tank navigates through all the waypoints. The guidance algorithm, as implemented in the target follower function, ensures that we avoid all the obstacles while staying inside a fixed proximity of the virtual target. The PTEM generated by the earlier module is used by this block. The outputs of this block are commanded heading and velocity. These outputs are passed on to the 'Controller' module.

6.2.5 Tank Controller Module

This module is responsible for controlling how the platform achieves the commanded velocity and orientation. As shown in Fig. 6.8, the tank controller is made of two proportional controllers: one for velocity and the other one for heading. This module takes inputs as commanded speed, commanded velocity, current speed and current velocity. The outputs of this module are left and right wheel speed. The gains of the controller are chosen such that we do not have any excessive transients

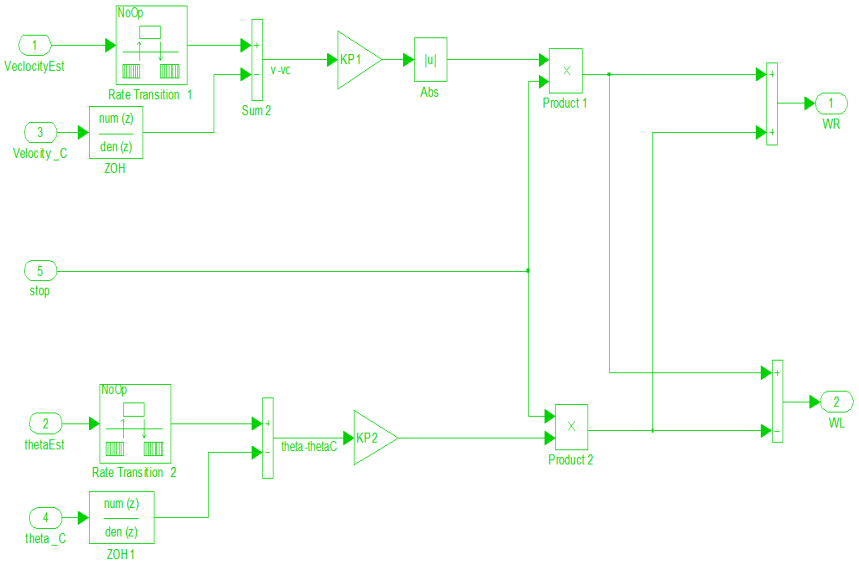

Figure 6.8. Tank Controller Module.

while obtaining fast responses in speed or orientation of the platform. This ensures that we do not violate the physical constraints of the platform.

6.2.6 PWM Output Module

The PWM output module is responsible for converting the right and left wheel speed commands into 4 PWM channel outputs.

Figure 6.9 shows the PWM output module. The two inputs are transformed into 4 outputs by a lookup table. This lookup table is developed considering the physical characteristics of the motor and the H-bridge driver. Since the tank platform is differential drive platform, we require four PWM outputs, two for each wheel/track. For one wheel/track we require one PWM output to move forward and other to move it in the reverse direction. So, we have four outputs representing right forward, right reverse, left forward and left reverse. These four outputs are given to four physical

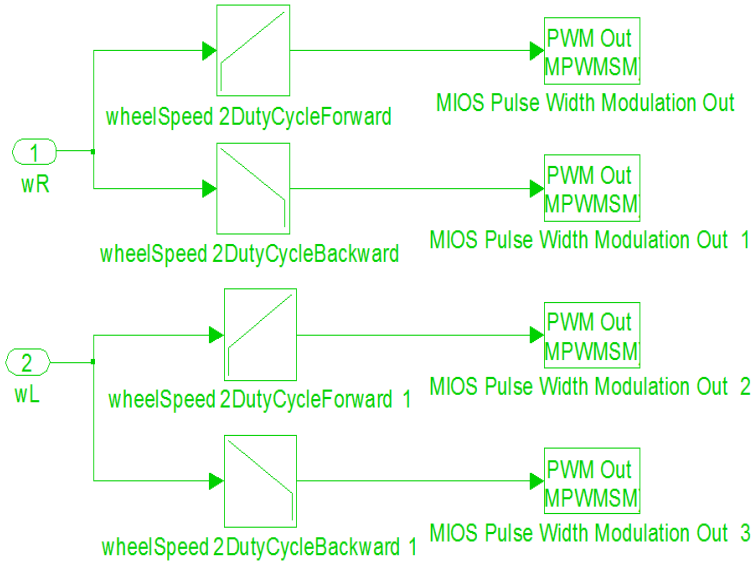

Figure 6.9. PWM Output Module.

pins represented by four MPWMSM blocks. In this block, the physical pin number where the signals will be outputted on the MPC555 is specified.

6.2.7 Processor Selection Blocks

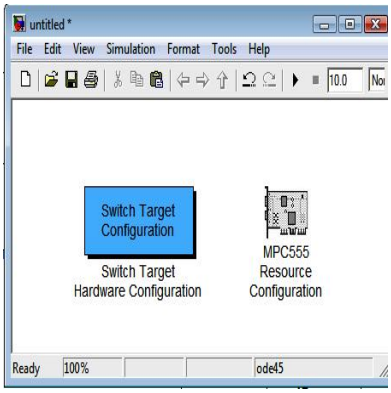

Figure 6.10. MPC555 Processor Selection and Target Configuration Selection Blocks.

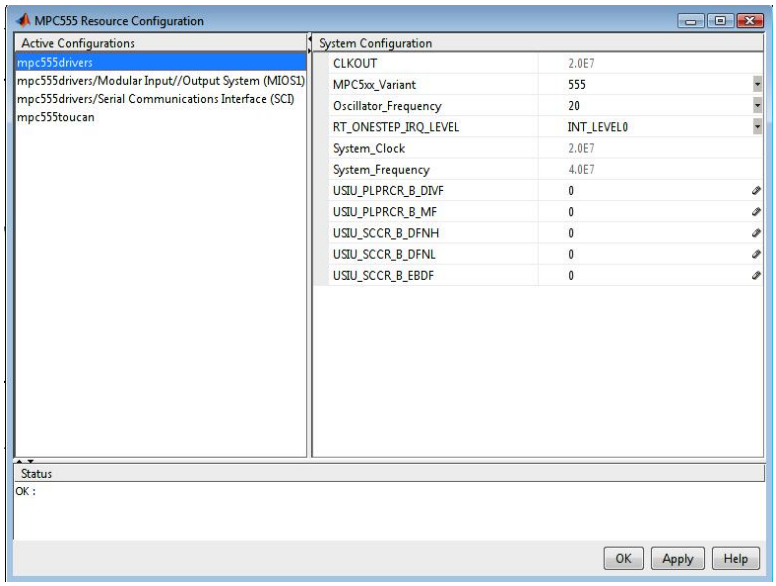

Figure 6.11. MPC555 Resource Selection block.

The Matlab-Simulink and real-time workshop toolchain supports many real-time processors. In order to select the MPC555 processor, two processor specific blocks are added to the model.

Figure 6.10 shows the 'Hardware Configuration' and 'MPC555 Resource Configuration' blocks. These blocks are important and customized for each processor. They specify the physical drivers and configuration of the target processor (MPC555). Figure 6.11 shows how different MPC555 resources can be custom tuned to match the design needs such as speed of the processor, serial ports, CAN modules, and various other configuration. MPC555 resource configuration block is used to configure the baud rate of serial port, the bit rate of the CAN network, interrupt priorities for the execution of the main function call and configuration of digital I/O ports.

6.3 Simulation of Navigation and Control Model

The simulation model developed uses the exact same guidance algorithm as used in the real-time model. Utilizing the simulation model and simulation environment, different tests are performed to test the guidance and navigation algorithms. The platform is modeled using the kinematic equations of the tank. The position and orientation of the platform is obtained by modeling rotary encoders. The inputs to the real-time model are obtained from encoders, digital compass, GPS, and LRF. The inputs to the simulation model are only obtained by modeling rotary encoders. The PTEM used in simulation model is pre-formed (i.e. it is not constructed dynamically). These are the major differences between the real-time model and simulation model.

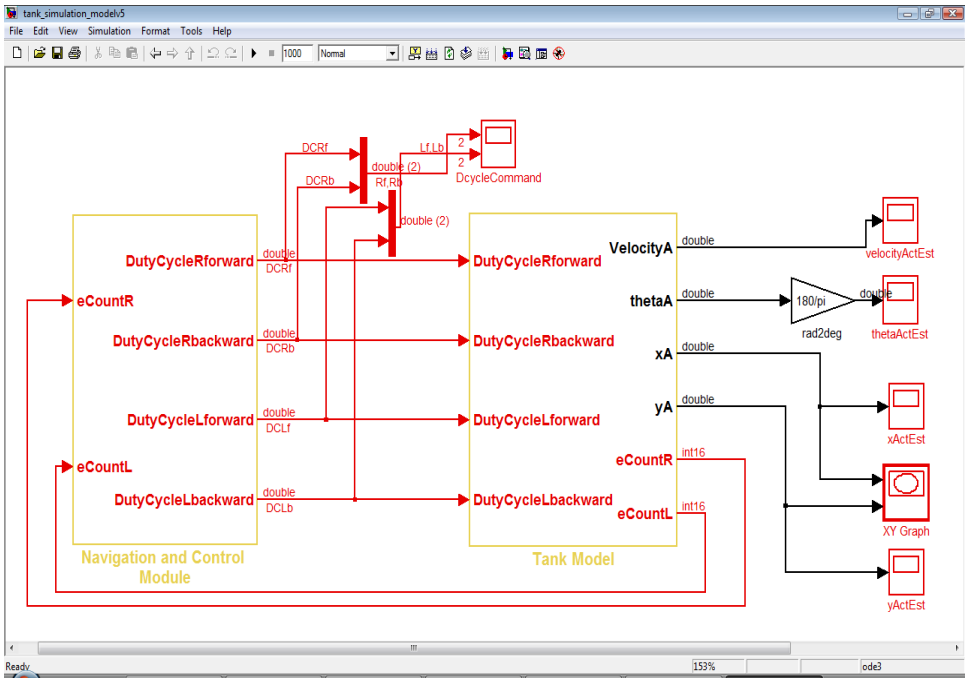

Figure 6.12. Simulation Navigation and Control Model.

Figure 6.12 shows the simulation model developed during this research project. The tank model in Figure 6.12 represents the model of the platform or vehicle. The

navigation and control block represents the real-time navigation and control simulink model.

6.3.1 Tank Model

The tank model creates a model of the physical platform by using the kinematic equations that guide the working of the platform. The tank model used in this research project is adopted from Dr. Dogan's previous research work.

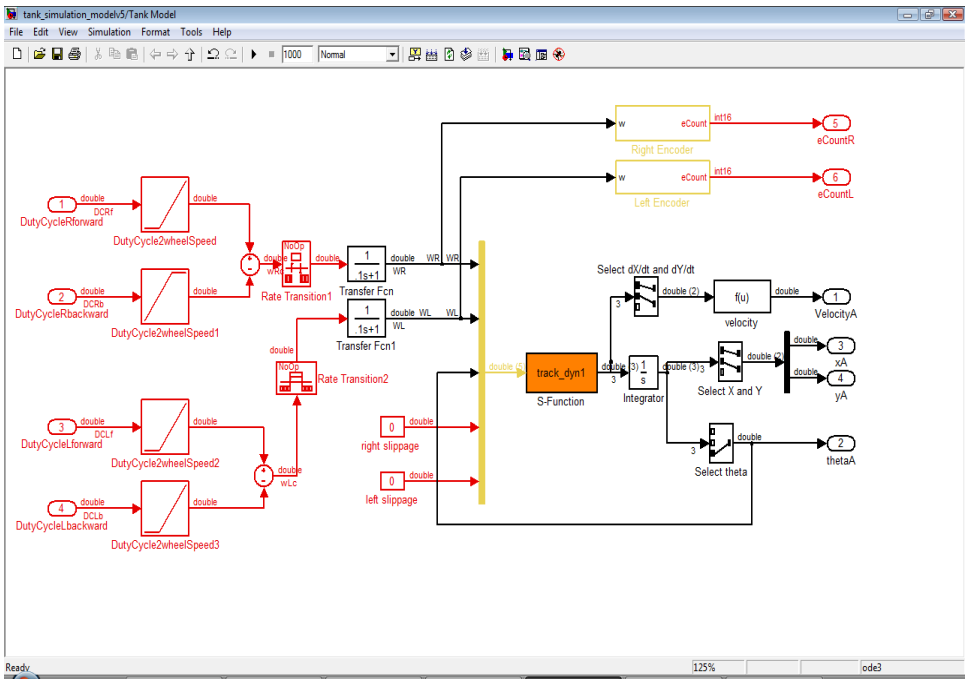

Figure 6.13. Tank Model.

Figure 6.13 shows the model of the tank developed in this research project. This model is designed to represent the actual platform. The inputs to the model are the PWM signals and the output are the encoder counts.

The input PWM signals are in the range from 0 to 1, where, 0 represents 0 percent and 1 represent 100 percent duty cycle. The PWM signals are converted into

angular velocities by a simple look up table. The minimum PWM value required to generate enough torque to move the platform is 30 percent or 0.3 PWM value. The data points of the look-up tables for right and left forward channels are as given in Table 6.1 where Vmax is the maximum velocity of the platform and rNominal is the radius of the wheel.

Range of Input PWM	0	0.3	1
Output Angular Velocity	0	Vmax/rNominal/100	Vmax/rNominal

Table 6.1. Forward PWM to Angular Velocity Lookup Table

Similarly, Table 6.2 gives a look-up table data points for the reverse channels. Linear interpolation is used for any other data point within the data range.

Range of Input PWM	1	0.3	0
Output Angular Velocity	-Vmax/rNominal	-Vmax/rNominal/100	0

Table 6.2. Forward PWM to Angular Velocity Lookup Table

By adding the forward and reverse channel angular velocities, we get Wr (right wheel angular velocity) and Wl (left wheel angular velocity). The left and right angular velocities are converted to right encoder ticks (eCountR) and left encoder ticks (eCountL) as shown below.

$eCountx = rNominal \times eTick \times \int Wx * dt$

where, eTick is the number of pulses or ticks per meter.

The equations that govern the kinematics of the tank calculates the translational and angular velocity based on the angular velocities Wl, Wr, orientation(θ), slippage constants (s_r) and (s_l).

$$\dot{X} = [((1 - s_r) \times rr \times Wr + (1 - s_l) \times rl \times Wl)] \frac{\cos(\theta)}{2} \quad (6.1)$$

$$\dot{Y} = [((1 - s_r) \times rr \times Wr + (1 - s_l) \times rl \times Wl)] \frac{\sin(\theta)}{2} \quad (6.2)$$

$$\dot{\theta} = [((1 - s_r) \times rr \times Wr - (1 - s_l) \times rl \times Wl)] \frac{1}{b} \quad (6.3)$$

where b is the diagonal length of the platform.

By integrating $\dot{X}$, $\dot{Y}$, and $\dot{\theta}$, we get X,Y and θ. The encoder counts eCountR and eCountL are fedback to the navigation and guidance module.

6.3.2 Navigation and Control Model

The navigation and control module used in simulation environment utilizes the exact same guidance algorithms as used in real-time model. The major difference between the simulation navigation model and the real-time navigation model is the nature of inputs. In the real-time model we have inputs from physical sensors, rotary encoders, GPS, LRF, and digital compass. In simulation model, the only sensor to be modeled are rotary encoders.

Figure 6.14 show the navigation and control model developed for simulation. The guidance and waypoint navigation module, controller module and PWM output module is exactly identical to the modules used in real-time navigation and control model. The PTEM construction module is absent in the simulation model since, the LRF has not been modeled. PTEM, however is entered as a fixed value in the target follower function. The input sensor modeled is the rotary encoder. The current position, orientation and velocity of the platform is obtained from position and orientation module using dead reckoning. This process is explained in detail in Section 6.2.2.

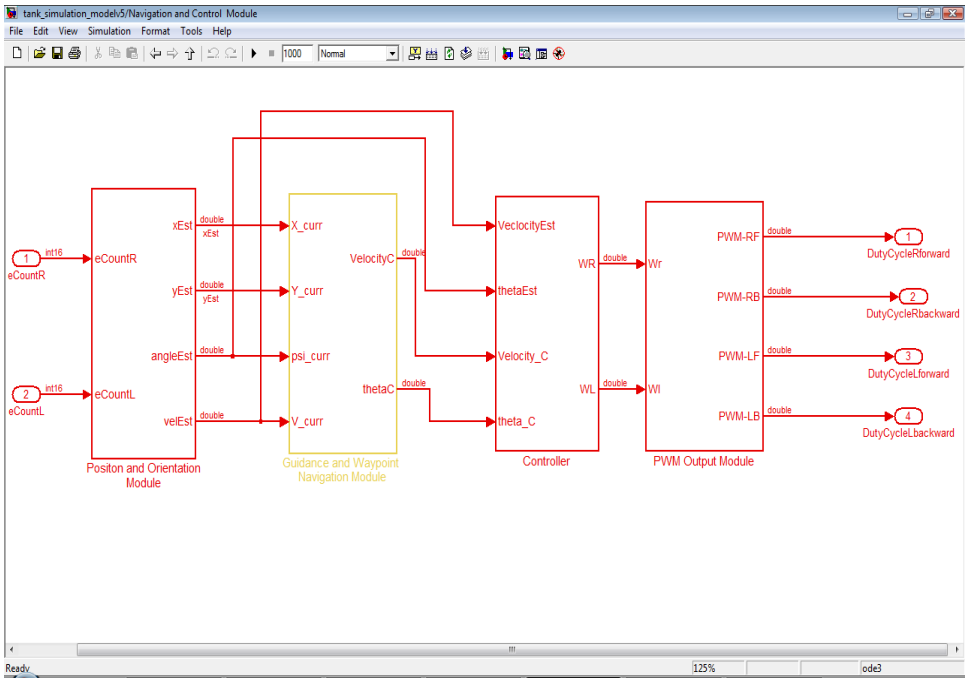

Figure 6.14. Simulation Navigation and Control Module.

The current position, orientation and velocity are provided as input arguments to the guidance and waypoint navigation control module. The value of PTEM is fixed as mentioned. The guidance and waypoint navigation control module generates commanded heading and commanded velocity control signals in the exact same way as described in Section 6.2.4. The control signals are converted in to PWM values using the same process to the one used in real-time navigation and control module. These PWM values are then applied to the tank model, which again converts the PWM in to encoders ticks and feeds it back to the simulation navigation and control module as described earlier.

CHAPTER 7

EXPERIMENTS

7.1 Introduction

This chapter provides a detailed description of the experiments conducted during the research project. The data collected from the experiments is analyzed to examine the system performance in the real world. It assists in providing system specifications and operating conditions. All the data is collected in real-time. The real-time data is analyzed against the simulation data from simulation experiments. This comparison abets in identification of difference in performance in the simulation environment and real-time environment. These experiments assist in the identification of various constraints for the system and algorithms developed. Sensor calibration experiments aid in deciding sensors and their operating orientation, to achieve the required accuracy for the algorithms being developed.

7.2 Sensor Calibration Experiments

This section discusses the experiments conducted for sensor calibration. The results of these experiments decide which sensors to use and in what configuration they must be configured, to provide effective estimates/readings.

7.2.1 Orientation Sensor Calibration And Accuracy Assessment

The digital compass is used as the orientation detection sensor, it provides heading from magnetic north. The navigation frame (NF) is referenced to true north. Magnetic north is converted into true north by adding a fixed deviation of 4.4 degrees

E (for Arlington Texas) to the compass heading data. The accuracy of heading sensor is very essential in the calculation of orientation of the platform.

7.2.1.1 Experiment 1 Setup And Results

The objective of this experiment is to test the accuracy and consistency of the data received from digital compass. The digital compass is mounted in a fixed position on the platform and the platform is moved at constant speeds in a straight line. The data from the digital compass is collected during this experiment. This data is analyzed by the plotting heading vs time (compass samples) plots using Matlab.

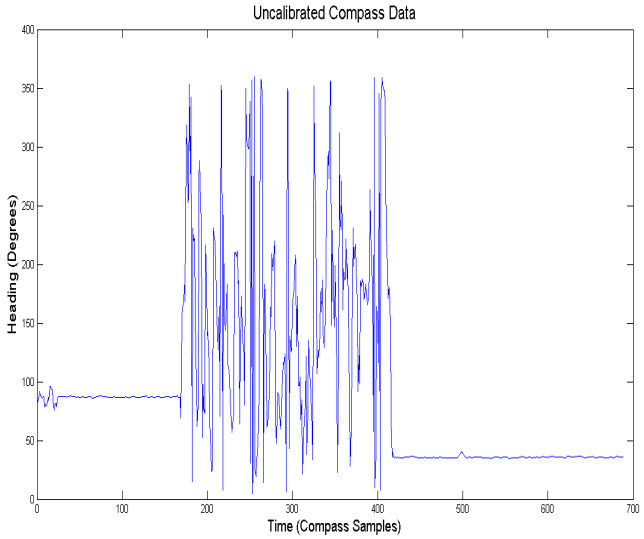

Figure 7.1. Uncalibrated Compass Moving in a Straight Line in East Direction.

Figures 7.1 and 7.2 show the data collected from the digital compass during experiment 1. Both the experiments shown in Fig. 7.1 and 7.2 are performed in similar environmental conditions. In the experiment shown in Fig. 7.1, the vehicle is moved in east direction while, in the experiment shown in Fig. 7.2, the vehicle

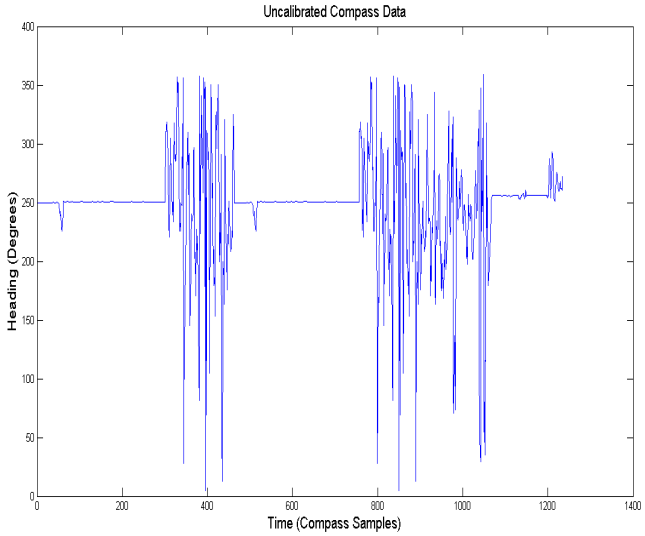

Figure 7.2. Uncalibrated Compass Moving in a Straight Line in West direction.

is moved in west direction. The data obtained from the digital compass contains enormous error and indicates that the platform is not moving in a straight line.

The procedure performed during this experiment is given below:

1. The physical setup of the experiment is put together as described earlier.
2. Power is applied to the system and data logging is began.
3. The platform is set into motion in a straight line.
4. The platform is brought to a halt when the required distance is covered.
5. The data logging is then stopped and power is removed from the system.

7.2.1.2 Experiment 1 Result Analysis

The setup and resultant graphs derived from the data analyzed. The platform is set in to motion from about 160^{th} to 180^{th} compass sample and the motion is stopped from about 410^{th} to 430^{th} compass sample in the first test indicated by Figure 7.1. The compass is sampled at 2 Hz, so the sample time is 0.5 seconds. A

start-stop motion is performed during the second test run as indicated in Figure 7.2. The results in Figures 7.1 and 7.2 indicate that once the platform is mobile there is significant variation/noise in the data coming out of the compass. This variation is attributed to magnetic interference caused by the magnetic field produced by the rotating motors, the current carrying conductors, the magnetic field of the objects in the environment as well as the variation/deviation due to vibration caused by the moving platform.

7.2.1.3 Experiment 1 Conclusion

To attenuate the noise caused by vibration, some shock/vibration absorbing material has to be used. To reduce noise caused by magnetic interference, magnetic shielding material, hard iron and soft iron calibration routines have to be used.

7.2.1.4 Experiment 2 Setup and Results

The aim of Experiment 2 is to implement the conclusions of Experiment 1. To decrease the effects of vibration on the digital compass a small foam pad is inserted between the digital compass and the platform. This foam pad absorbs some part of the vibrations when the platform is mobile. Soft iron and hard iron calibration tests are performed to negate the effects of permanent and temporary magnetic fields.

Figures 7.3 and 7.4 show the hard iron and soft iron magnetic tests performed on the digital compass through the interface software provided by the digital compass manufacturer. Hard iron errors are usually most severe and have the effect of shifting the center of a sphere from the origin within a 3 dimensional coordinate system. The purpose of a hard-iron calibration procedure is to determine where the center of the sphere lies so that appropriate X, Y, and Z offsets can be subtracted/added from/to the corresponding magnetometer signals to return the center to the origin.

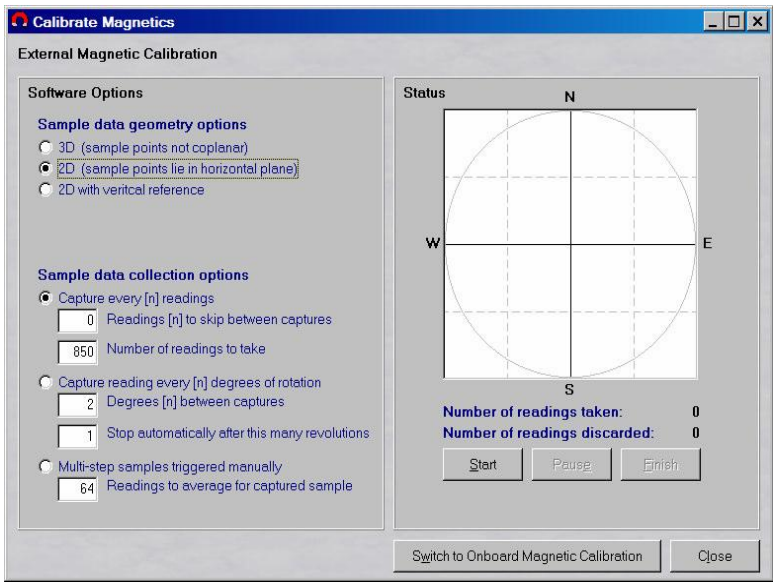

Figure 7.3. Performing 2-D magnetic calibration.

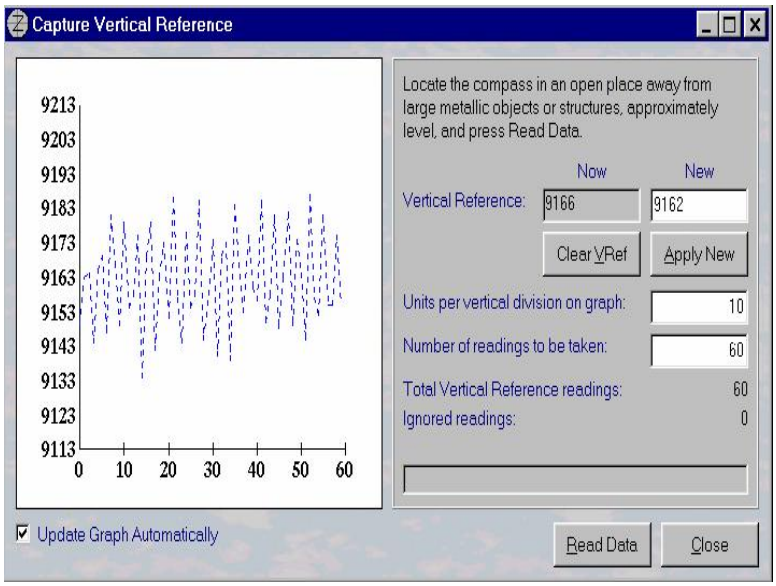

Figure 7.4. Calculating vertical reference for magnetic calibration.

Soft iron effects distort the shape of the sphere into an ellipsoid. The purpose of a soft-iron calibration procedure is to determine parameters of the ellipsoid, such as the relative lengths of the 3 axes and their orientation in space, so that an appropriate transformation can be generated that will reshape the ellipsoid into a sphere.

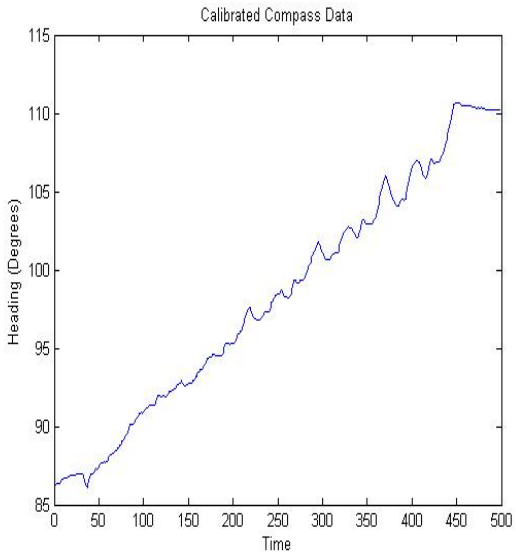

Figure 7.5. Calibrated compass moving in a straight line in East direction.

Figures 7.5 and 7.6 show the results after all the calibration procedures are performed. The calibrations are performed while the compass is mounted on the platform in its final position. The same procedure is followed to take the readings as described earlier. The experiments performed after calibration are similar to the ones performed before calibration. The tests shown in Figure 7.2 and 7.6 are similar. In Figure 7.2 a start-stop approach is used while the calibrated compass in Figure 7.6 is ran continuously till the desired distance is reached. The tests shown in Figures 7.1 and 7.5 are exactly same with same distance and same direction of motion.

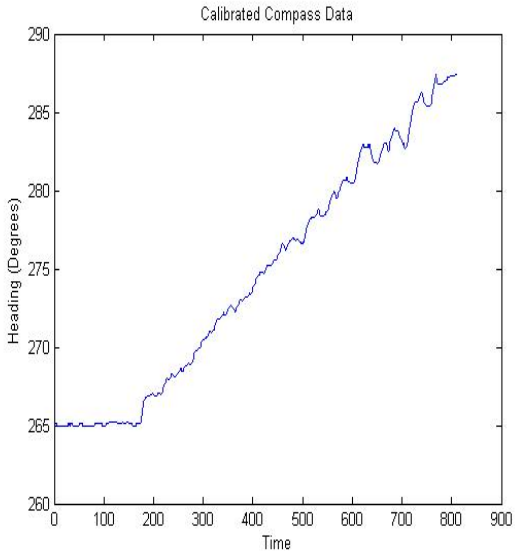

Figure 7.6. Calibrated compass moving in a straight line in West direction.

7.2.1.5 Experiment 2 Result Analysis

As depicted by the graph in Figures 7.5 and 7.6 after performing hard iron and soft iron calibration routines, we get a much better results. The calibration routines provide different results in different environments. Figures 7.5 and 7.6 also indicate that even after compensating for hard iron, soft iron and vibration, we still do not get perfect readings, but they are good enough to make successful decisions. The readings from the compass show that the tank has moved on an arc (constant turning) when in fact the tank was moved in a straight line. The difference in compass readings from the starting of a straight line and the ending is approximately equal to 20 degrees. The motion of the tank is controlled using the IsoPod^{TM} from NMIterm. Due to the differences in gear set and DC motors, the tank does not move in a straight line even when both the motors are given equal voltages. This accounts for about 8 to 12 degrees of error.

7.2.1.6 Experiment 2 Conclusion

Compass calibration routines have to be performed to obtain accurate/reliable data from the digital compass. To negate the effects of magnetic interference; a shielding material called Magnet Shield [54] can be been used to cover the D.C. motors.

7.2.2 Position Sensor Calibration and Accuracy Analysis

The position sensor is used to provide the current (X and Y) position of the platform, while the platform is mobile or static. It is essential for the position sensor to be accurate within 0.6 meters (15 percent of 4 m) of the actual position since the construction of PTEM requires current locations. The maximum range of the scanning laser range finder is 4 meters. In the final part of the transformation of the BBF obstacle array into the NF obstacle array, the current position is added as a bias. If the current position has an error of more than 0.6 meters, then the whole frame is *shifted by the same amount of error* in the position. This shifts the location of the obstacles during the construction of PTEM in NF.

In this research project the position sensors used are the GPS and rotary encoders. The position obtained from the rotary encoders is locally referenced. The GPS data provides the global position of the platform which is converted into a local position and used in the system. All the position sensor calibration experiments will be conducted with GPS.

7.2.2.1 Experiment 1 Setup and Results

The objective of the experiment is to test the accuracy of the position sensor (GPS). The GPS receiver unit is mounted on the platform, the platform is traveling

in a straight line. The distance between the points; the starting point and the final stop point is calculated. Data from the GPS receiver is logged, converted into local co-ordinates and plotted using Matlab. The experiment yields detailed analysis of the results and a brief conclusion.

The experiment is performed in the parking lot, with a clear view of the sky and no buildings in the nearby vicinity that would reflect the GPS signals. The motion control of the tank/platform is done using a IsoPod^{TM} and is controlled from the NMITerm [1] [43] interface. The vehicle carrying the GPS system is driven through the given path by the commands entered in the NMIterm. Data from GPS is logged on to the laptop using serial interface.

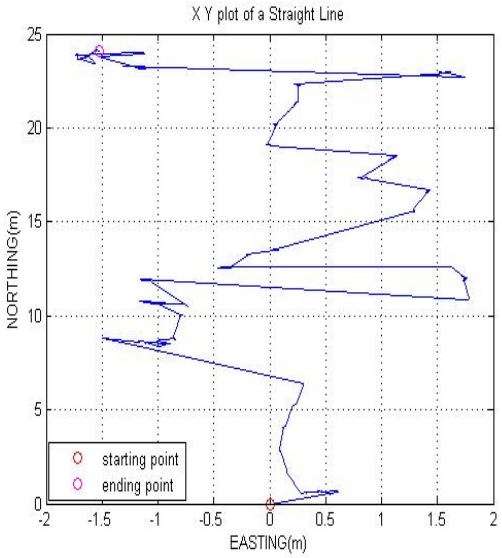

Figure 7.7. Single GPS Receiver Moving in a Straight Line (Towards North).

[1]NMITerm is an interface software developed by New Micros to communicate to IsoPod^{TM}/PlugaPod^{TM} over the serial channel

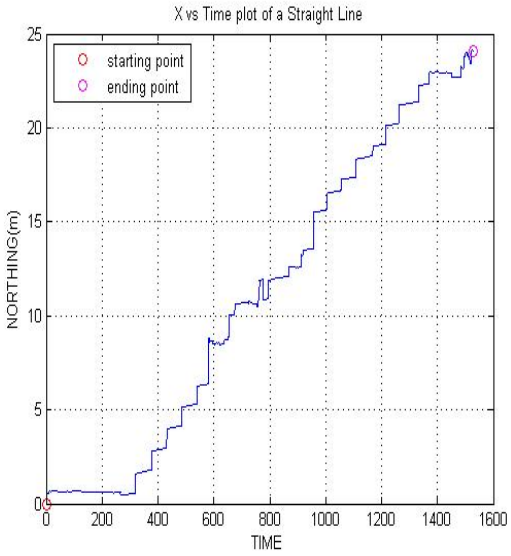

Figure 7.8. North Axis Motion of a Single GPS Receiver.

Figure 7.7 inidactes the path followed by the actual platform in navigational frame (NF) according to GPS receiver readings. Figure 7.8 shows the graph Northing (X) versus Time. This plot represents the path taken by the GPS readings when the platform is moved in a straight line in North.

7.2.2.2 Experiment 1 Result Analysis

Figure 7.7 shows the graph of Easting vs Northing, as shown in the graph the variation is +/- 2 meters in East/West direction. There are multiple sources of error in GPS readings. Ephemeris data variation, clock drift of the GPS receiver, measurement noise and multipath fading are some sources of GPS error [55]. Errors such as clock drift of the receiver, multipath fading, ephemeris and almanac data are time and phase dependent on the readings from the satellites. As a result we can expect to get various fluctuations in readings even if we perform the same experiments at the same location with similar environment conditions. As shown in Figures 7.7

and 7.8, there is not a lot of variation in North/South direction, but there is a large error present in East/West direction. Also, when we calculate the distance between the starting point and ending point and compare it to the actual distance measured, we get an error of about 20 to 40 percent. This error varies with distance and direction of travel. This amount of error is large and unacceptable for the implementation of our research project.

7.2.2.3 Experiment 1 Conclusion

Using a single GPS receiver, we get an accuracy of +/- 2 meters. Since, our obstacle avoidance sensor can only sense up to 4 meters, we cannot afford this much variation during the formation of the local map. This amount of variation has the potential to shift an obstacle 2 or 3 meters away from its original position. The GPS, in the current implementation and system setup, provides inaccurate readings which is not acceptable for this research project implementation.

7.2.2.4 Experiment 2 Setup and Results

The results from position calibration Experiment 1 indicate a more accurate method of determining the current position of the vehicle. The setup for Experiment 2 consists of a differential GPS (DGPS) station with Real Time Kinematics. The DGPS system consist of a fixed base station and a mobile rover station. The base station consists of a GPS receiver, antenna and a serial modem transmitter to communicate with the rover station. The rover station consists of similar equipment mounted on the platform itself.

Experiment 2 is performed in the same parking lot with similar environmental conditions. The rover station is moving in a straight line over the desired distance.

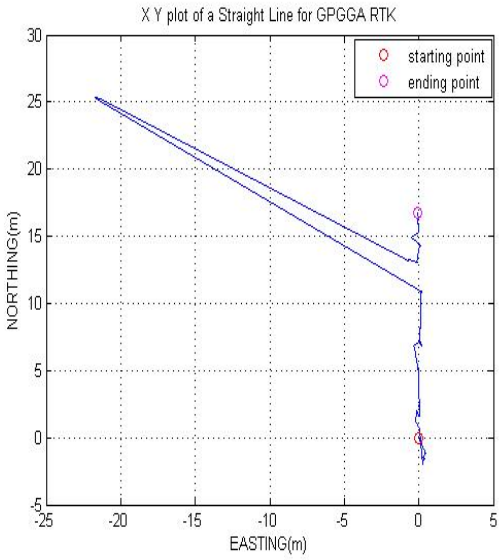

Figure 7.9. Differential GPS Receiver Moving in a Straight Line (Towards North).

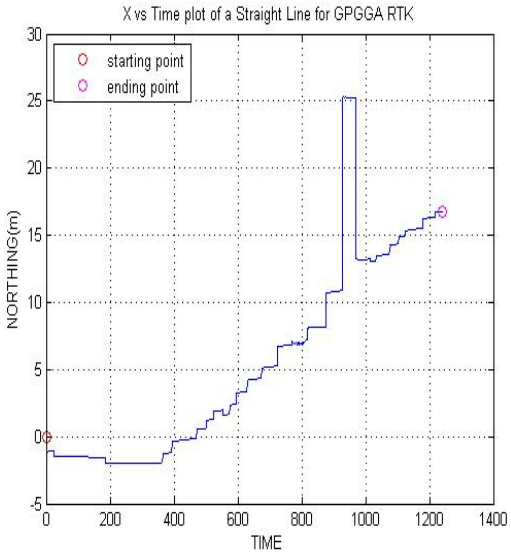

Figure 7.10. North Axis Motion of a Differential GPS Receiver.

Figure 7.9 shows the local map graph and the path followed by the actual platform according to DGPS receiver readings. While Figure 7.8 shows the graph X (North) versus time, it also shows that the platform is moving in a straight line in North direction.

7.2.2.5 Experiment 2 Result Analysis

As seen from the above Figure 7.9 the variations have reduced from +/- 2 meters to about +/- 0.3 meters by using Differential GPS with Real Time Kinematics. As shown in Figure 7.10 the platform is moving at constant speed in North direction. The distance traveled is calculated from differential GPS readings. The comparison of calculated distance against the actual distance provides an error of +/- 10 percent.

7.2.2.6 Experiment 2 Conclusion

From experiment 2 we can conclude that differential GPS with real time kinematics provide better accuracy than single GPS receiver. The error in position estimation is +/- 0.3 meters. This amount of error is acceptable for the implementation of this research project.

7.2.3 Obstacle Detection Sensor

The obstacle detection sensor is used to detect obstacles. The experiments are conducted to test the maximum sense radius and accuracy of obstacles detected. The obstacle detection sensor used during the research project is a Hokuyo URG-04LX laser range finder. The Hokuyo has a range of 20 millimeters to 4000 millimeters and an accuracy of 1 centimeter. It also has a scan radius of 270 degrees and a sense radius of 240 degrees. An experiment is conducted to test these claims and the results obtained are analyzed.

7.2.3.1 Experiment 1 Setup and Description

The setup of this experiments consists of the Hokuyo laser range finder and a cylindrical obstacle. The obstacle is static and circular in nature. The laser range finder is mounted on the platform and the data is logged. The logged data is converted into X Y co-ordinates in NF by the same algorithm that is employed in this research project. The X Y values of the obstacle are then compared to the actual co-ordinates of the obstacle physically measured. An array of obstacle points [x,y] in the NF is constructed and is plotted using Matlab to provide a visual indication of the obstacle/obstacles.

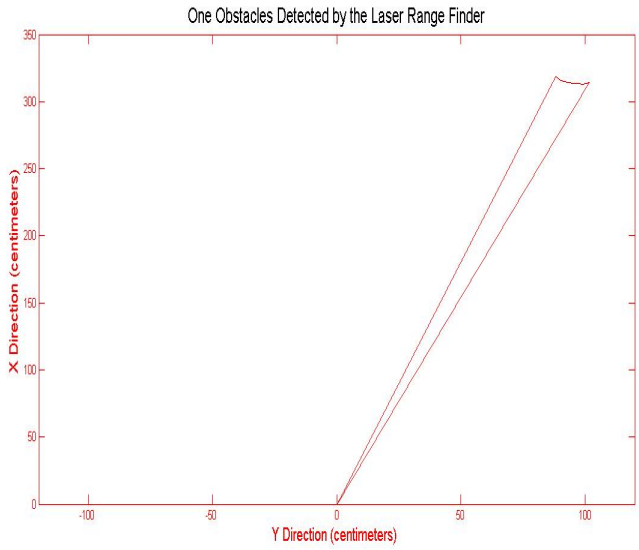

Figure 7.11. One Obstacle Detected by the Laser Range Finder.

Figures 7.11 and 7.12 shows one and two obstacles detected by the laser range finder. The obstacles used for this experiment were small cones.

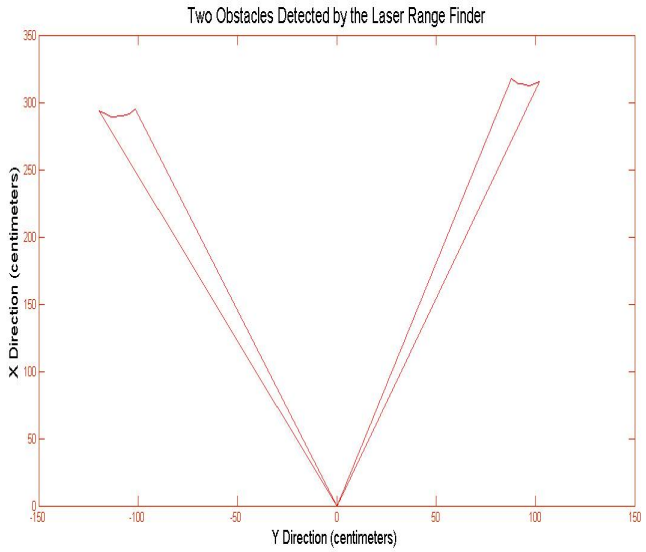

Figure 7.12. Two Obstacles Detected by the Laser Range Finder.

7.2.3.2 Experiment 1 Result Analysis

As shown in Figures 7.11 and 7.12 the obstacles when plotted using Matlab form a semi-circular shape. The [X,Y] co-ordinate of the obstacle in Figure 7.11 is [318.520, 101.6827]. The actual measured value of the obstacle is [323 99]. The [X,Y] co-ordinates of the obstacle on the right in the Figure 7.12 are [312.9195, 94.9231] against the actual measured values of [310 97]. For the obstacle on the left the results are [289.91 -107.7659] against the actual values of [289 104].

From the above discussion, it is clear that we have less than +/- 5 percent error from the data coming from the LRF.

7.2.3.3 Experiment 1 Conclusion

Obstacles are detected accurately by the LRF. Accurate and reliable data is obtained from the LRF in its current orientation. The LRF in its current orientation without any modifications is employed in this research project.

7.2.3.4 Experiment 2

The primary objective of this experiment is to test the maximum and minimum sense range of the LRF. The setup of this experiment again consists of the LRF mounted on the platform, cylindrical obstacles and a laptop for data logging.

In this experiment the LRF is turned on and the obstacles were moved in and out of the boundary radius of 4 meters (the maximum sense range of the LRF). Results indicate that maximum sense range is 3.82 meters. The maximum sense range is not constant. But, the average maximum sense range is 3.82 meters. The same experiment is repeated to find the minimum sense range. Here the obstacles are moved very close to the LRF. Results show that the average minimum sense range of the LRF is 3.92 centimeters.

7.3 PTEM Construction Experiment

This section describes and analyzes the experiment performed to construct the PTEM. The PTEM is constructed from the live data provided by the calibrated sensors. A single obstacle is used to construct the PTEM.

7.3.1 Experiment Setup and Results

The physical experimental setup consists of UGV mounted with all the sensors as described earlier, a laptop for data logging, an obstacle, and measuring tape. The X-Y axis representing navigation frame (NF) are drawn on the ground. The obstacle used in this case is a red cone with base diameter of 10 inches. The center of the cone is placed at [2,2]. The Revolution GS digital compass is used as an orientation sensor. Hokuyo LRF is used as the obstacle detection sensor. The encoders along with digital compass are used to identify the current position by *dead reckoning*. The

UGV is placed at the origin. The LRF data, digital compass data, and encoder readings are logged. Multiple scans of LRF are logged. All the scans are converted in to a X-Y array of obstacles in the navigation frame using current position and orientation data. The array of obstacles is passed to the Construct PTEM module. The output PTEM is plotted using Matlab.

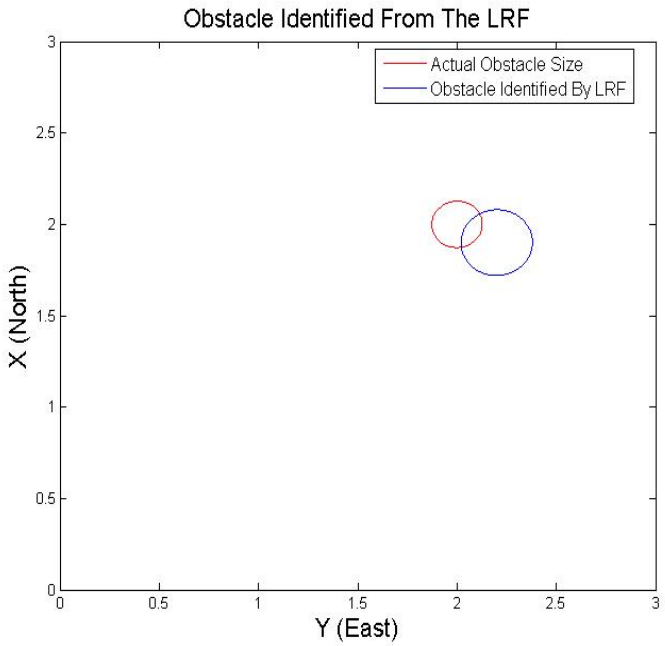

Figure 7.13. Actual Size Vs Identified Size by the LRF.

Figures 7.13 and 7.14 show the results obtain from the experiment. In Figure 7.13 blue circle indicates the actual size of the obstacle. While the red circle indicates the size of the obstacle identified by the LRF and clustering algorithm. Figure 7.14 indicates the PTEM, specifically the restricted area containing the obstacle, constructed by the PTEM construction module versus its actual size.

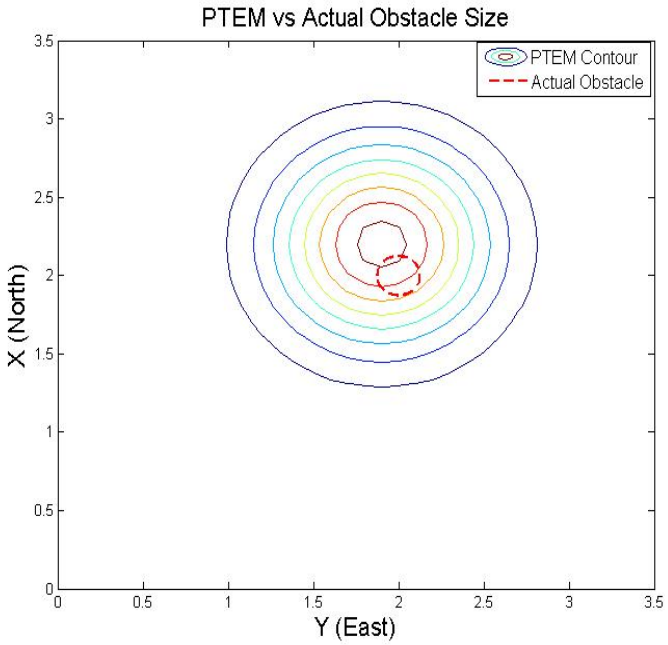

Figure 7.14. Constructed PTEM Vs Actual Obstacle Size.

7.3.2 Experiment Result Analysis

As shown in the Figure 7.13 the size of the obstacle identified is greater than the actual size of the obstacle. The obstacle identified also appears to be shifted from the actual location. The increase in size of the obstacle is actually due to the clustering algorithm developed. In the clustering algorithm developed, the size of the cluster keeps on growing. As the size increases the unwanted area or area where obstacle points are not present also increases. The shift in the obstacle position is due to the error in X-Y positions from position sensor. The error from the orientation sensor also adds to the total error. It is worth noting that the restricted area defined by the constructed PTEM fully encircles the actual obstacle, which should result in the avoidance of the obstacle by the guidance algorithm.

As indicated in Figure 7.14 the actual contour of the PTEM formed is larger in size compared to the actual size of the obstacle. This error is due to the variance

calculation of a cluster. Considering threshold as a constant value the size of counter is directly proportional to the variance.

7.3.2.1 PTEM Experiment Conclusions

The experiment concludes that the PTEM construction algorithm constructs an PTEM from the live data from the sensors. The clustering algorithm developed should be modified to construct a PTEM that models the obstacle accurately. Variance calculation should be modified to output accurate results. The area of PTEM contours developed should represent the area occupied by the obstacle clusters.

7.4 Waypoint Navigation Experiment

The waypoint navigation experiment is performed to examine the waypoint navigation ability of the guidance algorithm. Since the guidance algorithm is designed to track moving targets, waypoint navigation is carried out by following a virtual target. The data from the real-time experiment is compared with the data from the simulation experiment. The waypoints used for the real-time experiment are fed to the simulation model. The simulation tests are performed on simulation navigation and control model developed in simulink environment. The data generated by the simulation is sampled at various intervals and logged using scopes displays available within the Simulink package.

7.4.1 Waypoint Navigation Setup and Results in Simulation

The waypoints used for the experiment are shown in Table 7.1. A pre-defined PTEM is fed to the guidance algorithm for operation. All the obstacles are placed at the point [20000,20000]. When the tank reaches the last waypoint it stops. The

mission contains 5 waypoints in all. The initial position of tank is at the origin [0,0]. The tank passes through the waypoints in the order shown in the Table 7.1

Table 7.1. X-Y Waypoint Table

Waypoint No.	1	2	3	4	5
X Waypoint	2	4	2	0	2
Y Waypoint	2	8	10	5	2

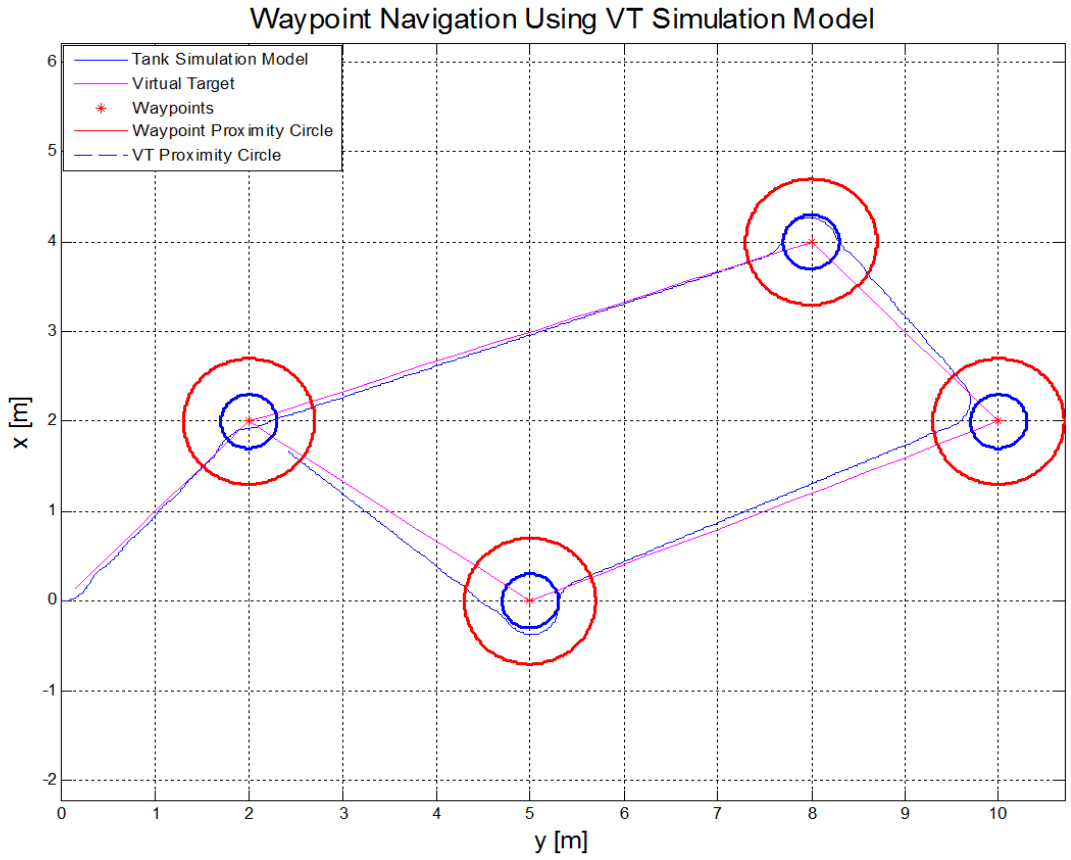

Figure 7.15. Waypoint Navigation via Virtual Target Simulation Results.

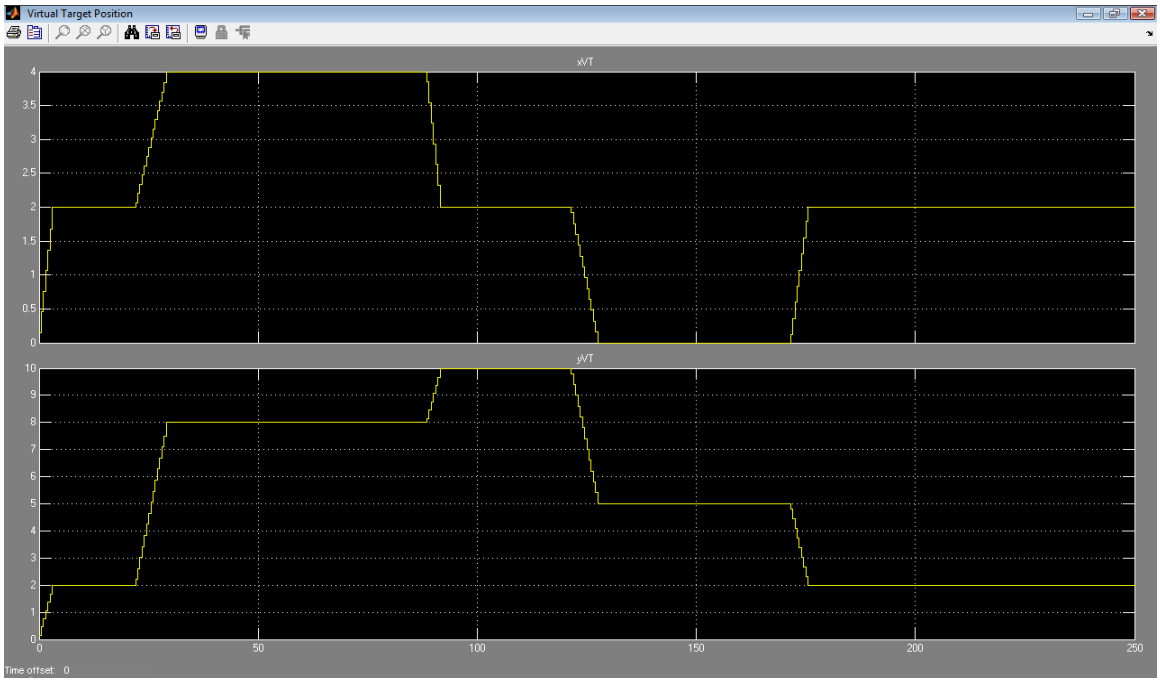

Figure 7.16. Waypoint Navigation via Virtual Target Simulation Results.

Figure 7.15 shows the simulation results of the navigation and control model. The magenta line represents the path taken by the virtual target. The blue line represents the path taken by the simulated tank. There are two circles around the waypoints. The red circle represents the waypoint proximity circle. The waypoint proximity circle is 0.7 meters. Once the vehicle is within the waypoint proximity circle, it implies that the vehicle has achieved the desired waypoint. If the waypoint achieved is not the last waypoint, the waypoint navigation block switches over to the next waypoint and the guidance module pursues the new waypoint. If the achieved waypoint is the last waypoint and the tank is within the waypoint proximity circle, a stop flag is outputted. This stop flag stops the tank model. The second yellow circle indicates the virtual target proximity circle. As mentioned earlier, the guidance algorithm is designed to follow a moving target. The virtual target represents the

moving target. The virtual target proximity circle determines how closely the vehicle wants to track the virtual target. The virtual target proximity circle is 0.4 meters for this simulation.

The virtual target dynamics are generated by the virtual target module. Figure 7.16 shows the X and Y values generated by the VT module. The VT dynamics include predicted position, orientation and velocity. The virtual target dynamics are generated for a particular waypoint. The virtual target navigates through the waypoints. Once a desired waypoint is achieved, the VT resides on that waypoint till the platform is within waypoint proximity circle. The waypoint is switched to the next waypoint when the above condition is satisfied.

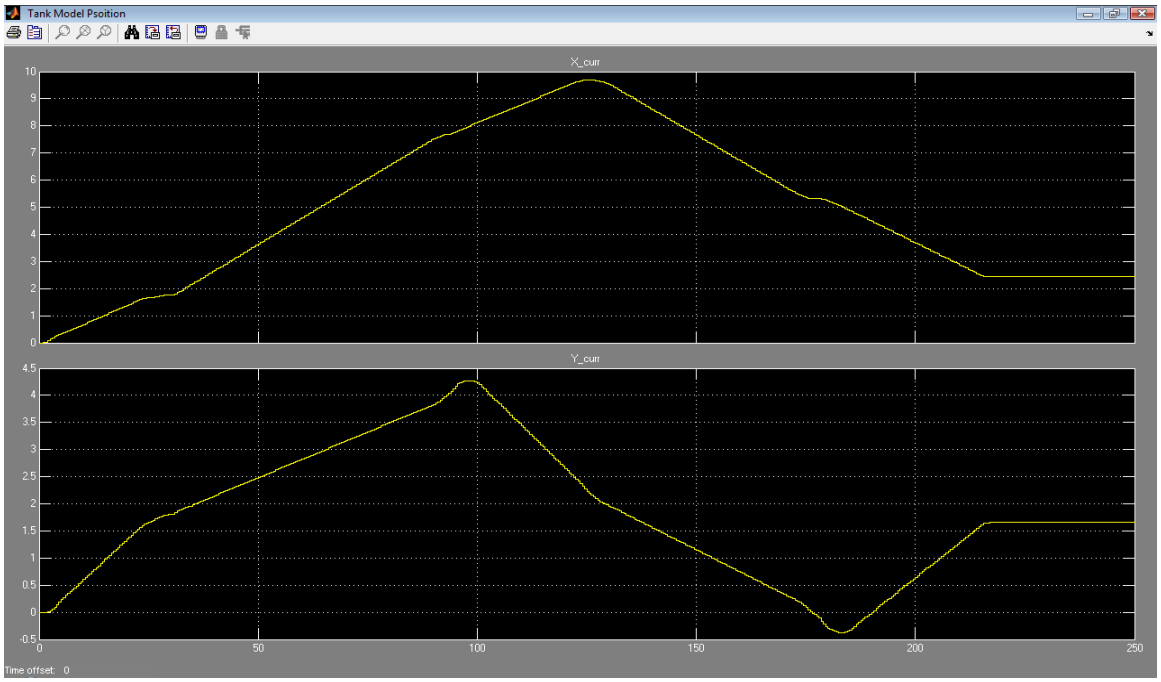

Figure 7.17. Waypoint Navigation via Virtual Target Simulation Results.

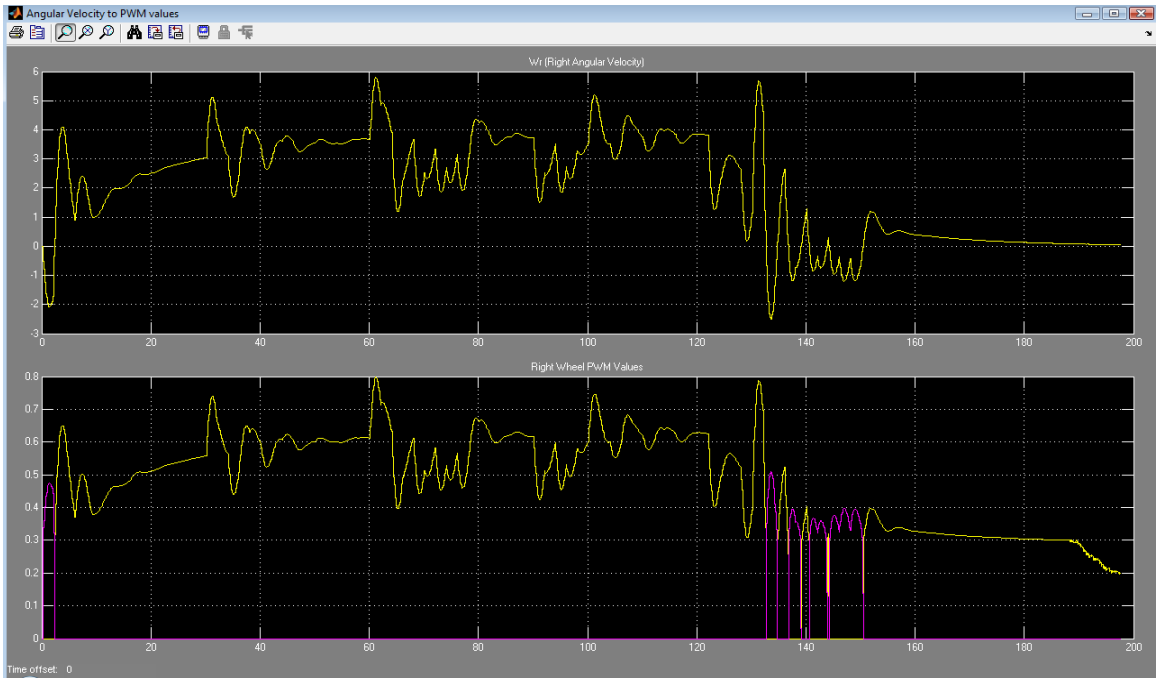

Figure 7.18. Angular Velocity to PWM values.

Figure 7.17 represents the actual X,Y positions achieved during the course of the tank model simulation. Figure 7.18 shows the conversion of angular velocity in to PWM values based on the lookup Tables 6.1 and 6.2. Table 6.1 converts the right wheel angular velocity (Wr) in to right wheel forward PWM value. Table 6.2 converts Wr in to the right wheel backward PWM value. The top graph of Figure 7.18 shows the right wheel angular velocity (Wr) outputted from the controller module. The bottom graph of Figure 7.18 shows the forward and backward channel PWM values. The forward channel is represented by yellow and the backward channel is represented by the magenta color.

7.4.2 Real-time Waypoint Navigation Experiment Setup and Results

The experimental setup of this experiment consists of a UGV equipped with a digital compass, rotary encoders, LRF and GPS receiver. An additional PlugaPod^{TM} is used for data logging. Data logging is done over the CAN network. The variables to be logged are written to CAN port B from the MPC555. The PlugaPod^{TM} is connected to CAN port B and receives all the CAN messages over the CAN bus. The PlugaPod^{TM} parses CAN messages and displays CAN ID and message information on NMIterm[2]. The screen of the NMIterm is captured and Matlab is used to interpret and plot the logged variables. X-Y axis constructed on the ground represent NF, i.e. X represents North and Y represents East. Each waypoint has a pre-determined proximity circle of 0.7 meters. The VT proximity circle is 0.4 meters. The waypoints to be pursued are shown in Table 7.1. To perform waypoint navigation current position and orientation estimations of the tank are required. Encoders are used to estimate current position using dead reckoning. GS revolution digital compass is used as an orientation detection sensor and provides current orientation data of the platform.

Figure 7.19 shows waypoint navigation by following the virtual target. The data logged during the real-time experiment consists of current position, current orientation (digital compass output), virtual target position, left and right encoder ticks, and commanded output values.

In Figure 7.19, the green line represents the path taken by the virtual target. The magenta line shows the path taken by the tank simulation model. The blue line represents the path taken by the actual tank during the real-time experiment. The waypoint proximity circle is indicated by a red circle around the waypoints. The VT proximity circle is shown by blue lines around the waypoints.

[2]NMIterm is a proprietary serial interface software by New micros for IsoPod^{TM} and PlugaPod^{TM}

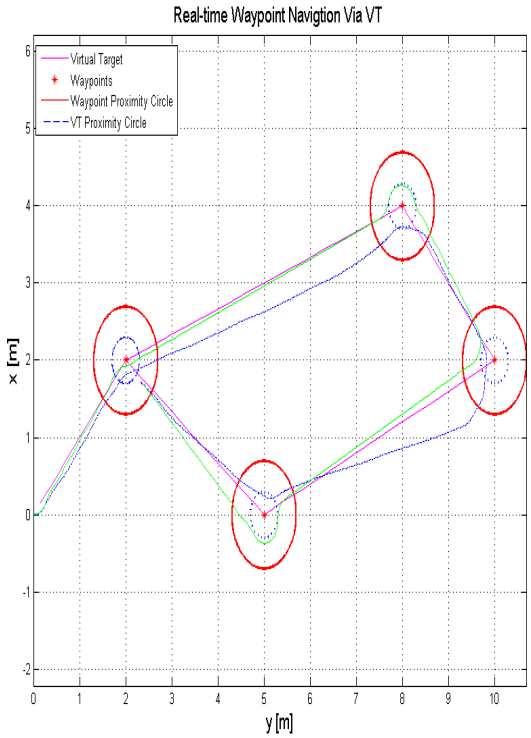

Figure 7.19. Waypoint Navigation via Virtual Target.

Figure 7.20 shows the GPS position estimates versus the position obtained over the waypoint navigation experiment. The data from the GPS receiver is not used during the generation of control signals. Differential GPS is used to conduct this experiment. RTCMv3 format is used for transmitting and receiving corrections. The GPS data is parsed and logged by the PlugaPod^{TM}. The data contains current time, latitude, longitude and other variables of the GPGGA data sting. GPGGARTK format is used for data logging on the rover station side.

Figure 7.21 shows the current heading and commanded heading. The current heading is identified by the digital compass.

Figure 7.22 shows the right and left encoder ticks. The encoder ticks are provided by the rotary encoder. The ticks of each rotary encoder is counted by a counter

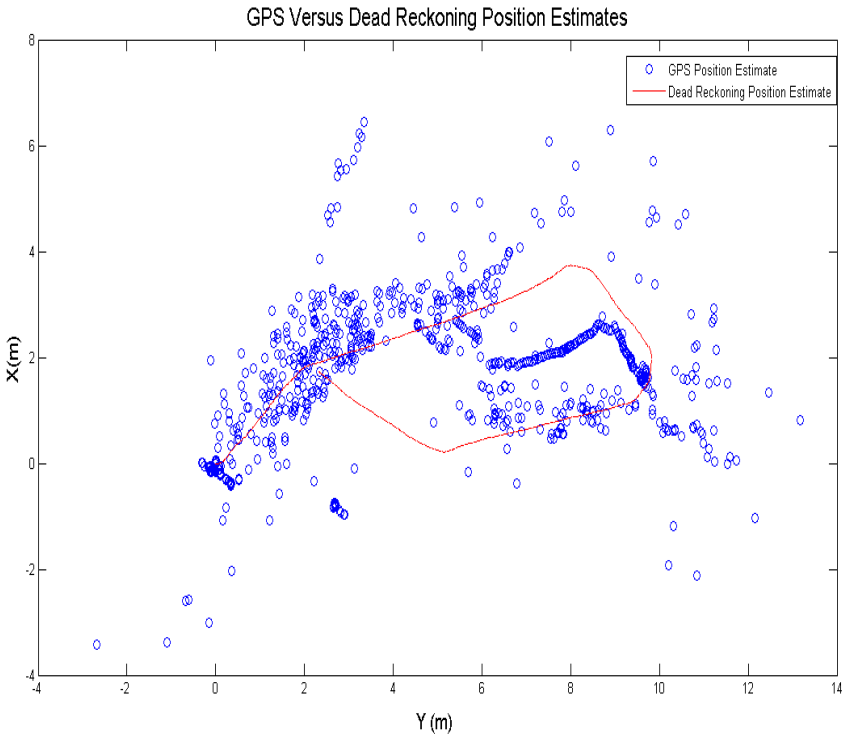

Figure 7.20. GPS Position Estimates versus Dead Reckoning Position Estimates.

in the MPC555 and the current position of the platform is calculated using dead reckoning.

7.4.3 Real-time Waypoint Experiment Result Analysis

As shown in Figure 7.19 the path taken by actual platform is close to that taken by virtual target. When the platform reaches the target waypoint, it pursues the next waypoint and stops at the last waypoint. The Figure 7.19 shows that the tank navigates through all the waypoints successfully. But, on the ground, the tank misses second waypoint [4,8] by 0.5 meters. The first and last waypoints are the same. On the ground, the tank stops before reaching the last waypoint. This discrepancy is due to the current position estimates provided by the rotary encoders. Dead reckoning

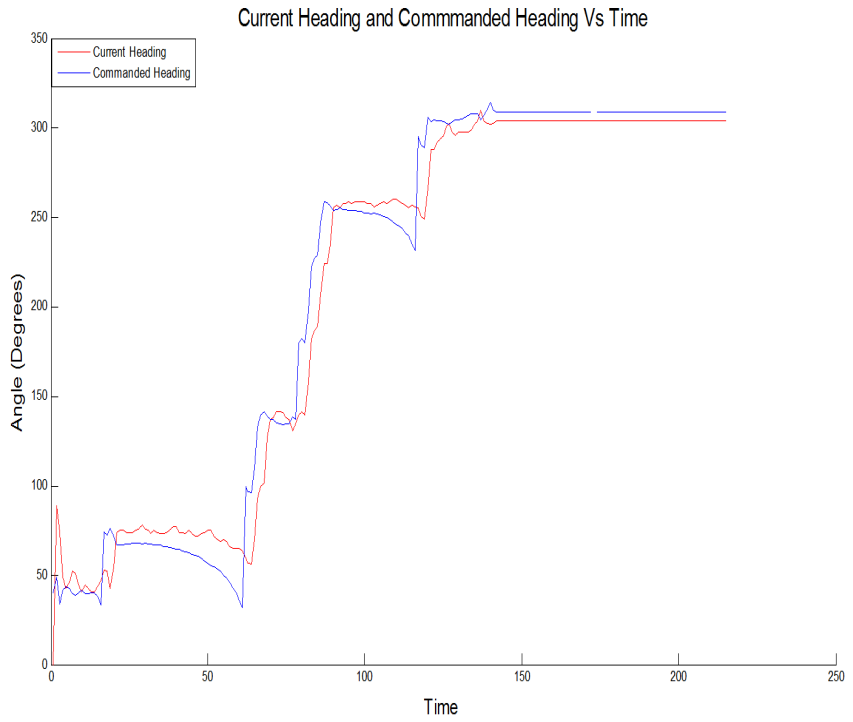

Figure 7.21. Current Heading And Commanded Heading Versus Time.

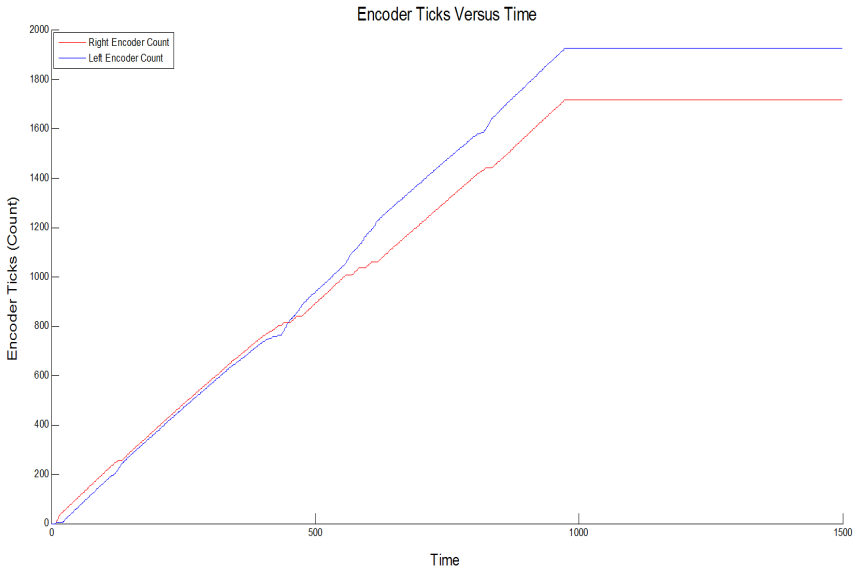

Figure 7.22. Encoder Ticks Versus Time.

is used to calculate current x and y position. In dead reckoning, a small position error accumulates over time and becomes large. So, to decrease this error, a active sensor like GPS can be used to provide better x and y position estimates.

As seen in the Figure 7.20, the GPS position estimates are not reliable and accurate. This partially opposes the claims made in the experiment in Section 7.2.2. In that experiment the GPS is moved with a constant speed in a straight line. During the current experiment the vehicle constantly changes speed and orientation in a short span of time. The RTK implementation solution has various ages. The ages are pseudo-correction, narrow flat and narrow integer. Narrow integer is the age with the maximum accuracy (0.1 cm CEP theoretically). When the tank changes orientation and speed, the current DGPS system changes the solution age to pseudo-correction which has lesser accuracy (80 to 150 cm CEP). It takes about 3 to 5 seconds for the GPS to attained the narrow integer correction age. In the mean while the tank again orients and changes speed. This effect causes the GPS to loose its accuracy over the waypoint navigation course.

7.4.4 Real-time Waypoint Experiment Conclusion

From the above results we can conclude that waypoint navigation part of the guidance module, CAN receive module, current orientation and position module, controller module, and PWM output modules function satisfactorily. If an accurate system is available, a GPS data should be used to provide better position updates over longer ranges. The virtual target dynamics sub-module developed works reliably. The other DGPS formats like RTCM, CMR must be tested for better GPS position estimates, while the vehicle is mobile and turning. The novatel GPS system used for the research project also supports proprietary formats. This formats also should be tested for accuracy.

7.4.5 Comparison Between Simulation Results and Real-time Results of Waypoint Navigation Experiment

Comparing the simulation and real-time experiments and Figures 7.15 and 7.19 we can conclude that:

1. Both the simulation tank model and real-time tank closely follow the virtual target
2. The simulation tank model functions reliably and repeatably over any distances. Whereas, the real-time tank model has a discrepancy in position calculations over distances greater than 10 meters.
3. The path followed by real-time tank is closely related to the path followed by simulation tank model.

7.5 Obstacle Avoidance and Waypoint Navigation Experiment

This experiment concludes this research project. Results and conclusion obtained from this experiment assist in evaluating the real-time obstacle detection and avoidance capability of this algorithm. Prior to the real-time experiment, the experiment is simulated and the results of simulation are compared with results from real-time experiment.

7.5.1 Obstacle Avoidance and Waypoint Navigation Setup and Results in Simulation

The waypoints used for this experiment are the same as that used in previous experiment, specified in Table 7.1. There are two simulations performed with the same obstacle placed at different locations. A PTEM is constructed using the given obstacles as specified in Table 7.2.

Table 7.2. PTEM

	Mean(μ)	**Variance(σ)**	**Threshold**	**Radius**
Obstacle 1	[5.94, 2.74]	[0.1033, 0.1033]	0.01	0.1270
Obstacle 2	[6.15, 0.6]	[0.1033, 0.1033]	0.01	0.1270

This PTEM is pre-fed into the guidance algorithm. The radius of the obstacle used for real-time experiments is 5 inches which is equivalent to 0.1270 meters. The threshold is kept fixed for the simulation. The variance is calculated using Equation 5.8. The mean of the obstacles represents the [x, y] co-ordinates of the obstacles in the real-time experiment. The target proximity circle and VT proximity circle is 0.7 and 0.4 respectively.

Figures 7.23 and 7.24 show the simulation results of obstacle avoidance and waypoint navigation using the virtual target for obstacles 1 and 2 respectively. As seen in the figures, at both the instances the simulation tank model avoids obstacle successfully. The waypoints for both the simulations are the same as shown in Table 7.1. The PTEM used for the simulation was described earlier. The target/waypoint proximity circle and virtual target proximity circle is represented by red and yellow circles respectively. The path taken by the tank model is represented by blue line. The magenta line represents the path taken by virtual target. The obstacles are represented by contours of PTEM constructed from Table 7.2.

Figures 7.25 and 7.26 represent the output of the guidance module for simulation results with obstacles 1 and 2 respectively. The top plots in both Figure 7.25 and 7.26 represent the commanded heading. The bottom graph in both the figures represent the commanded velocities. Due to the location of obstacle 1 in the waypoint course, it is encountered earlier than obstacle 2. This can be seen by the change in commanded heading during 60th second to 80th second of Figure 7.25. Similar observation is

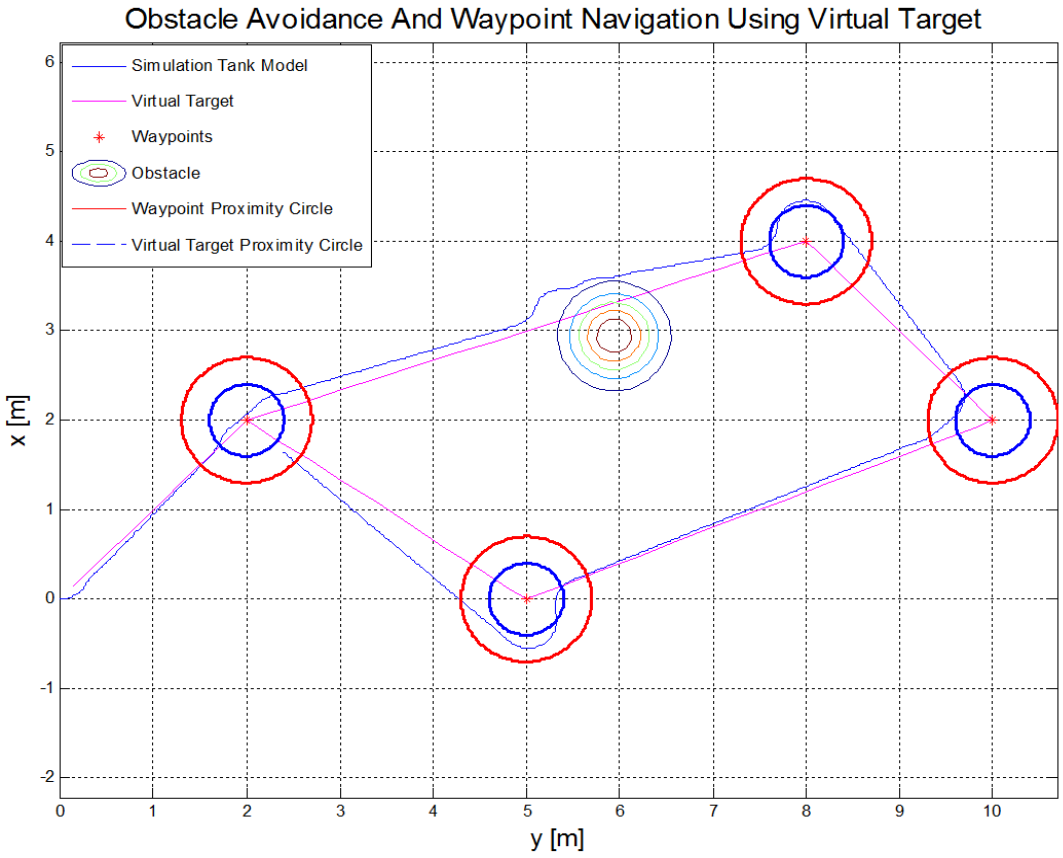

Figure 7.23. Waypoint Navigation and Obstacle Avoidance Via Virtual Target Simulation Results using Obstacle 1.

observed between the 210th second to 250th second sample of Figure 7.26 when obstacle 2 is encountered. The change in the heading indicates the guidance algorithm avoids obstacles when encountered. Since the sample time of the guidance module is 1.6 seconds, the outputs are updated every 1.6 seconds. The sample time of the digital compass and rotary encoders is 0.4 seconds. We needed something that is 4 to 5 times and in multiples of 0.4 seconds, so 1.6 seconds was selected as the sample time. Figure 7.27 represents the virtual target position (xVT and yVT) outputs. The virtual target advances or navigates through the waypoints in steps. The size of the step depends on the parameters such as estimation time, velocity and sample time.

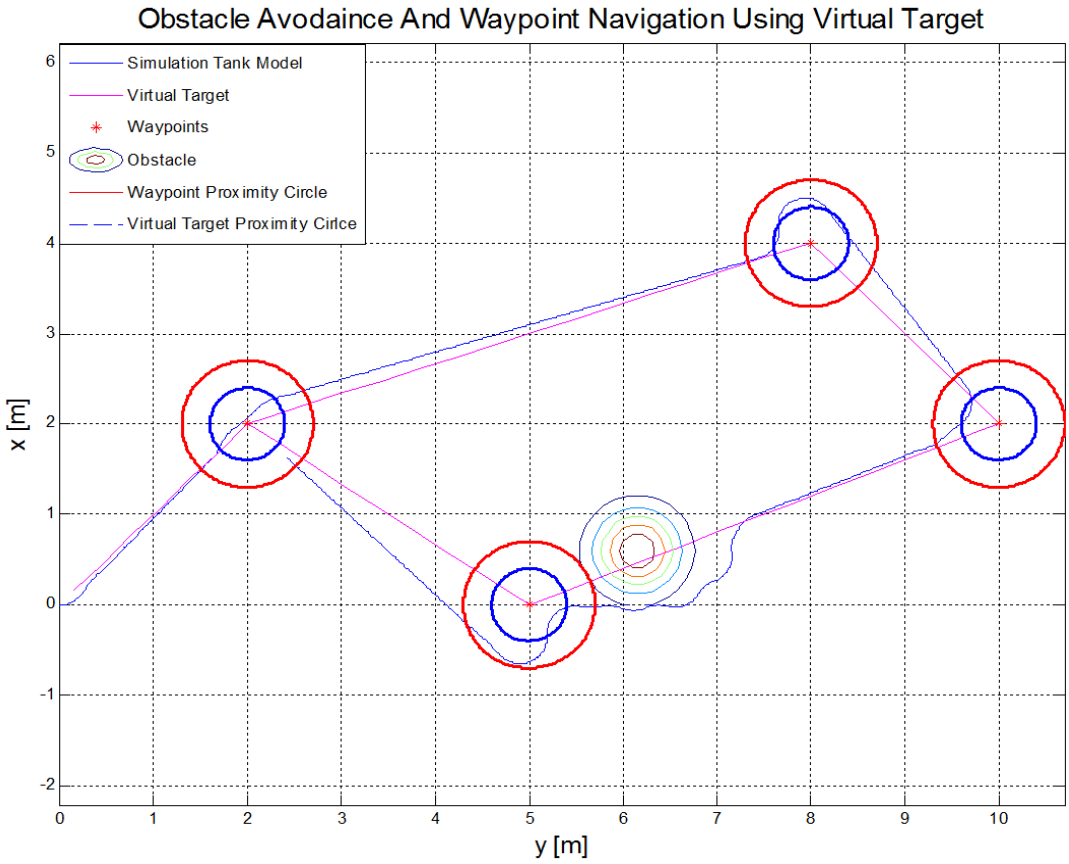

Figure 7.24. Waypoint Navigation and Obstacle Avoidance Via Virtual Target Simulation Results using Obstacle 2.

This parameters are fixed and are mentioned in virtual target dynamics module. The virtual target dynamics generated are same for both the simulations.

7.5.2 Real-Time Obstacle Avoidance and Waypoint Navigation Experiment

The experimental setup of this experiment is similar to the previous experiment. An UGV is equipped with a LRF, digital compass, rotary encoders and GPS receiver. Two real-time experiments are performed with the waypoints shown in Table 7.1. One obstacle is used for each real-time experiment. The obstacles for real-time experiments

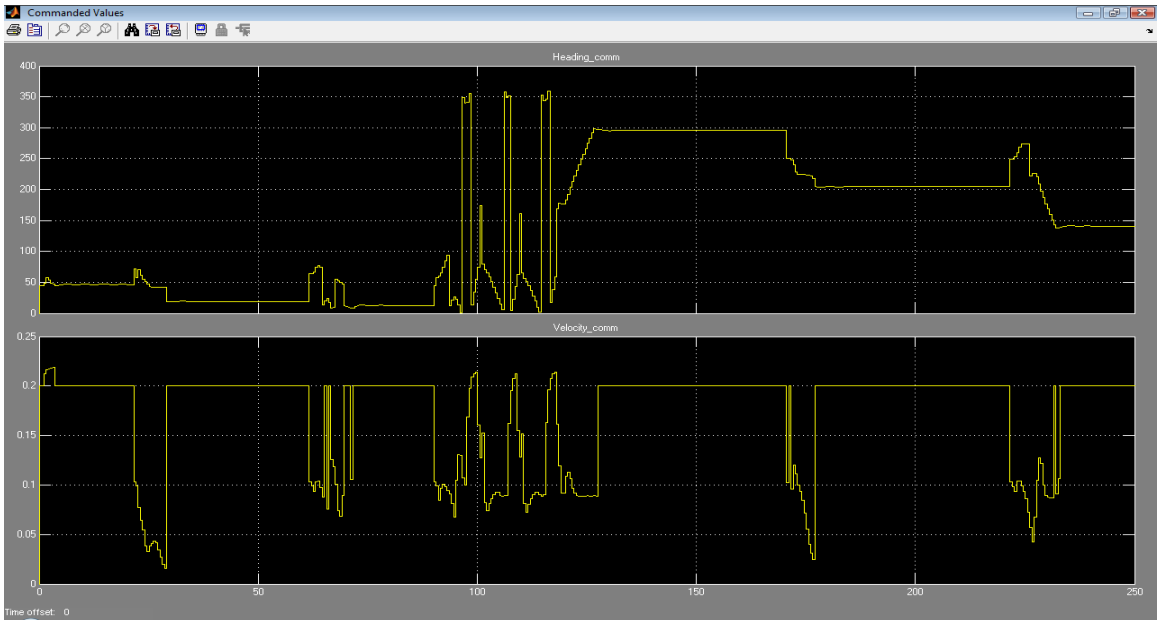

Figure 7.25. Commanded Heading and Velocity Outputs For Simulation using Obstacle 1.

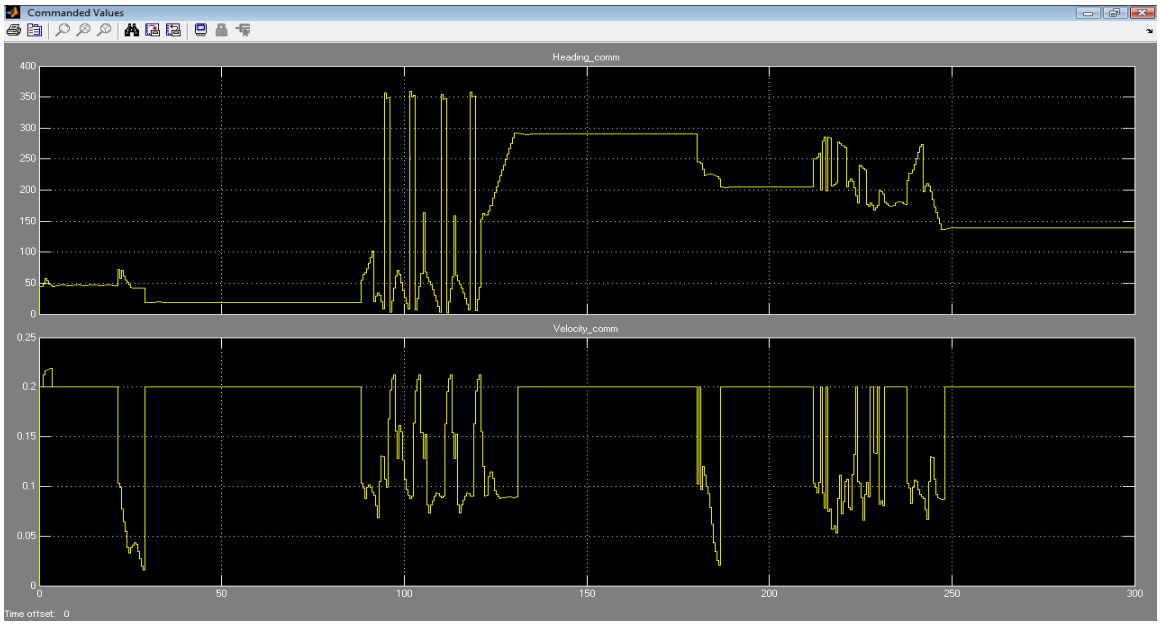

Figure 7.26. Commanded Heading and Velocity Outputs For Simulation using Obstacle 2.

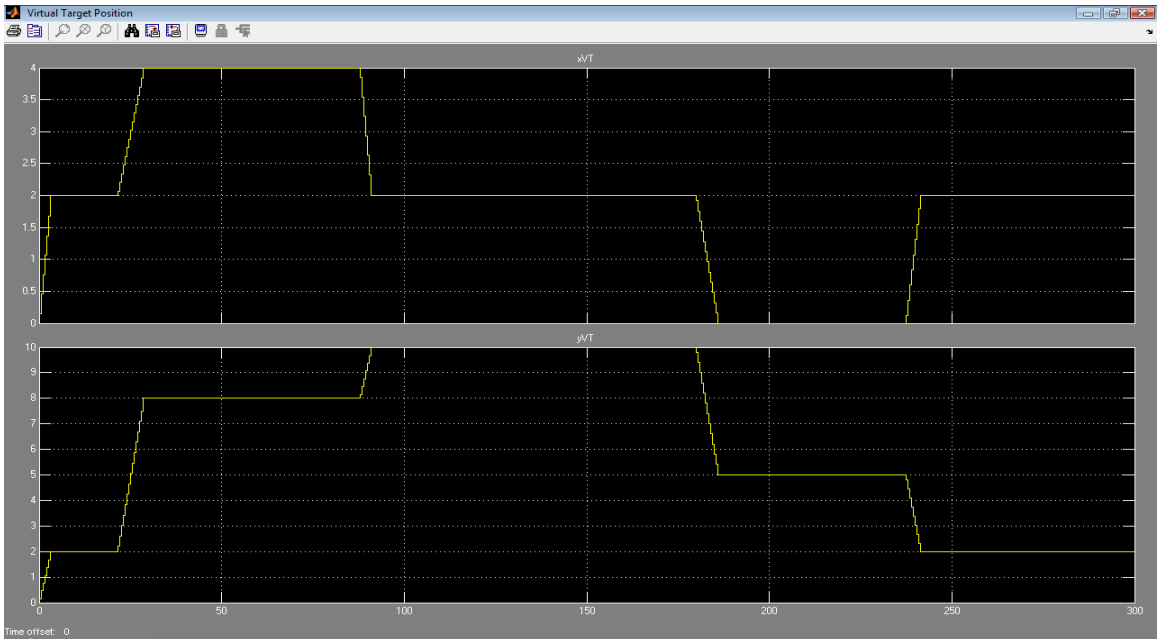

Figure 7.27. Virtual Target Position (xVT and yVT).

are placed at the same locations as in the simulation experiments, as given in Table 7.2.

A separate IsoPod^{TM} is used for data logging. Data logging is done over the CAN network, similar to previous experiments. The variables logged are LRF data, compass data, encoder data, PTEM value, current position, and commanded control signals. Matlab is used to plot logged variables. For this experiment, an X-Y grid is marked on the ground to represent the vehicle's navigational frame. The waypoints used in both the real-time experiments are shown in Table 7.1. The LRF is used to detect the obstacles. The digital compass is used to identify orientation. Encoder data, along with orientation data, is used to identify current position using *dead reckoning*.

Figure 7.28 shows the path followed by the tank in the real-time experiment. As seen in the figure, the tank does not avoid the obstacle, instead it collides with

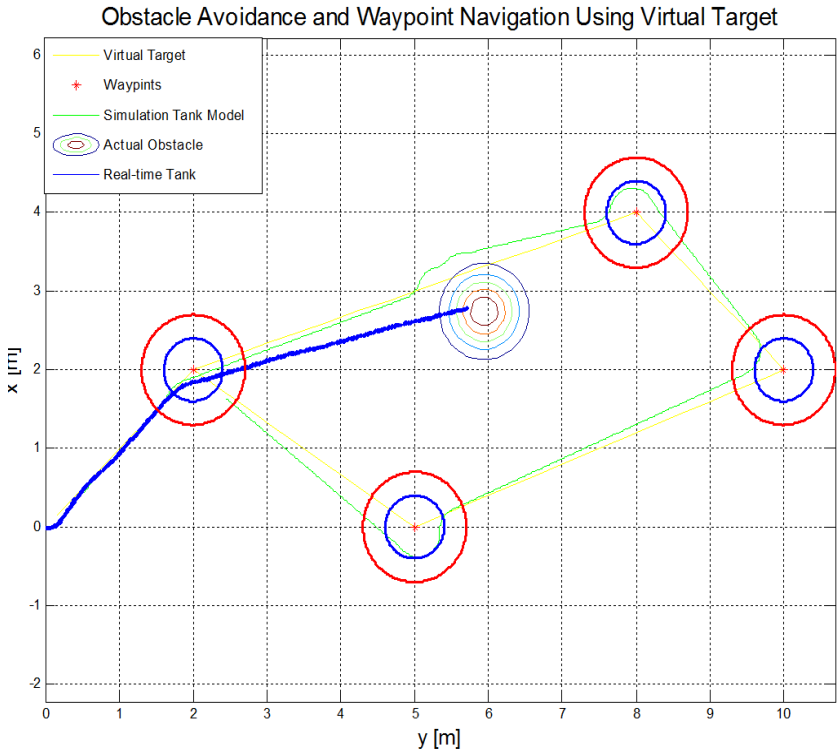

Figure 7.28. Real-Time Obstacle Avoidance and Waypoint Navigation for Obstacle 1.

the obstacle. This is a discrepancy between the simulation results and real-time results. The path followed by the tank is represented by the blue line. The yellow line represents the path taken by the virtual target and the green line shows the path taken by the tank simulation model. The waypoint proximity circle is indicated by a red circle around the waypoints. The VT proximity circle is shown by blue lines around the waypoints.

Figure 7.29 shows the PTEM formed in the real-time experiment. The PTEM constructed is does not represent the actual obstacle on the ground.

Figure 7.30 shows the actual heading versus commanded heading graph. The actual heading is measured obtained from the digital compass. The commanded heading is the output of the guidance module.

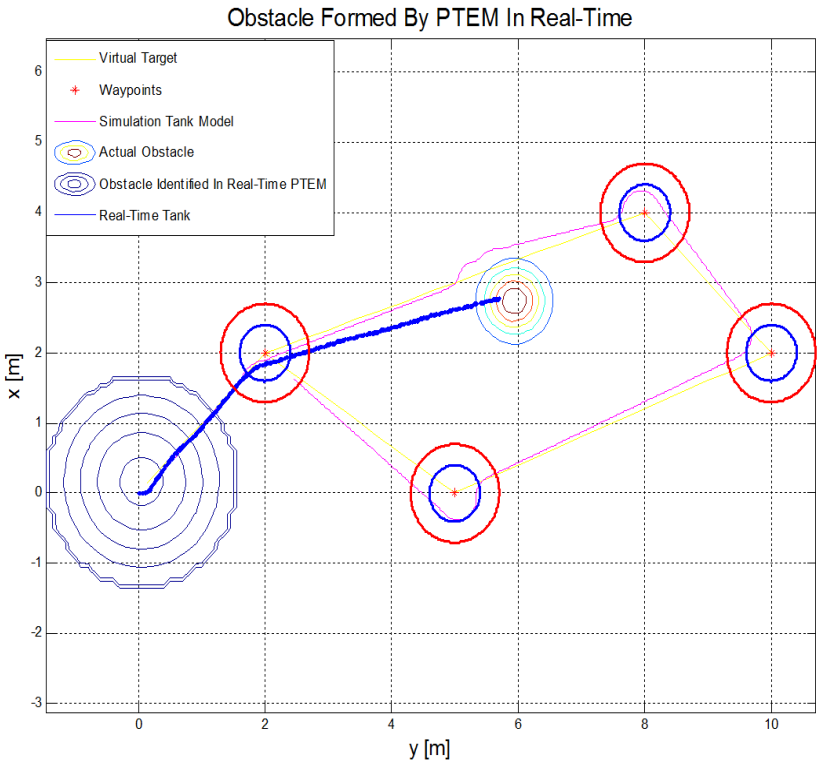

Figure 7.29. PTEM Identified in Real-Time for Real-Time Experiment using Obstacle 1.

The second real-time experiment consists of the same waypoints. The obstacle is placed as shown in Table 7.2. The setup of the experiment is similar to the first real-time obstacle avoidance experiment.

Figure 7.31 shows the real-time path taken by the tank in this experiment. As seen in the figure, the tank collides with the obstacle. The real-time path of the tank is shown by the blue line.

Figure 7.32 shows the PTEM constructed in real-time. As shown in the figure the PTEM constructed in real-time does not represent the actual obstacle on the ground. Figure 7.33 shows the actual heading versus commanded heading graph. As seen in the Figure 7.33 the actual heading follows the commanded value closely.

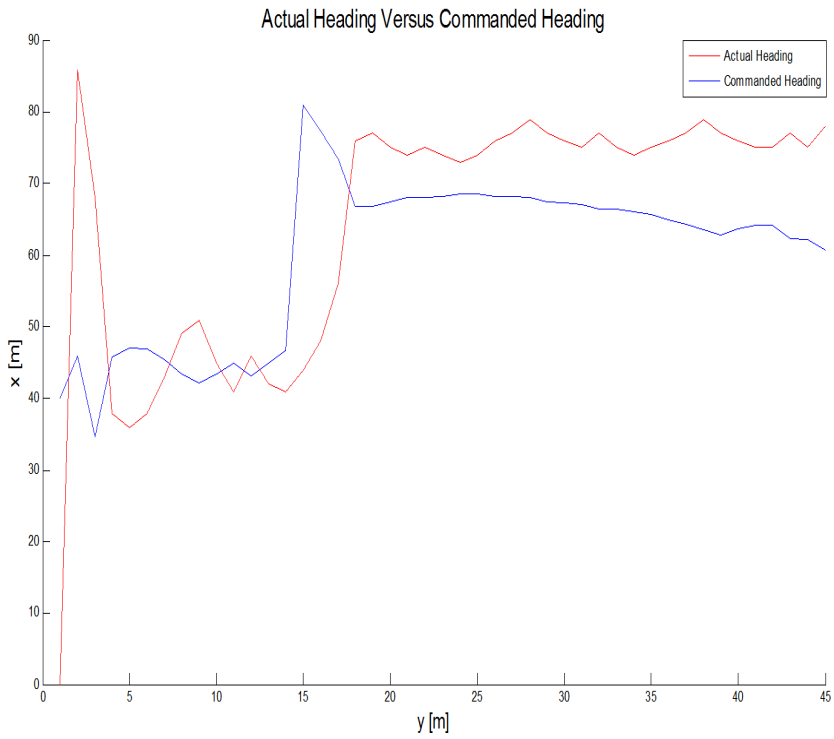

Figure 7.30. Actual Heading Versus Commanded Heading for Real-Time Experiment using Obstacle 1.

7.5.3 Real-time Obstacle Avoidance and Waypoint Experiment Result Analysis

From the above experiment we can conclude that the system developed does not perform obstacle avoidance in real-time. The simulations developed suggest that obstacle avoidance is performed successfully when the a pre-developed PTEM is entered in to the simulation system. As seen from Figures 7.29 and 7.32 the PTEM developed by the real-time system does not represent the actual obstacles on the ground.

From Figure 7.29, a PTEM is constructed in real-time which is at [0.06,0.17] with a variance of 0.02. The guidance algorithm developed does not avoid this PTEM. This is due to the fact that when the PTEM is constructed for the first time, the

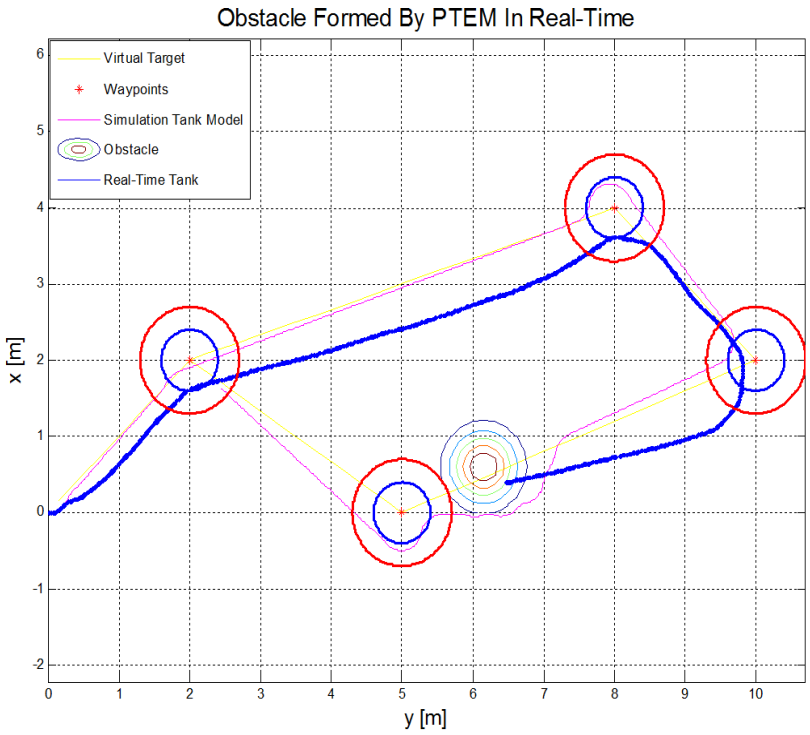

Figure 7.31. Real-Time Obstacle Avoidance and Waypoint Navigation Via Virtual Target for Obstacle 2.

real-time position of the tank is [1.99,2.5]. This means the tank has already passed the constructed PTEM in real-time. Figure 7.30 shows that the current heading follows the commanded heading within 10 degrees. This indicates that the guidance algorithm steers the vehicle directly towards the waypoint as the constructed PTEM does not show any obstacle in the way. From Figure 7.32, a PTEM is constructed in real-time which is at [1.95,7.26] with a variance of 0.02.

The failure to develop the PTEM may be due to the following reasons:

1. The parsed variables from the laser range finder over the CAN network are inaccurate and flawed.
2. The messages from the LRF, digital compass or encoders are not received properly.

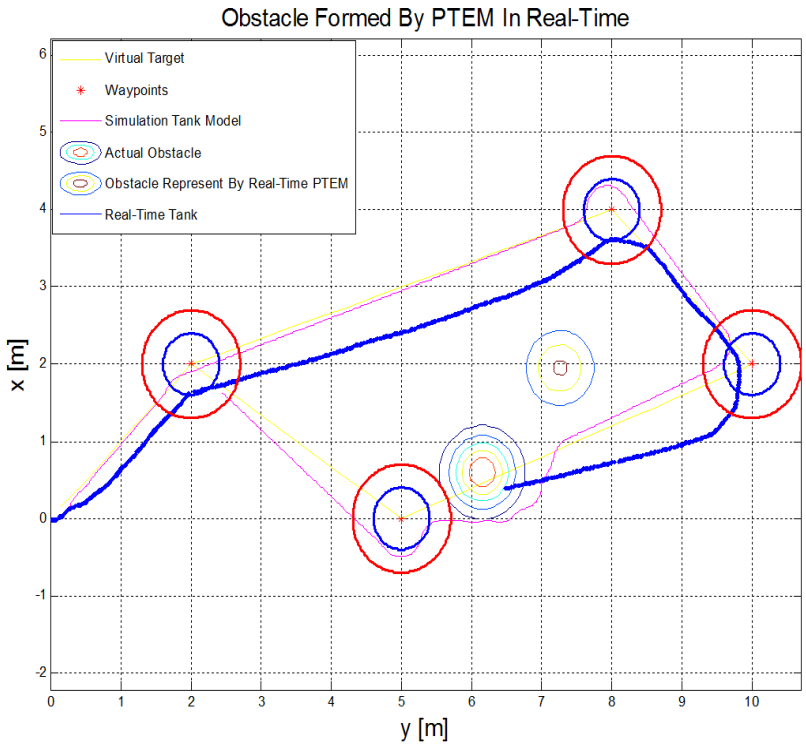

Figure 7.32. PTEM Identified in Real-Time for Real-Time Experiment using Obstacle 2.

3. The algorithm developed to construct the local map is faulty and incorrect.
4. The algorithm developed to construct the PTEM from local map is incorrect.

To address the discrepancy between the simulation and the real-time experiment all the reasons stated above are described and debugged in detail.

7.5.3.1 Inaccurate LRF Messages Over The CAN Network

The LRF is used to detect the obstacles. The LRF is connected to the IsoPod^{TM} for string parsing and the parsed variables are passed to the MPC555 over the CAN network. Each CAN message consists of the radial distance of the obstacle point and the radial angle of the obstacle point detected. The string parsing algorithm and the accuracy of the LRF is tested in the experiment in Section 7.2.3. From the conclusions

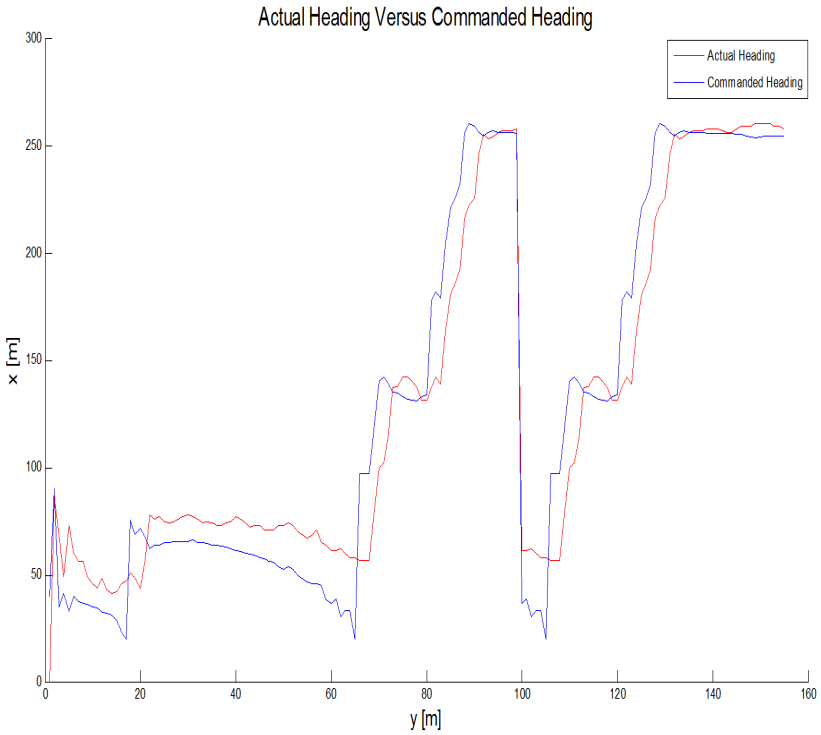

Figure 7.33. Actual Heading Versus Commanded Heading for Real-Time Experiment using Obstacle 2.

of experiment in Section 7.2.3, it is concluded that the correct string parsing routine is being performed by the IsoPod^{TM} and the LRF CAN messages passed on by the IsoPod^{TM} are correct and accurate.

7.5.3.2 LRF, Digital Compass and Rotary Encoder messages receiving inaccuracy

The digital compass and rotary encoders are read successfully and accurately in the real-time waypoint navigation experiment described in Section 7.4. The modules used in the real-time obstacle avoidance experiment are the same modules employed in the real-time waypoint navigation experiments. The data logs from real-time obstacle avoidance experiments are analyzed and the encoder data and digital compass data

is found accurate and reliable. Figures 7.30 and 7.33 also support this claim. From the data logs it is seen that reliable LRF data is received from the IsoPod^{TM} over the CAN network. The experiment described in Section 7.2.3 also supports this claim. The CAN receive block has been employed using a hardware buffer of 25 bytes to assure that no CAN messages are lost. The number of CAN messages received for each obstacle detected, ranges from 3 messages to 20 messages, depending on the radius and distance of the detected obstacle. We can therefore conclude that the data received from the LRF, digital compass and rotary encoders is reliable and accurate.

7.5.3.3 Inaccurate Local Map Formation

The inputs to the local map formation module are LRF data, digital compass data, and current position which is obtained from the dead reckoning block. From the above discussion it is concluded that the all of the sensor data is received correctly. The experiment described in Section 7.3 concludes that the local map formation and the PTEM formation module functions effectively and reliably in simulation. To support this claim a PTEM is constructed in simulation using the data logged during Experiment 1. Table 7.3 refers to the LRF data logged during Experiment 1. The separators indicate different scans from the LRF. The latest current position and orientation value is used to convert each scan in to an array of obstacles in NF. This table is converted into a local array of obstacles in the navigation frame (NF) in simulation by the same algorithm that is employed in the real-time system. This data is converted into a NF based obstacle array by the use of Equations derived in Chapter 5, Section 5.2.1.

All the obstacles are passed through the clustering algorithm. A single cluster center at [4.18,2.24] with a radius of 1.85 meters is created. The actual location of the obstacle is [5.94,2.74] with a radius of 0.1270 meters. The local map constructed

consist of an error of +/- 1.5 meters. This error is partly due to the error in position sensor and orientation detection sensor. This error can partly be eliminated by using precise position sensor and orientation detection sensor. The other problem is the real-time dynamic map building algorithm. The rotation and transformation matrix should be verified. The location of the obstacles detected in the real-time experiment is [0.06,0.17] which is largely erroneous and not comparable to the actual obstacle position.

This concludes that the local map formation algorithm does not work reliably in real-time. The local map formation algorithm developed works reliably in simulation environment.

7.5.3.4 Inaccuracies With PTEM Formation

From the above section, it is concluded that the local map formed at real-time is erroneous. The PTEM is constructed based local map. So, the PTEM constructed in real-time is bound to be incorrect since, the local map is incorrect.

Figure 7.34 depicts the PTEM that was generated by the simulation using the logged data generated during Experiment 1. As shown in the figure the PTEM generated is huge and does not cover the actual obstacle. In real-time, the PTEM generated is erroneous and does not represent the actual obstacle.

7.5.4 Conclusion from the Of Real-Time Obstacle Avoidance Experiment

The local map construction module does not function desirably in real-time. It works efficiently and reliably in simulation environment. The PTEM constructed in real-time does not represent the actual obstacles on the ground. The algorithms used in the local map formation module must be modified to adapt to real-time map local map formation. A real-time PTEM construction algorithm should be developed

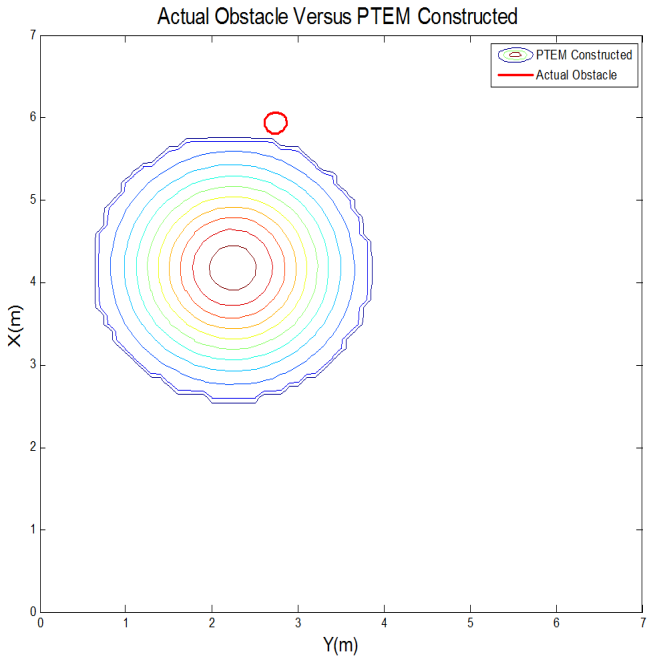

Figure 7.34. PTEM Constructed Versus Actual Obstacle.

that represents the real-time obstacles accurately. The clustering algorithm used to construct PTEM should produce efficient results i.e. the cluster develop should represent the are of the actual obstacle accurately.

7.5.5 Comparison between Simulation Results and Real-time Results of Obstacle Avoidance Experiment

The simulation setup and results are shown in the Section 7.5.1. The real-time experiment setup and results are shown in the section above. The major differences between the two experimental (simulation and real-time) results are given below:

1. The simulation model performs obstacle avoidance and waypoint navigation successfully and repeatably in simulation environment. While, the real-time tank model does not perform obstacle avoidance successfully.

2. The PTEM is successfully constructed and developed in simulation from obstacle points generated in the simulation. Successful PTEM is also constructed in simulation from the real-time obstacle points in the simulation environment. In real-time, the local map constructed is erroneous and inaccurate. The PTEM constructed subsequently from this local map is also erroneous and does not represent the dynamic environment in which the tank operates.
3. The flawed and erroneous construction of PTEM is the main reason for the failure of obstacle avoidance algorithm in real-time.

Table 7.3. LRF Readings From Experiment 1

LRF obstacle point	LRF obstacle point	Current Heading	Current X	Current Y
Radial Distance	Radial Angle			
3816	169	75.9	184	196
3816	170	75.9	184	196
2925	201	73.0	211	295
2908	202	73.0	211	295
2908	204	73.0	211	295
2055	204	75.9	234	289
2028	205	75.9	234	289
2008	207	75.9	234	289
2030	210	75.9	234	289
2065	211	75.9	234	289
1184	210	75.9	258	480
1153	211	75.9	258	480
1132	213	75.9	258	480
1127	216	75.9	258	480
1128	217	75.9	258	480
1125	218	75.9	258	480
1133	220	75.9	258	480
374	247	77	383	245
377	248	77	383	245
370	250	77	383	245
370	252	77	383	245
366	254	77	383	245
366	255	77	383	245
369	256	77	383	245
369	258	77	383	245
366	260	77	383	245
370	262	77	383	245
370	263	77	383	245
373	264	77	383	245
388	268	77	383	245
393	270	77	383	245
402	272	77	383	245

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Introduction

This Chapter will conclude this document and summarize the work done during this research. It will also give directions for future research.

8.2 Conclusions

To obtain accurate and reliable sensor sensor data, all the sensors must be calibrated in the real world operating environment. If the operating environment changes, recalibration of all the sensors must be performed to obtain accurate results.

Successful real-time code is generated from a simulink model using Matlab-Simulink toolchain with Real-Time Workshop. Real-time environment cannot always be accurately represented in simulation world. Simulink is used to perform successful simulation tests on the tank simulation model.

The serial data string parsing was unsuccessful on the MPC555 using the Matlab-Simulink toolchain. Successful serial data string parsing is achieved by the processors $IsoPod^{TM}$ and $PlugaPod^{TM}$ in Forth environment. CAN network is successfully used for inter-processor communication.

The Matlab-Simulink toolchain has limited real-time data logging capability on MPC555. Data logging capability was successfully implemented using $IsoPod^{TM}$ and CAN network.

The algorithm developed by Zengin performs successful waypoint navigation in real-time. It performs successful waypoint navigation and obstacle avoidance in

simulation environment. The construction of the local map, and thus the PTEM, was not successful due to possible problem in the algorithm. Due to inaccurate real-time map, the algorithm developed by Zengin could not perform obstacle avoidance in real-time.

Accurate position and orientation data is obtained from rotary encoders and GS Revolution digital compass. Obstacles are identified in real-time by the Hokuyo URG-04LX scanning LRF. The differential GPS system with RTCMv3 corrections provides current position estimates with an error of +/- 1 meter.

8.3 Suggested Future Work

The local map formation algorithm has to be modified to function reliably and repeatably in real-time. Some of the suggested reasons for the failure are:

- The conversion of LRF data into an array of obstacle points in the NF should be re-checked. The rotation matrix used to transform the data from body fixed frame (BFF) to the navigation frame (NF) should be confirmed and re-checked.
- The transformation from BFF to NF requires the current orientation and the current position of the vehicle. One LRF scan requires 3 to 4 seconds. The position, orientation and LRF sensors are operating at different sample rates. The sample rate of the position and orientation sensors is much faster than that of the LRF sensor. The data transfers occur between two LRF and slower sensors through rate transition block. The rate transition block acts as a dual buffer. Thus, the position and orientation data used to convert the BFF into the NF, in the worst case, can be delayed by about 6 to 8 seconds or previous copies of data are used. This causes enormous errors since the LRF scan is done in some orientation, but its transformation into NF is performed with some other orientation.

- Dead reckoning is used to obtain current X and Y positions. Errors in dead reckoning accumulate over time which causes the LRF data transformation to shift by the amount of error.

The PTEM was developed assuming static and cylindrical obstacles in a 2-D reference frame. This can be extended to mobile obstacles of any shape. Since the PTEM developed assumes a fixed maximum number of Gaussian distributions, a fading function can be employed to integrate a greater number of obstacles. This function will assign a fade value to each Gaussian pdf. If the obstacle is not detected within a given span of time the Gaussian pdf will be removed and its place will be assigned to another newly detected obstacle. This will ensure that the developed algorithm will successfully work with any number of obstacles.

The height of an obstacle is also not considered while modeling the obstacle as a Gaussian distribution. This can be included to extend the algorithm in 3-D space. To calculate height an image sensor or a 3-D scanning LRF can be employed.

After applying a threshold to the PTEM, restricted areas are created. The size of this restricted areas should be comparable to the actual size of the obstacle or obstacles it models. Currently, the size of the restricted area is potentially larger than the size of the obstacle or obstacles it models.

The variance calculation for the developed PTEM was done by an iterative method which has a constant threshold. A linear approach to calculating variance can be developed. The GPS sensor can be used for position information. It can be used to provide much more information like heading, velocity, time. By using this information, sensor fusion can be performed, which will provide more reliable and accurate data from the sensors.

The hardware setup including the physical wiring and location of components must be reliable and robust. All the sensors must be calibrated in real world environ-

ment and tested for accuracy, reliability, and repeatability. A data logging tool using CAN can be developed to log live data. CAN communication protocol (CCP) can be utilized to tune variables and for data logging. CCP is supported by Matlab-Simulink toolchain.

APPENDIX A

WIRING DIAGRAM OF HARDWARE IMPLEMENTATION ON THE UGV

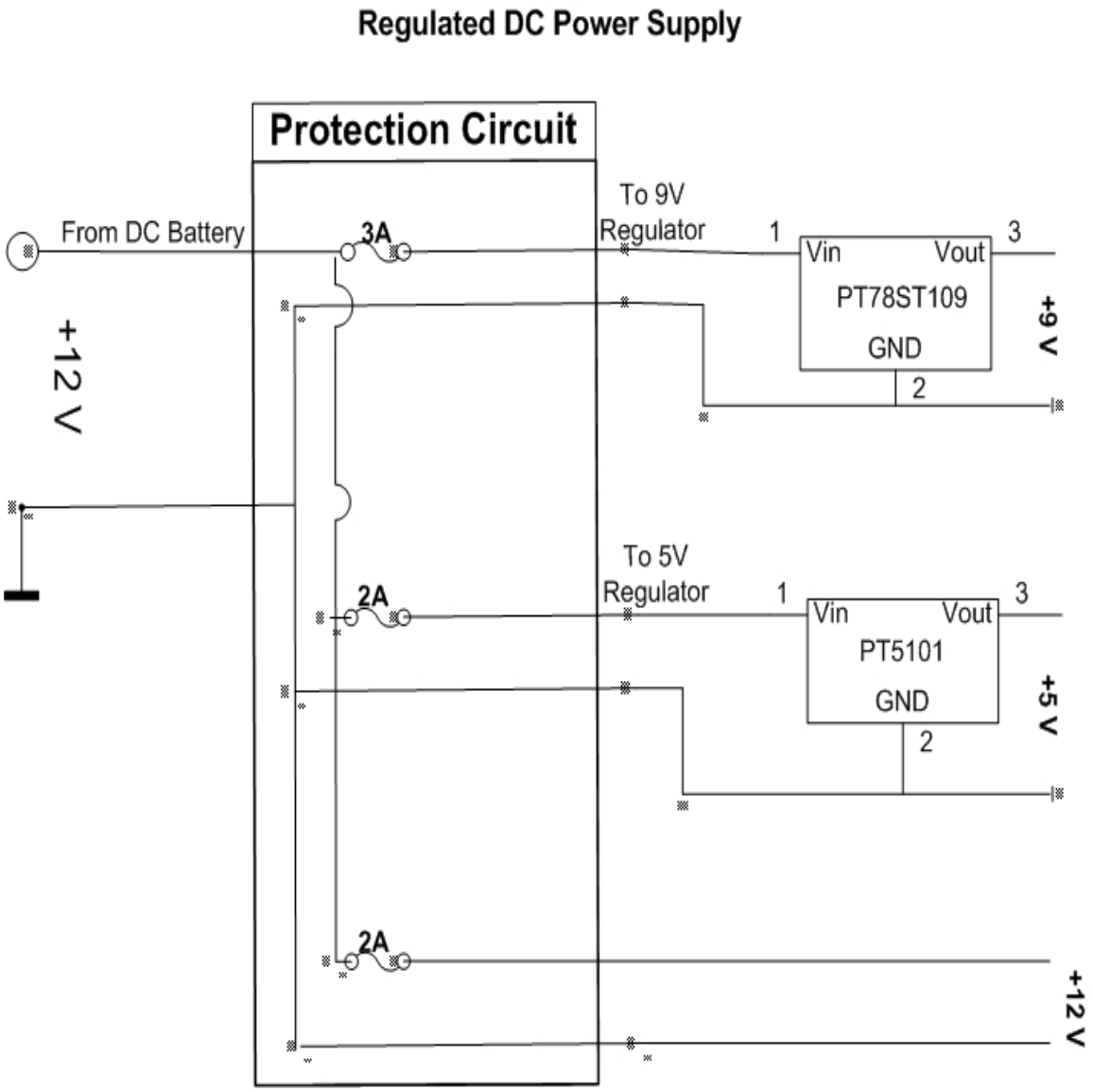

Figure A.1. Regulated DC Power Supply.

Wires and the corresponding colors on the physical tank.

- CANH = Green Blue
- CANL = White Blue
- GND = Yellow Blue

- GPS RX = Blue
- GPS TX = Green
- GPS GND = Black
- Digital Compass RX = Blue
- Digital Compass TX = Green
- Digital Compass GND = White
- Digital Compass Power +9V = Red
- Digital Compass Power GND = Black
- LRF RX = Blue
- LRF TX = Green
- LRF GND = White
- LRF Power +5V = Brown
- LRF Power GND = Blue
- Soft UART RX IsoPod 1 (PE7) = Pink
- Soft UART RX IsoPod 2 (PE7) = Green Blue
- Soft UART RX IsoPod 2 (PE5) = Red
- Rs232 to TTL Board Power GND = White
- Rs232 to TTL Board Power +5V = Red
- H-Bridge Right Forward = Blue Black
- H-Bridge Right Back = Yellow Violet
- H-Bridge Left Forward = Red White
- H-Bridge Left Back = Green Black
- Left Encoder CH0 = Purple
- Left Encoder CH1 = Orange
- Right Encoder CH0 = Grey Green
- Right Encoder CH1 = White Blue

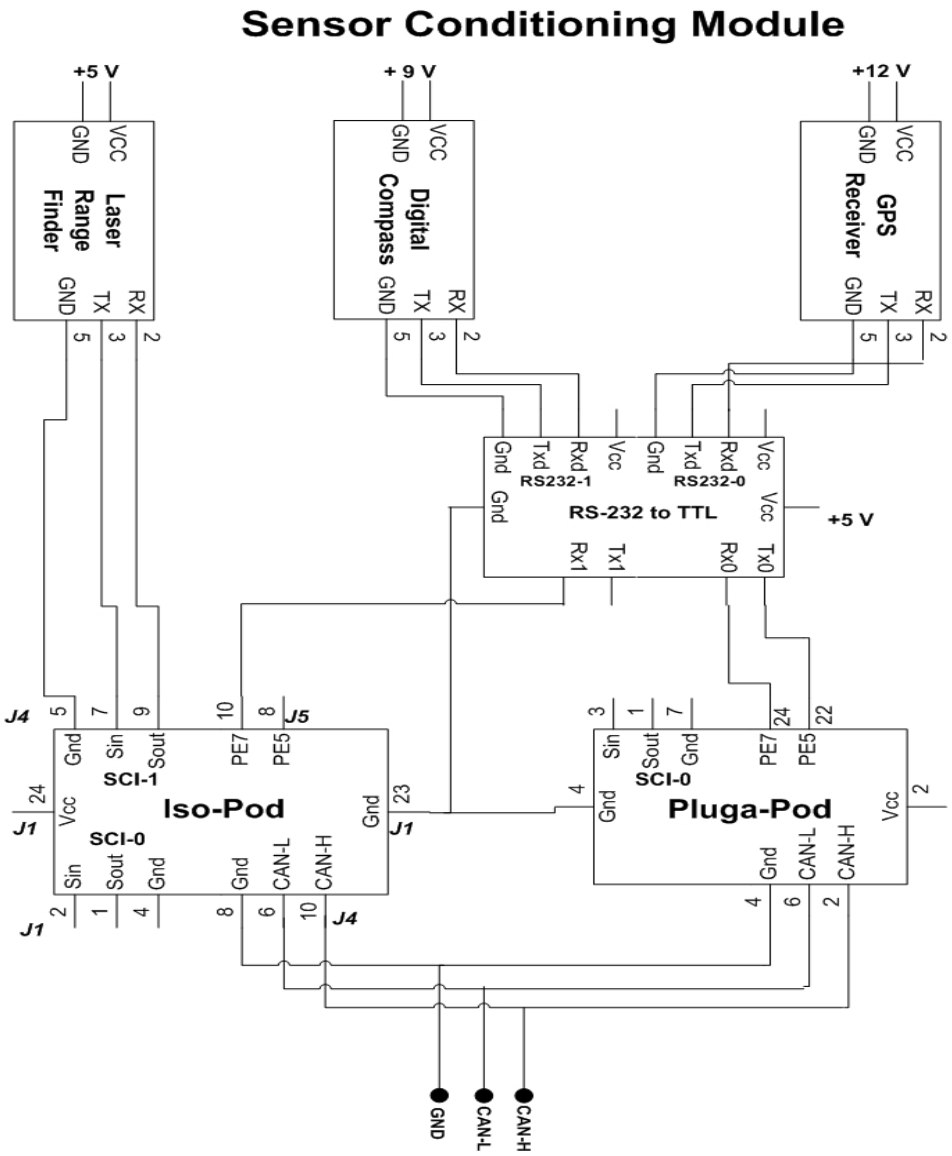

Figure A.2. Sensor Conditioning Module.

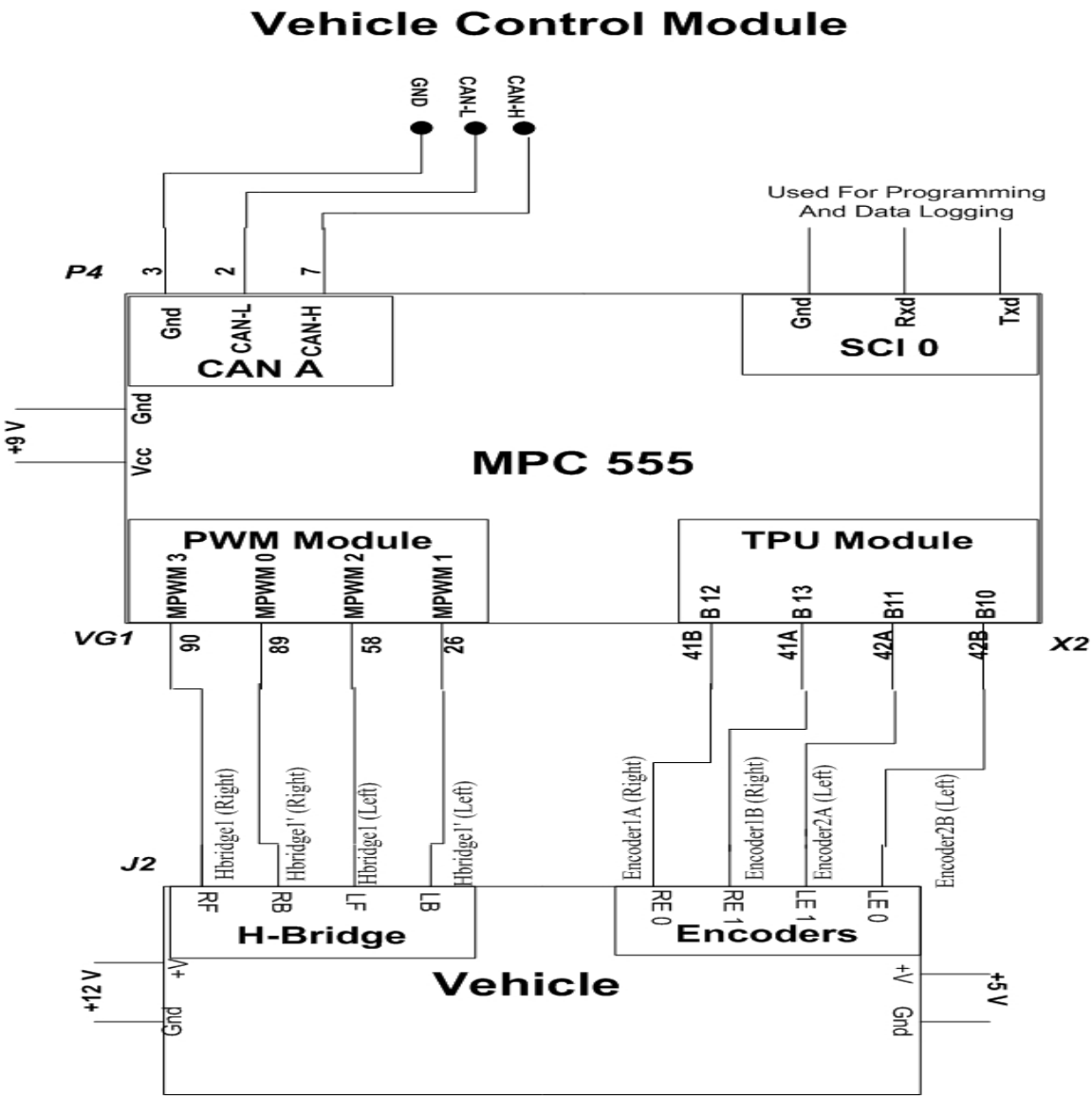

Figure A.3. Vehicle Control Module.

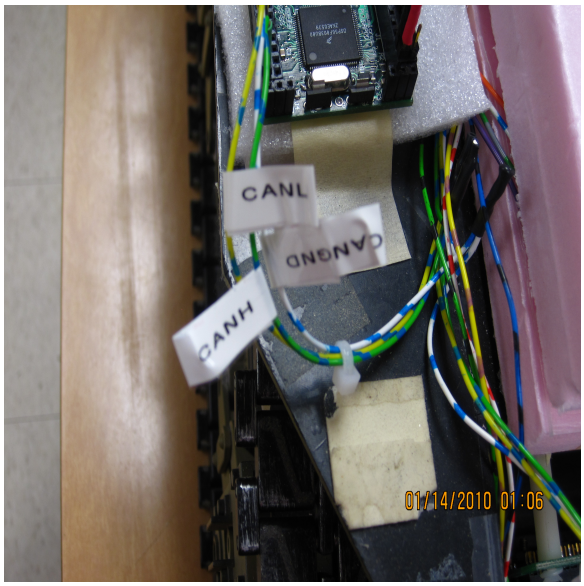

Figure A.4. CAN Wires.

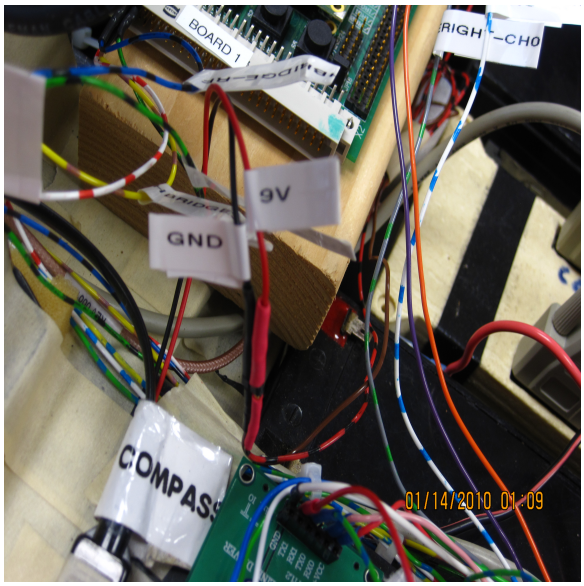

Figure A.5. Power Supply for Digital Compass.

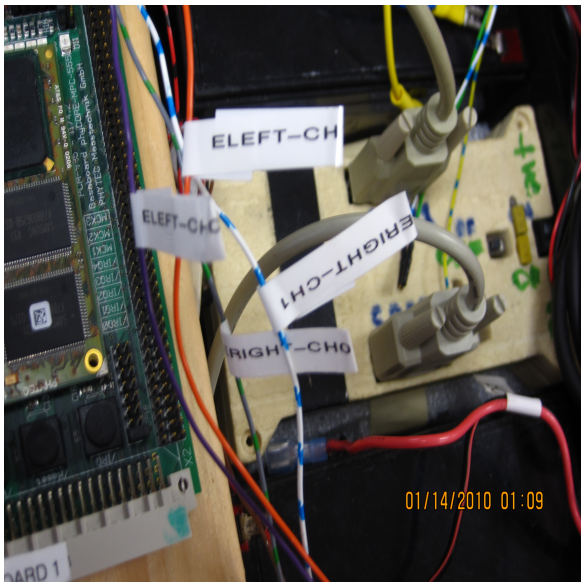

Figure A.6. Rotary Encoder to MPC555 Board.

Figure A.7. IsoPod 1.

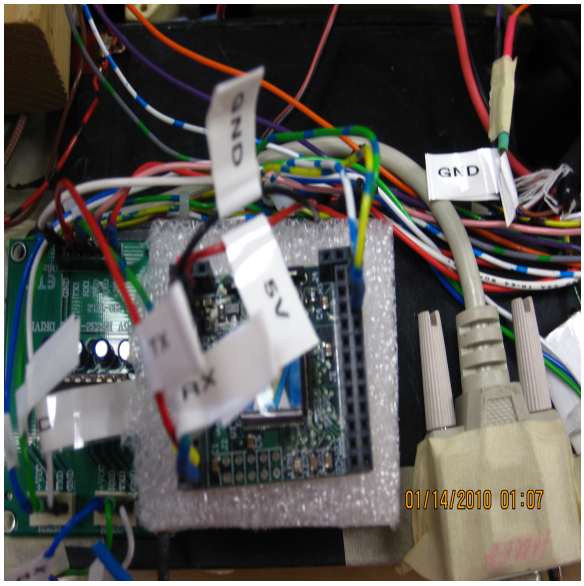

Figure A.8. IsoPod 2.

REFERENCES

[1] L. Wenchuan, Cai Weng and A. Dhaliwal, "Development of real-time control testbed for unmanned mobile vehicles," *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, November, 2006.

[2] Z. Ugur, "Autonomous and cooperative multi-uav guidance in adversial environment," PhD Dissertation, The University Of Texas At Arlington, Arlington,Texas, May 2007.

[3] Y. Michael, Jason Lum, "Advances in autonomy for small ugvs," *SPAWAR Systems Center, San Diego*, 2007.

[4] Z. Ugur and D. Atilla, "Autonomous guidance of uavs for real-time target tracking in adversarial environment," in *Chapter 34 in Aerial Vehicles*, Vienna, Austria, 2009.

[5] Z. Ugur and A. Dogan, "Probabilistic trajectory planning for uavs in dynamic environments," in *AIAA 3rd 'Unmanned Unlimited' Technical Conference, Workshop and Exhibit*, Chicago, Illinois, Sep 20 - 23 2004.

[6] A. Dogan, "Probabilistic approach in path planning for uavs," in *2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations - Aerospace, Land, and Sea Conference and Workshop and Exhibition*, San Diego, CA, Sep 15-18 2003.

[7] S. Marc, "Intelligent autonomy for unmanned naval systems," *Proc. SPIE, Vol. 6230, 623013 (2006)*, 2006.

[8] A. Pongpunwattana and R. Rysdyk, "Real-time planning for multiple autonomous vehicles in dynamic uncertain environments," *Journal of Aerospace Computing, Information, and Communication, vol. 1, pg. 580*, Dec 2004.

[9] D. N. Borys and R. Colgren, "Advances in intelligent autopilot systems for unmanned aerial vehicles," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, San Francisco, CA, USA, Sept 2005.

[10] Y. Joe Hadi, Kanaan and S. Maarouf, "A neural-network-based path generation technique for mobile robots," *IEEE International Conference on Mechatronics 2004,ICM'04, p 176-181*, 2004.

[11] H. Eiichi and T. Kazuo, "Probablistic approach to path planning with object boundary uncertainties," *Fifth International Conference on Advanced Robotics - '91 ICAR, p 1698*, 1991.

[12] M. Lydia, E. Kavraki and L. Jean-Claude, "Analysis of probabilistic roadmaps for path planning," *IEEE transactions on robotics and automation, VOL. 14, NO. 1*, February 1998.

[13] H. Yongxing and A. Sunil, "Planning and control of ugv formations in a dynamic environment: A practical framework with experiments," *Robotics and Autonomous Systems, v 51, n 2-3, p 101-110*, May 31, 2005.

[14] H. Maohai, Li Bingrong and C. Zesu, "Learing sensor based mobile robot simultaneous path planning and map building," *Proceedings of 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering, IEEE NLP-KE'05, v 2005, p 802-807*, 2005.

[15] E.-S. Jung-Hwan, Jang and Kim, "Stereo camera-based intelligent mobile robot system for path planning and navigation," in *Unmanned Ground Vehicle Technology VII*, Orlando, FL, USA, March 2005.

[16] N. NajJaran, H. Kircanski and A. Goldenberg, "Map building for a terrain scanning robot," in *International Conference on Robotics and Automation*, Seoul, Korea, May 21-26 2001.

[17] K. Shoji Suzuilt Aakira, Uegakil Shinya and T. Fujitat, "An infra-red sensory system with local communication for cooperative multiple mobile robots," in *IEEE International Conference on Intelligent Robots and Systems,*, 1995.

[18] J. Wilson, N., *Principles of Artificial Intelligence.* Berlin: Springer Verlag, 1982.

[19] J. Nilsson, N., "A mobile automaton: An application of artificial intelligence techniques," in *Proc. 1st Int. Joint Conf. on Artificial Intelligence*, Washington D.C., 1969.

[20] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized voronoi diagrams," *IEEE Transactions on Robotics and Automation, vol. 5, no. 2, pg. 143-150*, Apr. 1989.

[21] J. Borenstein and Y. Koren, "The vector field histogram- fast obstacle avoidance for mobile robots," *IEEE Journal of Robotics and Automation 7(3)*, June 1991.

[22] I. Ulrich and J. Borenstein, "Vfh+: Reliable obstacle avoidance for fast mobile robots," *IEEE International Conference on Robotics and Automation, pg 1572*, 1998.

[23] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," *In Proc. ICRA, pages 341-346*, 1999.

[24] M. Lodaya and R. Bottone, "Least squares approach to asynchronous data fusion," in *SPIE, The International Society for Optical Engineering, vol. 4048, pg. 333-344*, 2000.

[25] Y. Zhou, "A kalman filter based registration approach for asynchronous sensors in multiple sensor fusion applications," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Quebec,Canada, May 2004.

[26] P. J. Lanzkron and Y. Bar-Shalom, "A two-step-method for out-of-sequence measurements," in *IEEE Aerospace Conference*, 2004.

[27] S. Rao, Nageswara, "Projective method for generic sensor fusion problem," *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, p 1-6*, 1999.

[28] B. Ahuja, G. Sights and H. Everett, "Sensor fusion for intelligent behavior on small unmanned ground vehicles," in *SPIE - The International Society for Optical Engineering, Unmanned Systems Technology IX*, 2007.

[29] G. Arpit, Gupta Abhishek and Sastry, "Avoidance of threat zone by uav for automated navigation," *INDICON 2008 IEEE Conference and Exhibition on Control, Communications and Automation*, December, 2008.

[30] B. Hyeong, Jik Kong Ho and K. H. Sung, "The verification of uav's autopilot system with realtime hils," *ICMIT 2005: Control Systems and Robotics*, 2005.

[31] H. Zhiqiang, Ma Fangdong and Y. Zhenhua, "Multi-agent systems formal model for unmanned ground vehicles," *2006 International Conference on Computational Intelligence and Security*, October, 2006.

[32] P. Armando, Rodriguez Mariano and D. Jeff, "Description of a modeling, simulation, animation, and real-time control (mosart) environment for a class of 6-dof dynamical systems," *American Control Conference, 2007*, July 2007.

[33] W. C. Castillo, Moreno and K. P. Valavanis, "Unmanned helicopter waypoint trajectory tracking using model predictive control," *Control and Automation, 2007. MED '07*, June 2007.

[34] K. S. C. Taner, Mutlu Sertac and B. Ismail, "Development of a cross-compatible micro-avionics system for aerorobotics," *2007 IEEE Intelligent Vehicles Symposium*, June, 2007.

[35] "http://www.semiconductors.bosch.de/en/20/can/index.asp/," 2009, the CAN Website.

[36] "http://www.forth.com/starting-forth/," forth Online Book.

[37] "http://aprs.gids.nl/nmea/," the NMEA GPGGA string/log information.

[38] "http://en.wikipedia.org/wiki," the Wikipedia Website.

[39] "www.freescale.com/codewarrior," the Code Warrior Development Studio.

[40] "http://www.hokuyo-aut.jp/02sensor/07scanner/urg-04lx.html," 2009, the Hokuyo LRF Website.

[41] "http://www.novatel.com," 2009, the Novatel Website.

[42] "www.rtcm.org," the Radio Technical Commission for Maritime Services.

[43] "http://www.newmicros.com," 2009, the New Micros Website.

[44] "http://www.freescale.com," 2009, the Freescale Website.

[45] "http://www.phytec.com," 2009, the Phytec Website.

[46] "http://www.mathworks.com," the Mathworks Website.

[47] A. Dogan, "Probabilistic approach in path planning for uavs," in *IEEE International Symposium on Intelligent Control*, Houston, TX, Oct 5-8 2003.

[48] A. S. Miralles and A. S. Bobi, "Global path planning in gaussian probabilistic maps," *Journal of Intelligent and Robotic Systems*, Jan. 2004.

[49] J. H. Victoria and A. Jim, "Hierarchical growing cell structures: Treegcs," *IEEE Transactions on knowledge and data engineering*, March/April 2001.

[50] R. Bachmayer and N. E. Leonard, "Vehicle networks for gradient descent in a sampled environment," in *41st IEEE Conference on Decision and Control*, Las Vegas, Nevada, Dec. 2002.

[51] K. Konolige, "A gradient method for realtime robot control," in *International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, Jan. 1996.

[52] A. Schimel, "http://www.mathworks.com/matlabcentral/fileexchange/14804-wgs2utm," April 2007.

[53] P. Snyder, J., *Map Projections - A Working Manual*. Taylor and Francis, 1987.

[54] "http://www.lessemf.com," the Less EMF Website.

[55] "http://www.cmtinc.com/gpsbook/chap6.html," 2009, the GPS book: Introduction to the Global Positioning System for GIS and Traverse.

[56] U. Zengin and D. Atilla, "Real-time target tracking for autonomous uavs in adversarial environments: A gradient search algorithm," in *45th IEEE Conference on Decision and Control*, San Diego, CA, Dec 13-15 2006.

[57] U. Zengin and A. Dogan, "Multi-uav rendezvous in hostile environment," in *Ankara International Aerospace Conference*, METU Ankara, TURKEY, Aug 22-25 2005.

[58] B. Leo, *Starting FORTH*. CA, USA: Forth, Inc., 1981.

[59] U. Zengin and A. Dogan, "Real-time target tracking for autonomous uavs in adversarial environment: A gradient search algorithm," *IEEE Transactions on Robotics, v. 23, n. 2, pp. 294-307*, 2007.

[60] A. Dogan and K. Kaewchay, "Design of a probabilistic human pilot: Application to microburst escape maneuver," *AIAA Journal of Guidance, Control and Dynamics, v. 30, n. 2, pp. 357-369*, 2007.

[61] U. Zengin and A. Dogan, "Cooperative target tracking for autonomous uavs in an adversarial environment," in *Guidance, Navigation, and Control Conference*, Keystone, CO, Aug 21-24 2006.

[62] S. Bogdan and R. Maciej, "Autonomous control system for a 3 dof pitch-plane suspension model with mr shock absorbers," *Computers and Structures, v 86, n 3-5, p 379-385*, February 2008.

[63] "http://www.nmea.org," the NMEA Website.

[64] U. Zengin and D. Atilla, "Target tracking by uavs under communication constraints in an adversarial environment," in *AIAA Guidance, Navigation, and Control Conference*, San Francisco, CA, Aug 15-18 2005.

BIOGRAPHICAL STATEMENT

Pranav Desai was born in Ahmedabad,India 1986. He received his Bachelor's degree in electrical engineering (B.E) from Mumbai University, India in 2007. He is currently pursuing his Master's in Science (M.S) degree in electrical engineering from University of Texas at Arlington. His interests include real time operating systems, embedded systems, robotics, obstacle avoidance and path planning, and control systems. His currently research includes implementation of guidance and path planning algorithms on an unmanned ground vehicle.