# ROBUST CONTROL OF SKID STEERED ROBOTIC VEHICLES ON HIGH FRICTION SURFACES

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Andrew Kristian Narvesen

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Mechanical Engineering

April 2015

Fargo, North Dakota

North Dakota State University
Graduate School

**Title**

Robust Control of Robotic Skid Steered Vehicles on High Friction Surfaces

**By**

Andrew Kristian Narvesen

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Majura Selekwa
Chair

Dr. Yechun Wang

Dr. Mariusz Ziejewski

Dr. Jacob Glower

Approved:

| | |
|---|---|
| 5/11/2015 | Dr. Alan R. Kallmeyer |
| Date | Department Chair |

## ABSTRACT

The autonomous control of unmanned ground vehicles (UGVs) is a growing research area. Skid steered UGVs are desired because of their simple control inputs, however the control algorithm requires complex dynamic analysis. The dynamic model is required to properly implement the control algorithm and this paper presents a linearized model for use in optimal and robust linear control methods. For localization of the robot sensors are required and for many applications low cost sensors are desired. This study used low cost sensors which require proper handling because noise is often increased in lower cost sensors. This study investigated the use of Kalman filtering and fusion on low cost sensors along with a novel approach of satellite selection for improved GPS precision. The sensor information from the Kalman filter was then used in a robust control algorithm and the vehicle's path tracking ability was tested.

## ACKNOWLEDGMENTS

I would like to thank Dr. Majura Selekwa for all the time and effort you spent teaching and guiding me through my studies. I would also like to thank my committee members, Dr. Jacob Glower, Dr. Yechun Wang, and Dr. Mariusz Ziejewski for the time you spent guiding me in this project as well and the time you spent reviewing my work.

I would also like to thank the Bison Robotics Team for helping me with last minute repairs on the robot and being available to support my project. I would like to thank David Sundquist as well for beautifying my robot and Karl Klindworth for being willing to bravely act as a target.

Finally I would like to thank all my friends and family for being a part of making me what I am today.

# TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

## LIST OF APPENDIX FIGURES

# CHAPTER 1. INTRODUCTION

Around the northern half of the United States many areas get large amounts of snow fall each year. On average, Fargo, North Dakota receives 40.0 inches a year [1]. This snow can be annoying and time consuming for the average person to move. Not only is the snow a nuisance for most people to remove, it can also extremely difficult or even impossible for the elderly or disabled people to remove without assistance. Removing snow can also cause injury when people over exert themselves trying to clear snow as quickly as possible. In the United States in 2001, over 54,000 people were treated for injuries related to manual snow removal [2]. Snow removal services are available but they are often costly. Not only is private snow removal expensive, it also can be difficult to find a service that removes the snow in a timely manner. One solution to all these problems is to create an autonomous snowplow to remove snow without human assistance. An autonomous snowplow needs to be mobile and there are many problems associated with navigating a ground vehicle, especially on uncertain surfaces that may be covered in ice or may be dry pavement. Although ground vehicles can be tracked or wheeled, there are many applications where wheeled ground robotic vehicles are required, for example in certain mining operations and in agricultural operations. Wheeled ground robotic vehicles appear in many forms, differing on how they generate traction and how they maneuver to negotiate turns; for example there are skid steered four wheel vehicles, differentially steered three or four wheel vehicles, independently steered four wheel vehicles, and Ackerman steered four or three wheel vehicles, just to name a few. The performance of any wheeled ground robotic vehicle depends heavily on its navigation control system, however, most studies on control of such robotic vehicles have been limited on the vehicle kinematics only, which can be sufficient in applications involving low robot speeds, and in cases where the negative effects of friction can be ignored. This research focused mainly on the modelling and control of a four wheeled skid steered ground robotic vehicle for which both kinematics and dynamics are considered to better model the vehicle on high friction surfaces.

## 1.1. Background

Wheeled ground robotic vehicles have appeared more frequently in recent applications, and easy controllability of these vehicles has become a key design factor. Development of control algorithms for these and any other automated systems requires the presence of a good dynamic model that predicts the system's behavior. The control is implemented by using sensors that measure parameters representing the actual behavior of the system, which are then fed back to the system by the control algorithm. Since most natural systems are nonlinear in nature, developing a mathematical model that captures the behavior of any dynamic system has its own unique challenges because it requires the modeler to decide on what behaviors to capture and what behaviors to ignore. While ignoring certain dynamic behaviors may not affect the system performance, there are some behaviors that must not be ignored. Most existing models for skid steered vehicles assume that friction is not very high such that lateral skidding is possible so the vehicle is able to maneuver turns. Because of this assumption, many models are based entirely on the vehicle kinematics [3]–[7], or include dynamics that allow lateral skidding [8]–[10], which has so far been sufficient on low friction surfaces. However, when friction becomes high enough to prevent lateral skidding, such models fail to capture the dynamics and the vehicle fails to maneuver short radius turns. Inclusion of the effects of friction in these models has been difficult because the friction coefficients are unknown and are non-uniform as noted in [11], [12].

Even when the available vehicle model is sufficient, implementation the control system on such vehicles requires good sensors to provide localization data, particularly, position, velocity, and direction. Although very good sensors exist that can measure and provide such data very accurately, they tend to be very expensive and overwhelm the overall cost of the vehicle itself. Most low cost sensors suitable for low cost robotic vehicles have inherent errors that must be properly handled for proper functioning of the system. Localization sensors for ground robotic vehicles tend to be a combination of Global Positioning System (GPS) receivers, wheel encoders, a distance detection laser scanner, and Inertial Measurement Units (IMU) which typically are made of 3-D accelerometers and 3-D gyroscopes. While errors in the distance detection laser scanners and wheel encoders are easy to manage, errors in the GPS and IMUs are very difficult to manage and quite often very expensive infrastructure is used to address the errors. Alternatively, several sensors can be deployed and their results can be filtered and fused together to reduce errors. Various sensor fusion algorithms exist, the popular ones are derivatives of the Kalman filter. Unfortunately, such fusion algorithms also require the dynamic model of the robotic vehicle to be known, which goes back to the modelling problem mentioned above.

## 1.2. Goals and Objectives

The goal of this research is to improve the control of autonomous four wheel skid steered vehicles, especially on high friction surfaces by disallowing lateral skidding in the dynamic model. As explained above, the central problem that must be addressed is that of developing a good model that captures the dynamics of the vehicle on such surfaces. That model is required by both the controller and the sensor fusion algorithms. Among factors that contribute to the difficulty in controlling four wheel skid steered vehicles when friction is very high include lack of knowledge about the friction level and invalidation of the kinematic assumptions on such surfaces. As if those factors are not enough, the presence of noise and inaccuracies in sensor measurements required for localization and feedback control of the robot makes the problem even worse. This research addresses these problems systematically by first developing a vehicle model that combines both vehicle kinetics and kinematics so that friction forces are made part of the vehicle model. Using this model, the research proceeds by developing methods of reducing the effects of sensor noise especially those from the IMU and the GPS. To achieve the goals of the research there are four main objectives summarized in the following statements:

**1.** To develop an accurate mathematical model for capturing the vehicle dynamics on high friction surfaces, which would also work on low friction surfaces

**2.** To characterize reduce errors inherent in all low cost GPS and IMU sensors

**3.** Develop a robust control algorithm that uses GPS, IMU and encoder data to navigate a robot in path tracking mission

**4.** Experimentally validate the developed control on THUNDAR the autonomous snowplow

Successful completion of this research will be indicated by successfully testing the final control algorithm on the test vehicle on high friction surfaces.

# CHAPTER 2. NAVIGATION SENSORS

There are many methods used in locating a robot in relation to the intended course and the surrounding objects. All sensor measurements are characterized by the presence of noise and uncertainties that affect the accuracy and precision of the measured results. To better understand these uncertainties it is important to know the functional principles of these sensors. This chapter will present the navigation sensors for this study. These sensors can be split into two main categories: obstacle detection sensors and localization sensors. How the information from the sensors is fused and filtered together will be discussed in later sections of this report.

## 2.1. Obstacle Detection Sensors

Obstacle detection sensors will be defined here as sensors used to locate objects around the robot. The information about obstacles can then be used to determine if an object is an obstacle and then the path can be adjusted accordingly. There are two main types of obstacle detection sensors considered in this report: laser scanners and ultrasonic scanners. These sensors have two parts: the signal transmitter and the signal receiver. The unit uses the time of flight of the signal to determine the distance to the nearest obstacle where the transmitter emits a signal and the receiver monitors the reflection. The signal is generally a defined series of pulses that the receiver will recognize when it detects the reflected signal. This reflection covers twice the distance from the object to the robot so the distance from the object to the robot becomes half of the distance travelled by the reflected signal. The propagation speed of the signal is normally known therefore, by monitoring the time taken between the transmitter signal and the receiver signal, the distance to the object is determined. One distance measurement is usually insufficient because information about the entire environment is usually required. The sensors can measure a wide area of distances by scanning back and forth to find what objects are in a large field around it.

### 2.1.1. Ultrasonic Time of Flight

Ultrasonic time of flight measurements use ultrasonic signals to send a signal out for time of flight distance detection. They tend to be popular because they are lower cost than the laser scanners. As summarized in [13], ultrasonic sensors send a pulse or string of pulses at a specific frequency, normally above 40kHz, and wait for a response at that frequency. To obtain a range of measurements around the robot multiple sensors are typically required. This is because the time of flight is relatively slow and rotating the sensor back and forth is not very efficient or effective. In [13], they also point out that ultrasonic range finders are typically more susceptible to environmental changes such as changes in temperature or humidity because the wave propagation speed of sound is greatly affected by these parameters. Because of these drawbacks, these sensors were not used in this study despite their affordability.

### 2.1.2. Laser Time of Flight

Laser time of flight measurements are more expensive than the ultrasonic sensors but can be more versatile and more robust. The time of flight works on the same principle as the ultrasonic scanner but instead of pulses being sent and received using ultrasonic signals, they are sent and monitored using a laser transmitter and receiver. Because of the increased speed in wave propagation, the time of flight is much shorter. This makes a 2D or 3D scanning measurement more practical. A typical scanning pattern can be seen below.

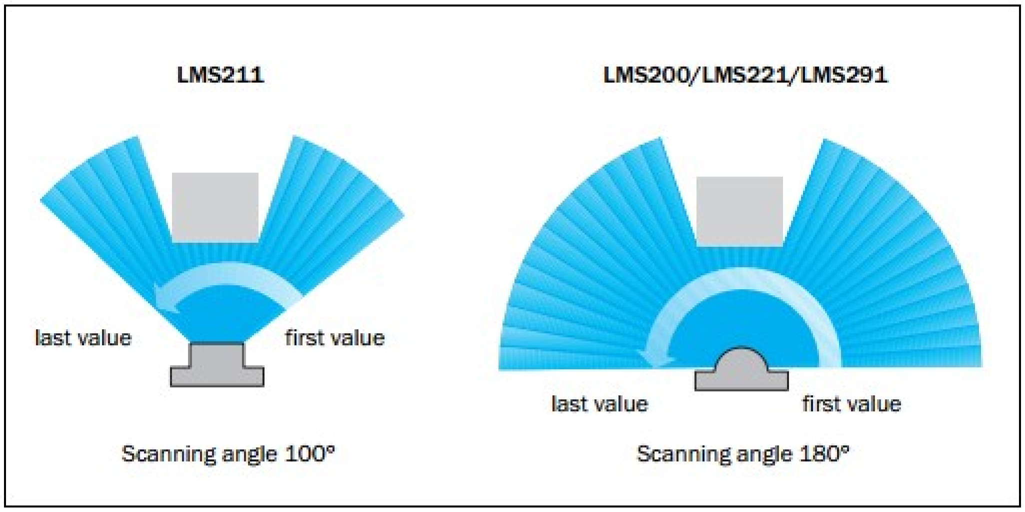


Figure 2.1: Typical scanning pattern [14]

Although there are scanners that can scan 360 degrees, most laser scanners are designed for scanning only up to 180 degrees because often times robot navigation concentrates moving forward and information about what is in front of the robot is sufficient. When scanning, the sensor will measure the distance at predefined angular instances and create a map of measurements. This is very useful in obstacle avoidance in mobile robots. The obstacle avoidance sensor used in this study was a SICK laser scanner.

## 2.2. Localization Sensors

Localization sensors will be defined here as any sensor that locates the robot within a predefined reference frame. They are also used in tracking the actual path versus the intended path and this data is vital in the control algorithm used for path tracking. These sensors can be split into external localization sensors and internal localization sensors. Typical external localization sensors include the global positioning system (GPS) which uses external satellites to calculate the receiver's overall position in relation to the planned path. There are also options for local external localization systems that use beacons or markers to locate the robot such as the Stargazer robot localization sensor [15]. Internal localization sensors are entirely contained by the robot. Typical internal localization sensors include angular displacement sensors that monitor wheel rotation, accelerometers and gyroscopes. Since these sensors appear in many forms the following subsection will discuss only those that were used in the research.

### 2.2.1. Global Positioning System

Global positioning systems have seen a large increase in use in recent years, largely due to recent improvements in precision and accuracy of the GPS receivers. Originally GPS use was restricted to the military. That changed in 1984 when a commercial airplane was shot down after it crossed into restricted airspace due to poor navigation. This prompted the president of the United States at the time, Ronald Reagan, to authorize the use of a segment of the GPS satellites for civilian use in the United States under a Selective Availability policy [16]. As technology increased, methods such as improved filtering and differential GPS techniques were developed, which led to increased accuracy in GPS units and eventually in the year 2000 the Selective Availability policy was abolished [17].

#### 2.2.1.1. Global Positioning System Architecture

The three main parts to the GPS are the space segment, the control segment and the user segment [18] as seen in Figure 2.2 below. The space segment consists of the set of satellites, as seen in Figure 2.3, which orbit the earth along predetermined paths. There are many different sets of satellites from the different global navigation services available such as NAVSTAR (GPS), GLONASS, Galileo and Beidou.

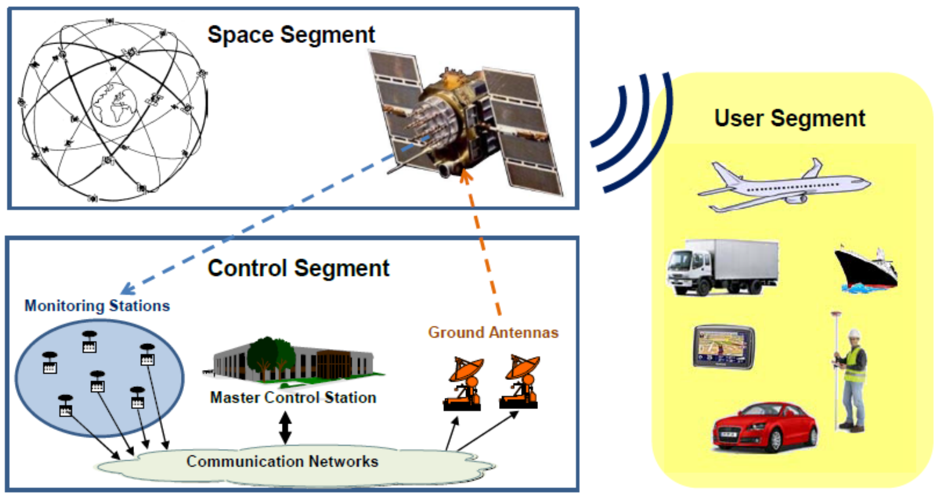


Figure 2.2: Architecture of global navigation satellite system [18]

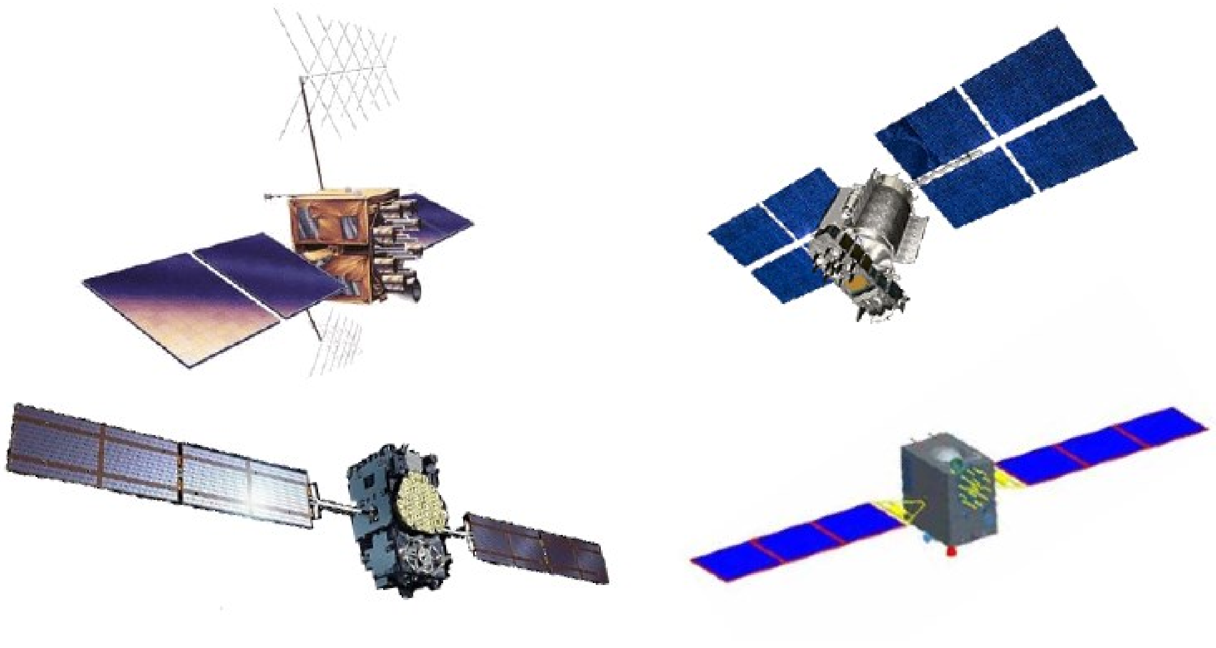


Figure 2.3: Navigation satellites [18]

These satellites beam information that is used by the receiver to calculate the satellite position which is then used to trilaterate the receiver location. The second portion is the control segment. This segment consists of multiple control sensors and antennas which track the position of the satellites, try and predict future atmospheric conditions and to update the satellites with accurate information. Finally the user segment, which is the receiver, takes the information beamed by the satellites and uses that information to calculate the receiver position. There are four types of information packets beamed by the satellite: Almanac, ephemeris, ionospheric and Universal Coordinated Time (UTC). The uses of each packet are summarized in Table 2.1.

Table 2.1: GPS data packets and their uses

| Packet | Function |
|---|---|
| **Almanac** | • Satellite health<br>• Reduced precision UTC and Ephemeris data |
| **Ephemeris** | • Calculate satellite positions |
| **Ionospheric** | • Used in calculating the delay from the ionosphere |
| **UTC** | • Converts satellite time to UTC time |

### 2.2.1.2. Basic GPS Trilateration Methodology

The trilateration problem is solved on the basic level by finding the solution to a set of four equations based on information from four satellites. The earth centered earth fixed (ECEF) Cartesian coordinates of each satellite can be calculated from the data sent to the receiver. These calculation will be outlined in a later chapter. The time the satellite sent the information to the receiver is also known. With the coordinates and time information from four satellites known, the unknown coordinates and time for the receiver can be found. There are four unknown variables in the problem which are the *X*, *Y*, and *Z* coordinates of the receiver and the time delay, $\Delta T$, of the receiver. The distance between each satellite and the receiver, $R_S$, can be found in two different ways. The first is by using the Pythagorean Theorem to find the square distance between the two objects as

$$R_S^2 = (X_S - X)^2 + (Y_S - Y)^2 + (Z_S - Z)^2 \tag{2.1}$$

The second is found by multiplying the time difference of when the satellite sends the information and when the receiver obtains the information by the speed of light, *c*, as

$$R_S = c\left( t_{GPS_S} - t_S - \Delta T\right) \tag{2.2}$$

These two equations can now be formed into the main function as

$$(X_S - X)^2 + (Y_S - Y)^2 + (Z_S - Z)^2 - c^2\left( t_{GPS_S} - t_S - \Delta T\right)^2 = 0 \qquad (2.3)$$

By forming equation (2.3) four times from four satellites, a set of four equations is formed which can be used to solve for the receiver's position. Further detail in specific calculations for the satellite coordinates and the time difference will be shown in Chapter 4.

### 2.2.1.3. Inherent Trilateration Errors

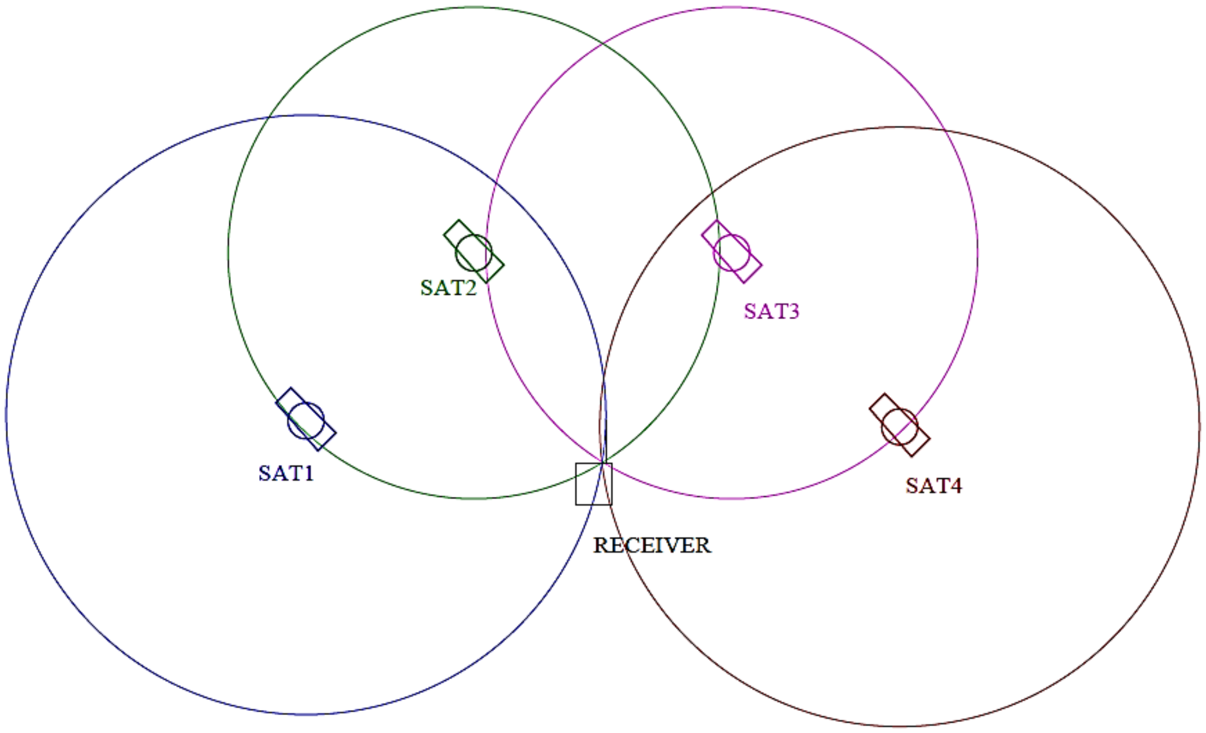


Figure 2.4: Consistent intersection of equations with ideal signals

Ideally, the set of equations described in the previous section have one unique solution and the receiver's ECEF coordinates could be calculated exactly as depicted in Figure 2.4. However, this is not the case and is mainly due to the assumption that the receiver obtains the information from all the satellites at the same time. This is usually caused by delays in the satellite signals which originate from two main sources. The first is through multipath error. This occurs when the satellite signal bounces off objects, such as buildings, on its way to the receiver.

This causes a delay in the reception time and it causes an increase in the satellite's perceived position in space. This will make the calculations shown above inaccurate. The multipath error has been studied and efforts to reduce the error can be seen in [19]–[21]. The other form of interference comes from the ionosphere where ionospheric particles slow down and reduce the intensity of the GPS signals. The strength of the interference is randomly effected by the sun's radiation leading to random signal dilution. Researchers have sought to decrease this interference with improved filtering as seen in [22], [23].

If these errors were consistent across all satellites the set of equations would still be consistent. The solution would be incorrect, as seen in Figure 2.5.

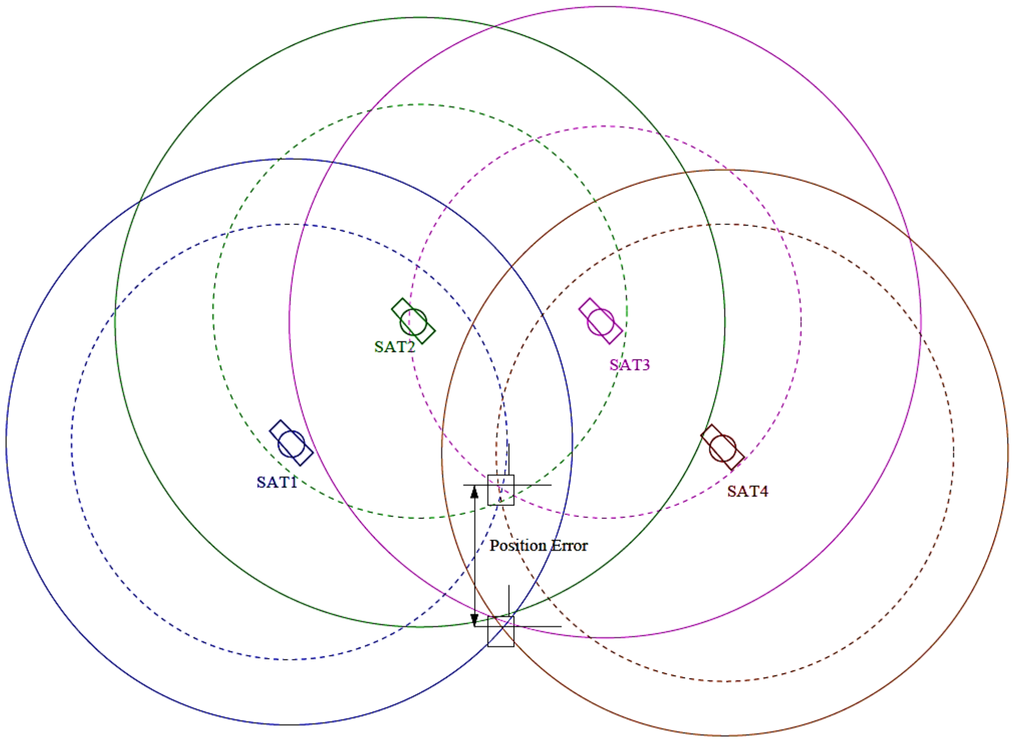


Figure 2.5: Trilateration equations receiving equal interference

This tracking error could be calibrated out in a control system. This, however, is not the case and each satellite receives different amounts of interference. This creates an inconsistent set of equations as illustrated below.

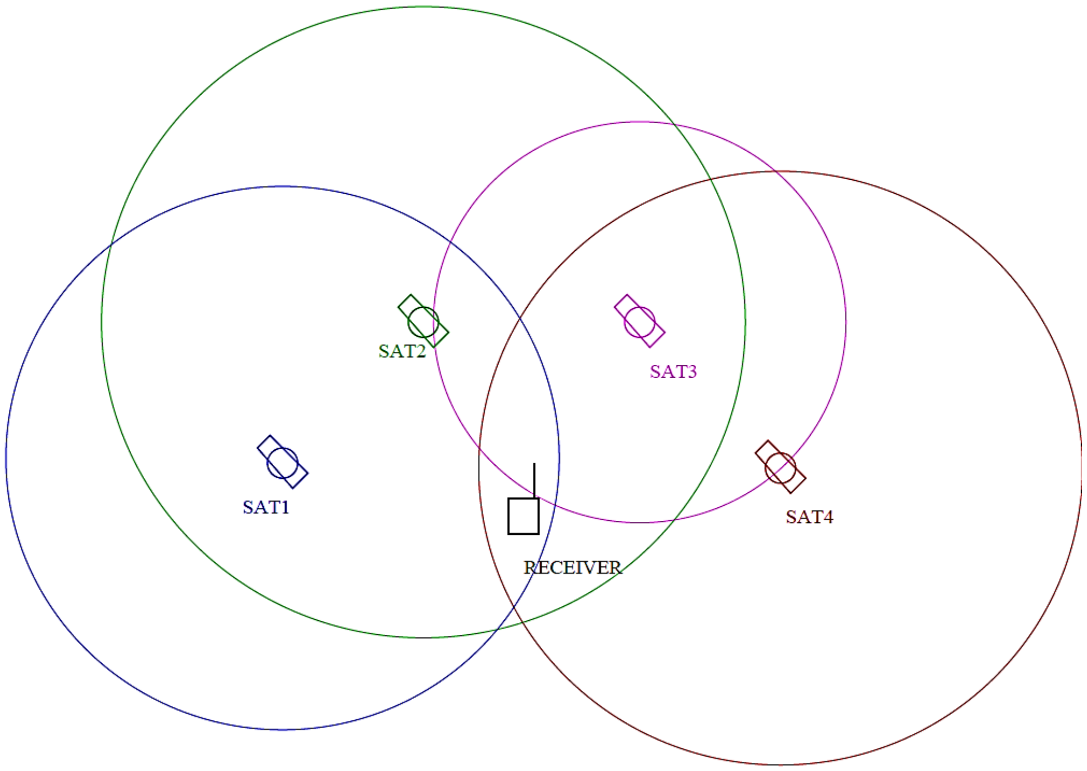


Figure 2.6: Inconsistent set of equations

This means that the solution of the set of equations needs to be estimated by using estimation methods such as the least squares solution or a quasi-Newton estimation method such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. The error estimation methods used in this study will be discussed in Chapter 4.

Traditional methods for solving the estimation uses a set of the strongest satellites with a least squares estimation to calculate the position of the receiver. This method can lead to precision loss in two ways. This first way is by constantly switching the set of equations being used for the position estimation. This happens when the set of strong satellites deteriorates in signal strength because of changing atmospheric conditions. When the satellites become weaker, stronger satellites replace the original satellites which leads to a different position estimation.

This phenomenon is illustrated in Figure 2.7 below.

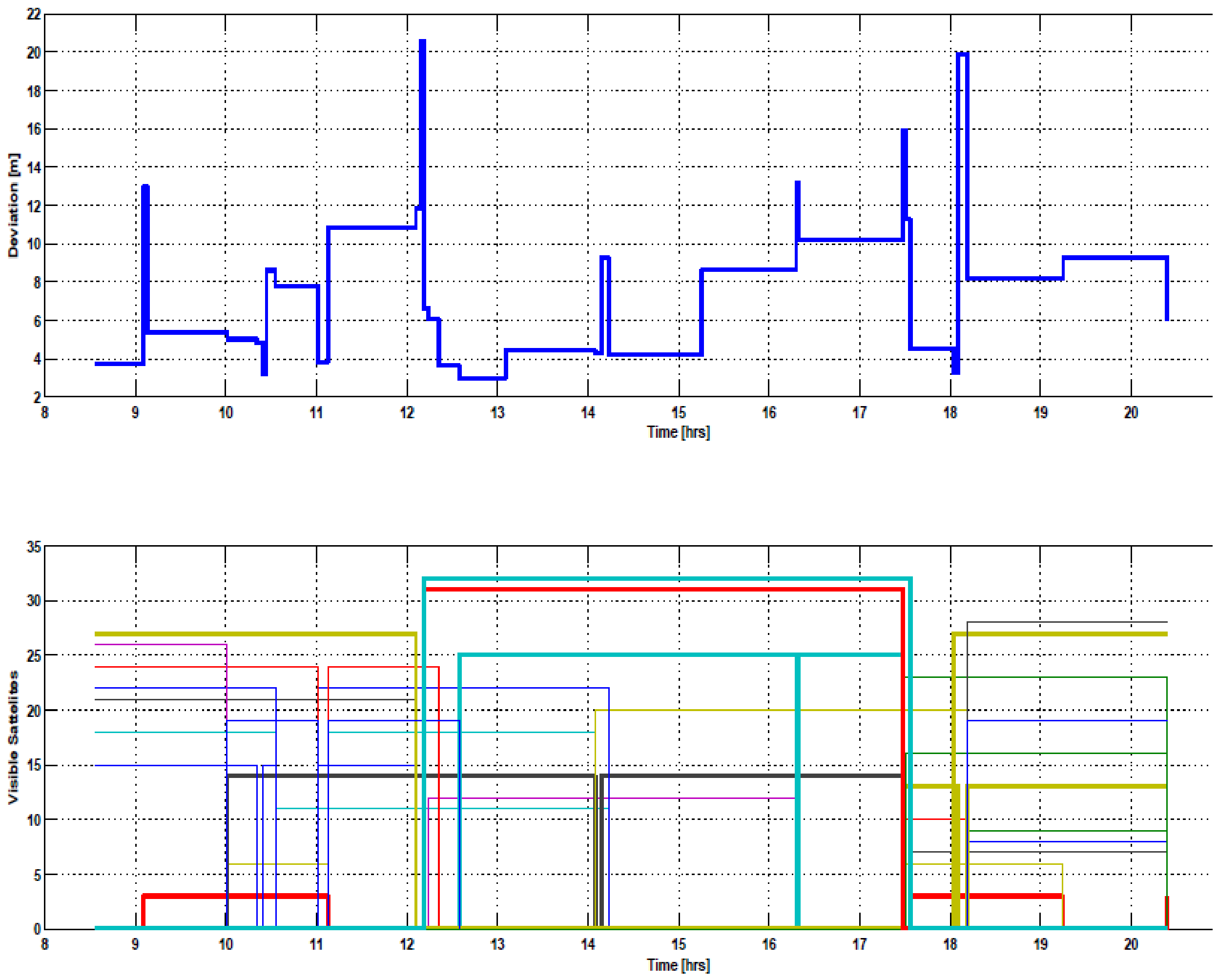


Figure 2.7: Overall position error (top) compared to satellites used in calculations

For this figure, a preliminary study was conducted to see how the satellites used in calculations effected the position fix. For this figure, satellite data was taken over a course of a day and the overall position error was compared to the satellites being used in the position calculation. Spikes in the position error, such as those at times of approximately 9 hours, 12 hours and 18 hours, correspond to new satellites being introduced to the calculation of the position as seen in the bottom half of the graph.

The precision error can also come from a mix of signal strengths being used. The set of equations used for a position fix is closest to being consistent when the signals receive similar amounts of interference which is indicated by similar signal strengths. Choosing one strong signal and three weak signals for a position fix can throw off the position estimation because the signals are far from being consistent. In the preliminary study [24], signal strengths were also monitored on a scale of 0 to 10, 10 being the strongest signals. The results of that study can be seen in Figure 2.8. In the graph, it can be seen that the majority of the signals fall in the signal strength range of 2 to 4. This means that when the strongest signals are picked for the calculations, there is likely one strong signal making the set of equations inconsistent. The improvement of precision by improved satellite selection will be discussed in more detail in Chapter 4.

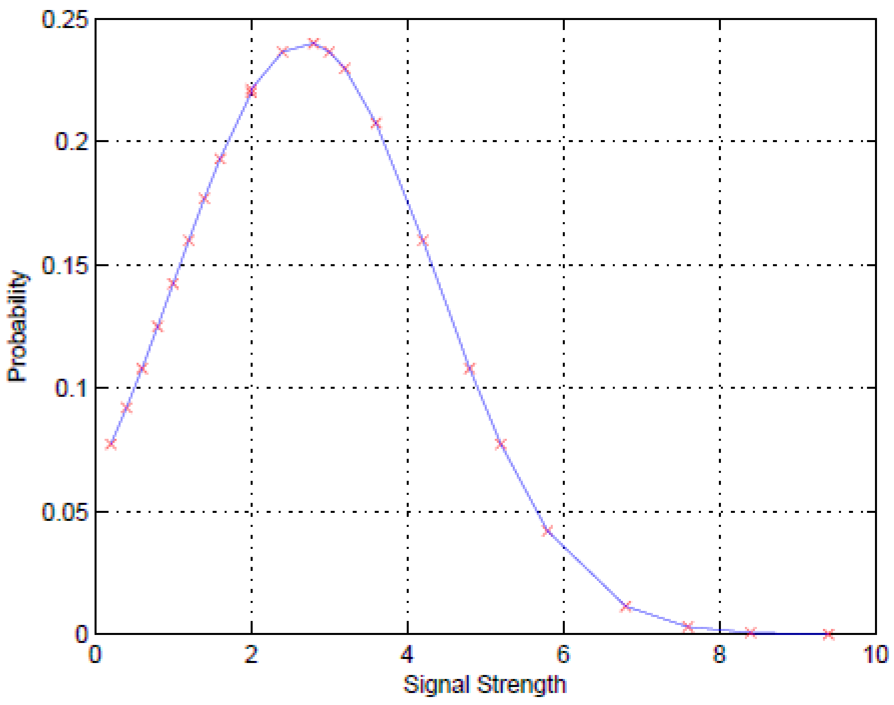


Figure 2.8: Signal strength distribution

### 2.2.2. Relative Positioning Sensors

Relative positioning sensors find the robot's position based upon an onboard transducer sensitive to the robot's movement. The most common structure of relative positioning sensors combine some kind of  odometry and orientation measurement system, and, typically, they are made of tachometers and  gyroscopes as discussed in [25], [26], although some use electronic compasses instead of gyroscopes. Alternatively, gyroscopes are coupled with accelerometers in units known as inertial measurement units (IMU) that may be used in parallel with wheel encoders. There are many varieties of each sensor differing in price ranges and performance characteristics.

#### 2.2.2.1. Inertial Measurement Units

Inertial measurement units monitor the changes in momentum of a system and the information from these sensors can be integrated to find the positions and velocities of the robot. Their measurements are based on the movement of some sort of suspended mass, known as a proof mass, within a moving reference frame. Typically, inertial measurement units consist of an accelerometer and a gyroscope to measure the linear acceleration and rotational velocity along one, two or three axes. Both the gyroscope and the accelerometer started as mechanical devices that shifted to electromechanical devices as technology increased. The advent of micro-electromechanical systems (MEMS), allowed for these devices to be miniaturized in solid state structures that are inexpensive, widely available, and suitable for robotic applications. The typical size and structure of MEMS sensors can be seen in Figure 2.9.

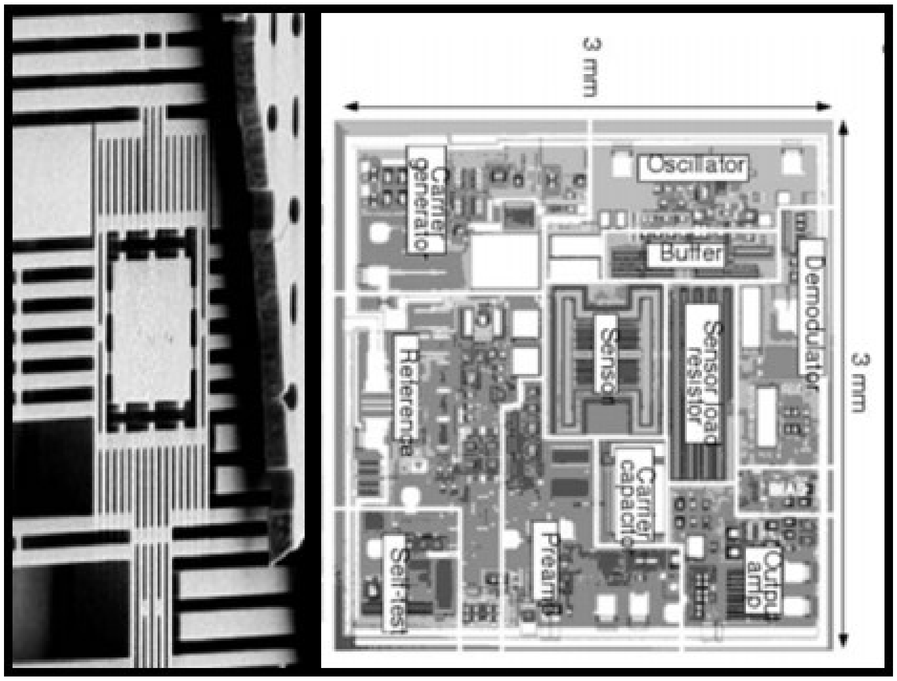


Figure 2.9: MEMS yaw rate sensor (left) and accelerometer (right) [27]

Accelerometers measure the linear acceleration along an axis. The traditional mechanical model that the MEMS equivalents are modeled after is shown below.

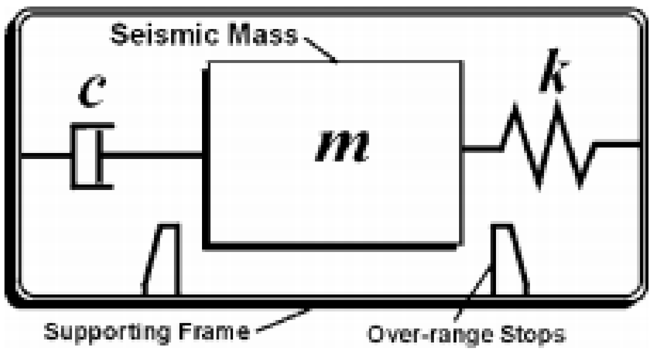


Figure 2.10: Mechanical accelerometer model [28]

Here the proof mass is suspended within the frame by a spring and dashpot. The force to move the mass can be calculated from the displacement of the spring and Hooke's law. That force can then be used to find the acceleration using Newtonian physics. One method to extend the accelerometer to MEMS technology is by measuring the capacitance between plates like the system pictured in Figure 2.11.

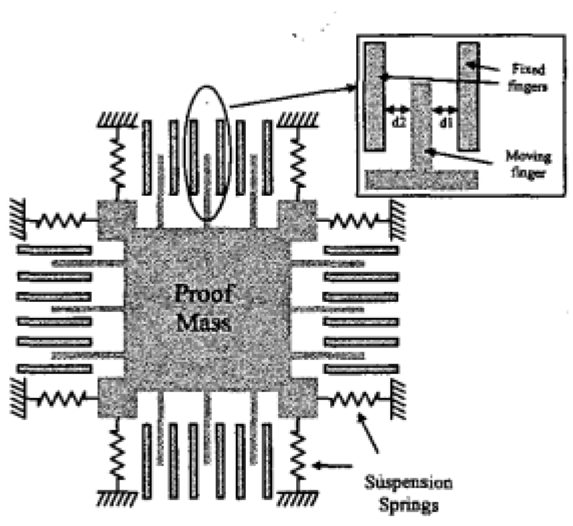


Figure 2.11: MEMS accelerometer [29]

As the robot accelerates, the proof mass moves which changes the distance between the plates. This distance can be calculated because the cross sectional area of the plates are known and fixed. The distance between the plates can then be related to the acceleration like the mechanical accelerometer.

Traditionally gyroscopes were mechanical systems made of a set of rotating masses (flywheels) whose angular momentum was monitored; a diagram of a typical mechanical gyroscope is shown in Figure 2.12. Although the measurement is useful, the flywheel style gyroscope are impractical for robotic applications. Improvements were made to the mechanical gyroscope and instead of a spinning wheel the vibrating mass technique was developed.

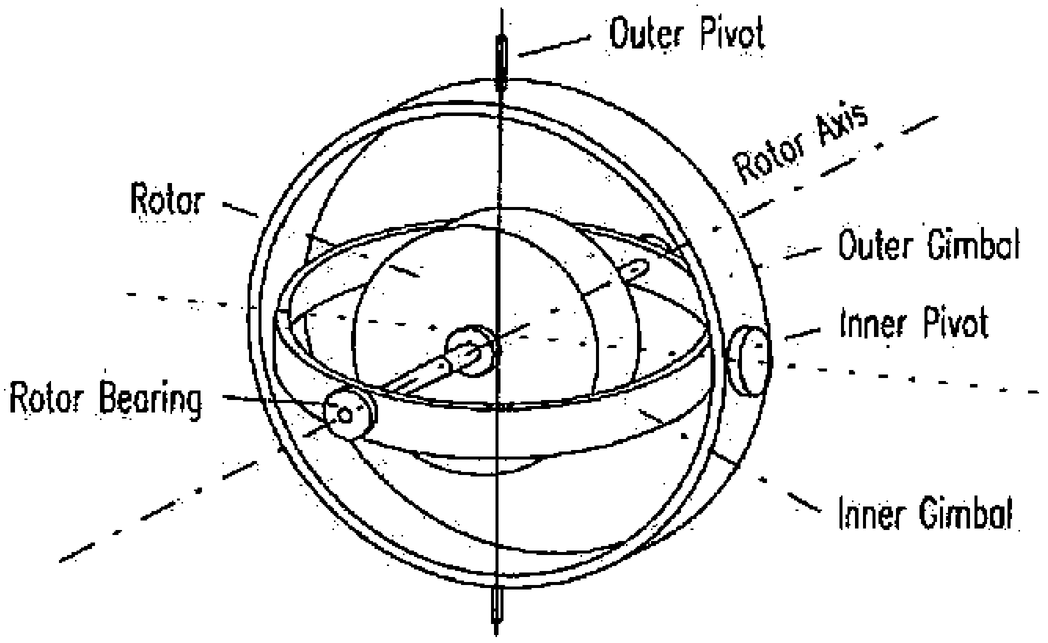


Figure 2.12: Flywheel gyroscope [13]

All vibrating mass gyroscopes are dependent on the Coriolis Effect which describes the forces experienced by a rotating mass. The Coriolis forces run perpendicular to the mass's axis of rotation. This force can be found by monitoring the displacement of a vibrating mass. The suspension of the proof mass in a vibrating mass gyroscope, seen in Figure 2.13, is more complex than that of a MEMS accelerometer.

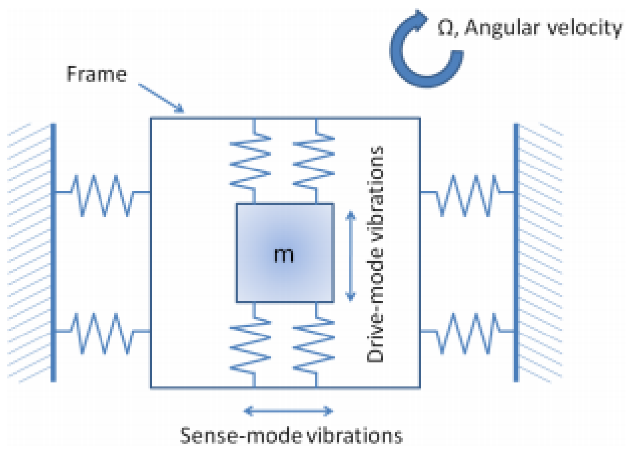


Figure 2.13: Vibrating mass gyroscope [30]

In the gyroscope, a vibrating proof mass is suspended within a frame that is suspended on the body of the sensor. The movement of the middle frame in relation to the body in a direction perpendicular to the angular velocity is proportional to the Coriolis force which is proportional to the angular velocity of the sensor body. This movement can be monitored in a similar fashion to the MEMS accelerometer. This method of monitoring angular acceleration by using a vibrating mass is known as a tuning fork gyroscope and it is one of the more popular MEMS gyroscopes.

Many MEMS gyroscopes tend to be low cost but are also very noisy and very dependent on changes in temperature as shown in [31]. Many high precision applications tend to prefer using the higher precision, higher price optical gyroscopes. The first practical optical gyroscopes were developed in [32]. They are made of two beams of light traveling in opposite directions through a circular tube. As they interact with each other they cause interference waves, seen in Figure 2.14.

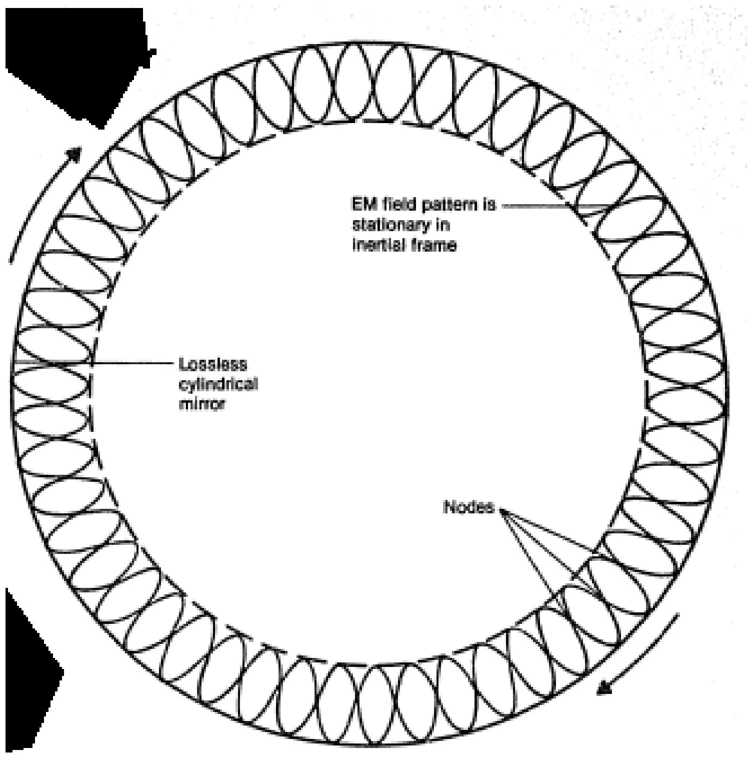

EM field pattern is stationary in inertial frame
Lossless cylindrical mirror
Nodes

Figure 2.14: Interference waves in an optical gyroscope [32]

The variance in the pattern, known as the Sagnac effect, was studied by Georges Sagnac. Sagnac noted that the interference pattern in a loop depends on the angle between the plane of the closed loop and the angular velocity of the loop itself [33]. This concept has been improved upon from the original four mirror design as described in [32], to fiber optic gyroscopes that were first developed in [34]. These gyroscopes are very accurate but are also very expensive.

**2.2.2.2. Optical Encoders**

Odometry sensors are used to track how many times the wheels have turned around. This information is used to then determine the speed and distance covered by the robot through vehicle kinematics. In the case of differential steering, odometry sensors can also be used track the heading angle of the robot. As shown in [25], [26], odometry sensors accumulate errors over time because of calibration errors. Even a well calibrated system will accumulate errors from slight irregularities in the system such as wheels being slightly out of balance or slightly different sizes. Along with the irregularities that accumulate error, odometry sensors also lose accuracy at other times like when the wheels slip or when the robot operates on uneven surfaces.

There many odometry sensors but the most popular sensors are optical encoders. A typical optical encoder layout can be seen in Figure 2.15. The main components are the LED, the disc and the photo detector. The LED shoots a constant beam of light directly at the photo detector. The disc has a series of slots which, when rotated, cause the photo detector to read a sequence of high and low signals which are then decoded to how far the axis has revolved. Optical encoders tend to be very reliable, however they have some inherent errors caused by either skipping the count if the wheel spins too fast, or the optical disk being dirty and the light is blocked for a longer interval.

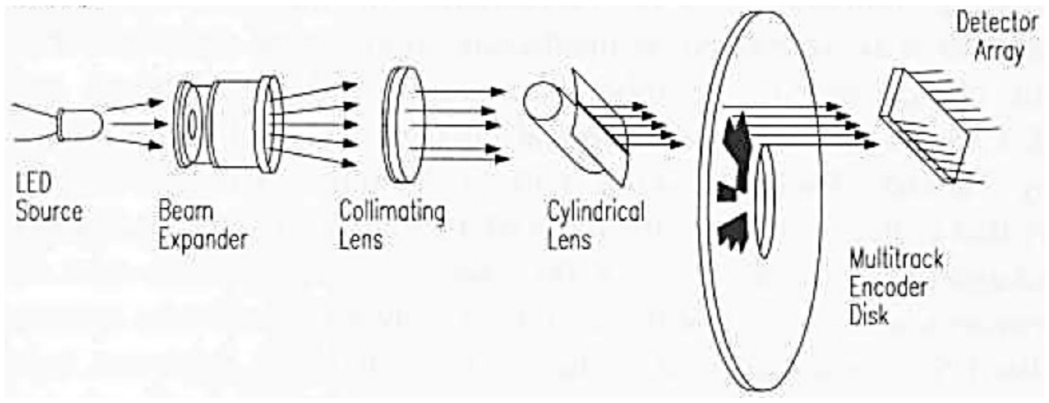


Figure 2.15: Optical encoder layout [13]

# CHAPTER 3. DEVELOPMENT OF A DYNAMIC MODEL

This chapter starts with a background in different vehicle steering methods and provides the motivation for choosing four wheel skid steering. Included in that motivation is a description of the difficulties in controlling skid steering. The second half of this chapter is devoted to the development of the dynamic model for the vehicle.

## 3.1. Methods for Steering Unmanned Ground Vehicles

Although this study focuses on four wheel skid steering, there are many ways to steer a ground vehicle along a desired path. Below is a brief overview of some of the main steering techniques seen in the literature. This helps explain the motivation behind studying four wheeled skid steering and some of the difficulties associated with the study.

### 3.1.1. Actuated Wheel Angle

One of the more traditional steering methods is to use steering similar to what is seen on most cars where actuators direct the angle of the steered wheels. These steering systems have mixed structures; they can have front wheel linked steering, also known as Ackerman steering, rear wheel steering or all wheel steering. For instance, the structure illustrated on left in Figure 3.1 steers either just the front or just the rear wheels and the drive wheels can be either in the front, rear or on all wheels. This steering has been studied in [35], [36]. The vehicle steering angles are configured such that at any time, the vehicle, as a rigid body, must have a unique instantaneous center of rotation (ICR). To meet the wheel speed requirements for the ICR, typically either an active or passive speed differential system is included in both the fixed and steered wheels to prevent wheel slipping during turning maneuvers.

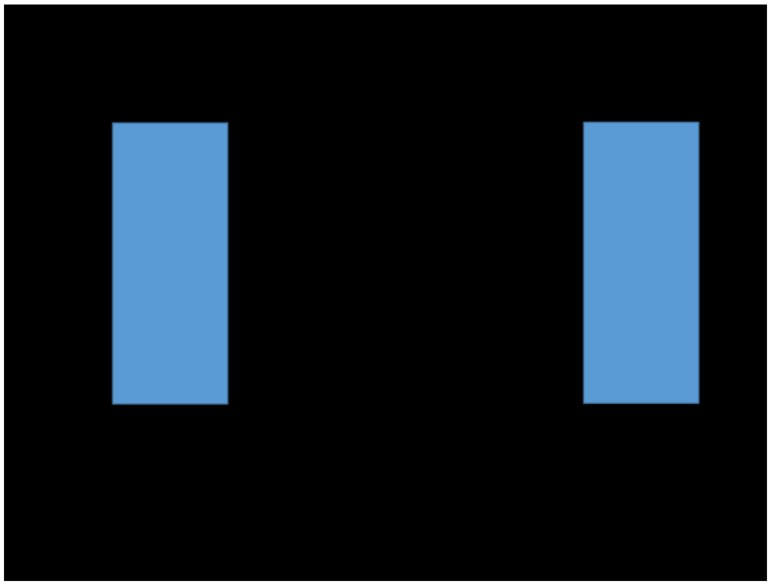


Figure 3.1: Two wheel actuated steering (left) and four wheel actuated steering (right)

Another structure of actuated steering has all four wheels steered each with its own actuated steering angle as illustrated on the right of Figure 3.1 and studied in [37]–[39]. These structures typically use independent steering angles and independent drive speeds for each wheel in order to meet the ICR requirement to prevent wheel slipping during cornering maneuvers. Both the two wheel and four wheel actuated steering structures are typically designed to allow wheels to run entirely on rolling friction without lateral skidding when cornering, therefore, they do not suffer significant uncertainties in their control systems. At high speeds, however, these systems can experience lateral skidding during turning maneuvers and control of high speed skidding in steered vehicles has been studied in [36], [40]. Despite the ability to control the vehicle within pure rolling, these steering systems are not popular among robotic vehicles because of the complexity of the required steering hardware as shown in [11]. The steering angle for the steered wheels must be controlled along with the speed of the vehicle which requires multiple actuators and steering linkages which add to the cost and potential sources of failure and error in the vehicle.

### 3.1.2. Differential Steering

In a differential steering system, a set of coaxial wheels are independently driven which allows for a difference in the traction force on the left and right side of the vehicle as seen in Figure 3.2. This difference causes a turning moment about the robot and allows for steering of the vehicle. Typically, most of these structures have either a caster or set of caster wheels for maintaining the balance of the vehicle; the two driven wheels are responsible for both traction and steering. Since the structure allows the robot to be driven and steered with purely on rolling friction without skidding, it is credited for being the easiest control and different control methods have been studied in [41], [42]. Most light weight ground robotic vehicles are equipped with this steering system. While this approach works well in light weight applications, it is not used on heavier vehicles, which require more traction as noted in [11].

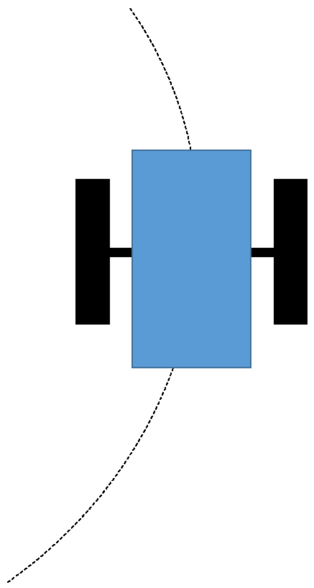


Figure 3.2: Differential steering

### 3.1.3. Skid Steering

More traction can be added to the differential steering design by adding a second set of coaxial wheels or a track system. Tracked systems as illustrated on the right in Figure 3.3 have been studied extensively in [43]. These systems are well studied and mostly for low speed applications and the tracks can be damaging to some surfaces. On the other hand, a skid steered system with a second set of coaxial wheels is less damaging to surfaces and has been attracting more interest in robotic applications. A four wheeled system's weight distribution consists of four point loads, which is different from the two distributed loads in the tracked system; as a result, the analysis of four wheeled skid steered vehicles is different from that of tracked vehicles.

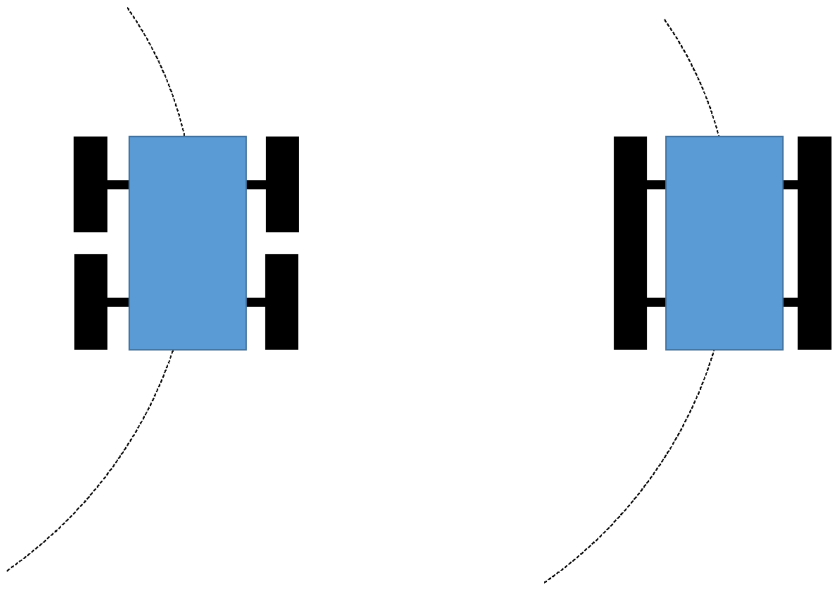


Figure 3.3: Four wheeled skid steering (left) and tracked skid steering (right)

This study will focus on a four wheel skid steered system illustrated on the left in Figure 3.3. The steering mechanisms works the same as differential steering except that it introduces skidding forces that come into play on the second set of coaxial wheels. When the wheels are purely rolling, the vehicle runs in a straight line with no turning motion; therefore there is no instantaneous center of rotation (ICR). To steer, there must be a speed differential between the left and right wheels and this causes skidding in the wheels. Researchers have approached the control of this skidding in different ways. Many studies take the kinematics of the skid steered vehicle and seek to improve the controller as seen in [3]–[7]. Other researchers seek to improve the control through dynamic modeling improvements as seen and supported by either simulations as seen in [8]–[10] or experimentation as seen in [10], [44]. In addition to improved vehicle dynamics researchers also seek to find better models for the interaction between the tire and the operating surface as seen in [45], [46]. Many times the researchers will separate the dynamics from the kinematics by using the dynamics to define constraints on the kinematic controller as seen in [11], [12], [47]. Most of these dynamic models work well in low friction environments where moderate lateral sliding of the vehicle is allowed. In high friction environments, however, lateral sliding does not occur and therefore the models can be improved on by modeling the dynamics without lateral slipping. This modeling approach is seen in [11], however, the dynamics are not fully included in the control model. Part of the goal of this research is to use a dynamic model suitable for high friction environments for the control of a skid steer vehicle which was started in [48].

## 3.2. Actuator Model

Most robotic vehicles are driven by using some kind of an electric motor, which could be a stepper motor, brushless DC motor or permanent magnet DC motor. Because of their relatively high torque capacity, ease of use, and availability, permanent magnet DC motors are very popular in automation. These motors can easily be controlled by using a microprocessor through pulse width modulation (PWM) and thus easily used in a closed loop control system. There are other options available such as pneumatic actuators but these systems are more complex and can be more costly. Also because of the model complexity in pneumatic actuators, they are often more popular in open loop control systems. The actuators used on the robotic vehicle in this study for closed loop control were two brushed DC motors, one on each side wheels of the robot.

Since the robotic vehicle under study was meant for an autonomous snow plow operation, there are two more  DC electric motors in the plow blade system, one linear actuator to control the blade angle of the snow plow and a winch motor to raise and lower the blade. The two wheel drive motors are controlled in a closed loop structure while the plow blade system motors are driven in an open loop control system. The control details of this research are focused on the two wheel drive motors only; the plow blade motors are not considered. The wheel motors are Motenergy ME0708 brushed DC electric motors and like all DC motors are characterized by a torque curve. The specific curve for the Montenergy motors is seen in Figure 3.4.

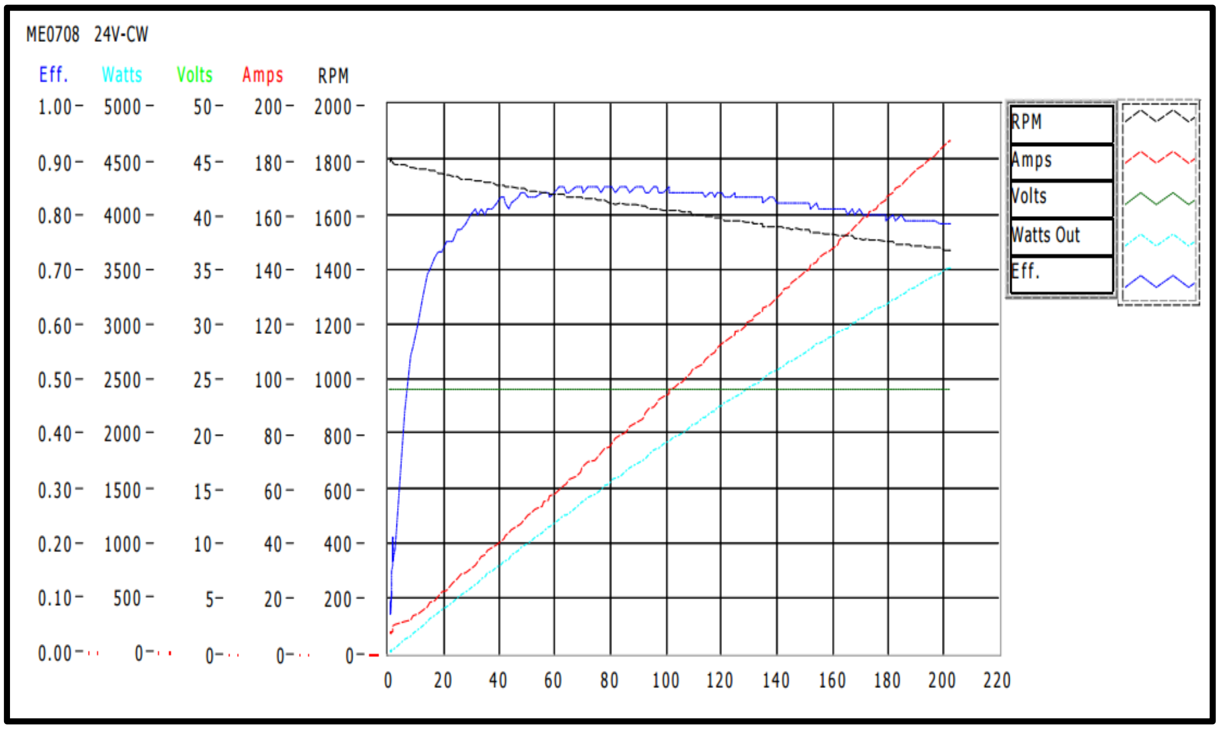


Figure 3.4: Montenergy performance curve at 24 volts [49]

For control of the DC motors it is important to have a good model of the actuator and this study will use a model adapted from [50]. In this model the torque on the motor can be described as

$$T = \frac{k_t}{R_a}\left(V_{DC} - \left(\frac{R_a v}{k_t} + k_b\right)\dot{\theta}\right) \tag{3.1}$$

where $k_t$ is the torque constant, $R_a$ is the motor resistance, $k_b$ is the back Emf constant and $v$ viscous dampening of the motor. In this study the viscous dampening will be ignored because of the lack of information available which will add some uncertainties to the model. This gives us relation of

$$T = \frac{k_t}{R_a}\left(V_{DC} - k_b\dot{\theta}\right) \tag{3.2}$$

to describe the torque of the motor in terms of voltage. It isn't practical to find the torque constant and the motor resistance individually because that information is not always provided by the manufacturer. Rather the ratio of the two terms are found by using the stall torque shown in the performance curves as

$$\frac{T_S}{V} = \frac{k_t}{R_a} \tag{3.3}$$

which can then be used to find the back Emf constant as

$$\dot{\theta}_n = T_s\left(k_b\frac{k_t}{R_a}\right)^{-1} \tag{3.4}$$

where $\dot{\theta}_n$ is the no load angular velocity. This model will be used later in the dynamic model to relate the states of the robot to the control inputs.

One of the main assumptions in this model is that the motor can draw as much torque as it needs to perform the required maneuvers. This however is not the case and all real motors have current limitations as well. It is shown in [50] that current drawn becomes especially excessive when the motor reverses directions, which is a common maneuver in this study. The typical drastic effects of voltage reversal on current draw can be seen in Figure 3.5. The motor direction can quickly change in this robot if a tight turning radius is required and this may affect the performance of the control algorithm.

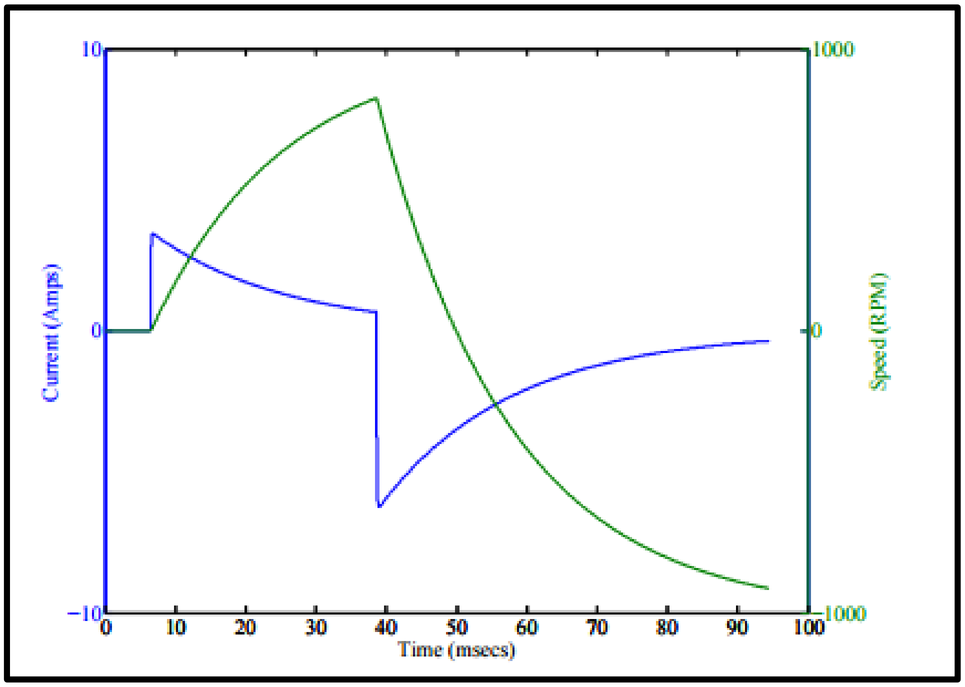


Figure 3.5: Current spike in angular velocity direction change [50]

## 3.3. Kinematic and Dynamic Modeling of Skid Steering

The kinematics of a skid steering vehicle can be very complex. Most of this complexity is introduced when the vehicle turns. This analysis will focus on curved motion because straight line motion is simple.  Turning requires the vehicle's tires to skid which introduces complex non-linear terms. The kinematics can be simplified by making certain basic assumptions that are true for most ground vehicles. The first assumption is that the traction of the robot is purely from the interaction between the ground and wheels under no slip conditions. This traction is assumed to run purely parallel to the vehicle. The second assumption is that the vehicle only turns based on the traction difference between the left and right side of the vehicle. The third main assumption is that lateral slip in the vehicle is ignored. Finally it is assumed the vehicle is non-deformable, operates on flat surface, and has an even weight distribution.

### 3.3.1. Kinematic Model

With those assumptions the vehicle model can be started and a diagram of the vehicle in the global and local coordinate system can be seen below in Figure 3.6.

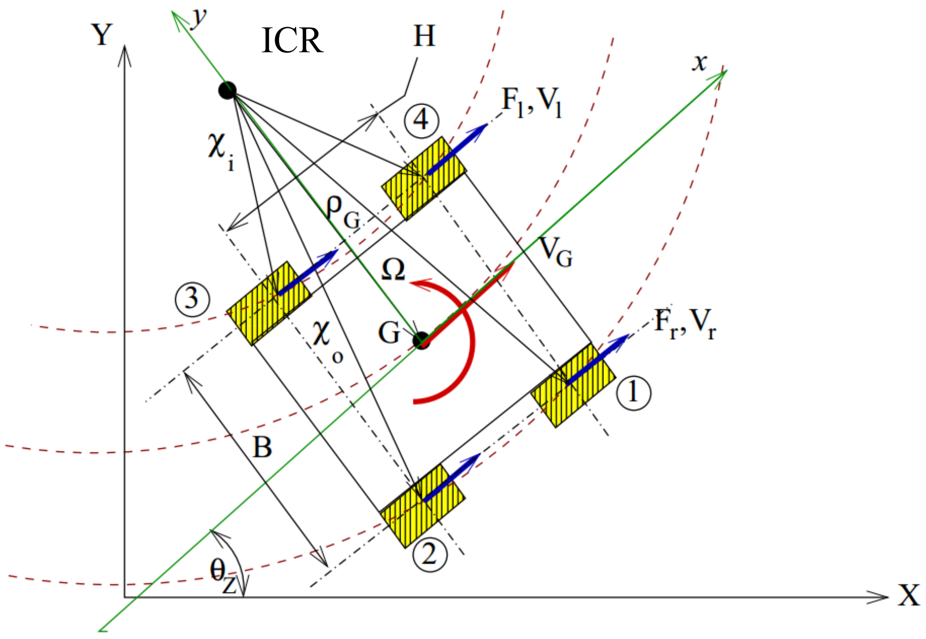


Figure 3.6: Dynamic model of four wheel skid steered vehicle

In the model, the axes (*X*, *Y*, *Z*) represent the global inertial frame and the local inertial frame is defined by the axes (*x*, *y*, *z*). On the vehicle all the wheels have an identical radius, $r_w$. The wheels on the left side of the vehicle, wheels 3 and 4, provide the left side traction, $F_l$, and velocity $V_l$. Similarly the wheels on the right side of the robot, wheels 1 and 2, provide the right side traction, $F_r$, and velocity $V_r$. The velocities for the left and right side of the robot are found as

$$V_r = \omega_r r_w,\ V_l = \omega_l r_w \tag{3.5}$$

where $\omega_l$ and $\omega_r$ are the left and right side wheel velocities respectively. The global heading angle, $\theta_z$, can be derived from the path gradient, $\frac{dY}{dX}$, which is related to the velocities in the global $X$ and $Y$ coordinates as

$$\theta_Z = \arctan\left(\frac{dY}{dX}\right) = arctan\left(\frac{V_Y}{V_X}\right) \tag{3.6}$$

These global velocities, $V_x$ and $V_y$, can also be further defined based on the velocities components in the local frame, $v_x$ and $v_y$, as

$$\begin{pmatrix} V_X \\ V_Y \end{pmatrix} = \begin{pmatrix} v_x \cos(\theta_Z) - v_y \sin(\theta_Z) \\ v_x \sin(\theta_Z) + v_y \cos(\theta_Z) \end{pmatrix} \tag{3.7}$$

Since lateral slip is ignored the local $y$-axis component can be set to zero and then the resultant vehicle velocity, $V_G$, becomes

$$V_G = v_x = \frac{1}{2}(V_r + V_l) \tag{3.8}$$

The angular velocity about the ICR can also be defined by the velocity of the left and right side of the vehicle as

$$\Omega = \frac{d\theta_Z}{dt} = \frac{1}{B}(V_r - V_l) \tag{3.9}$$

By using equations (3.6) through (3.9) the left and right vehicle velocities can be determined based on the desired velocity and path curvature. The selection of the exact heading angle and speed depend on the control algorithm used and that will be discussed in a later section of this paper. Once the left and right vehicle velocities are determined it is necessary to related them to the voltage that needs to be applied to the DC motors by the control system.

To find the wheel torque the path geometry needs to be further defined. With angular vehicle velocity and linear vehicle velocity defined in equations (3.8) and (3.9), the radius of curvature can be defined as

$$\rho_G = \frac{V_G}{\Omega} \tag{3.10}$$

This relation holds true because of the symmetry of the robot and the assumption that the ICR falls along the perpendicular bisector of the robot because of the even weight distribution. The curvature is also useful in deriving further equations and is defined as

$$\kappa_G = \frac{1}{\rho_G} = \frac{2}{B}\left(\frac{V_r - V_l}{V_r + V_l}\right) \tag{3.11}$$

in terms of the left and right vehicle velocities. From the geometry of the vehicle the inner and outer radius of curvatures can be defined as

$$\chi_i = \frac{1}{2\kappa_G}\sqrt{(2 - B\kappa_G)^2 + H^2{\kappa_G}^2} \tag{3.12}$$

$$\chi_o = \frac{1}{2\kappa_G}\sqrt{(2 + B\kappa_G)^2 + H^2{\kappa_G}^2} \tag{3.13}$$

respectively. Here *B* is the width of the vehicle and *H* is the length of the vehicle as shown in Figure 3.6. Since the vehicle is symmetric the analysis for the inner and outer curvatures are identical. A simplified model that represents the inner and outer radius of curvature can be seen below in Figure 3.7.

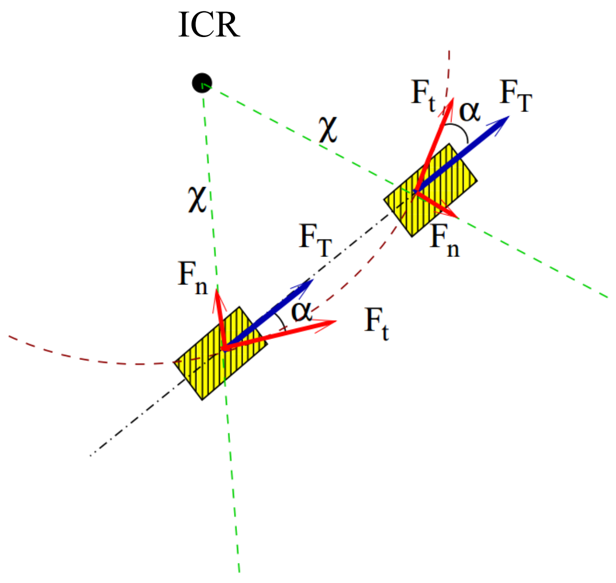


Figure 3.7: Reduced dynamic model

To find the moment about the center of rotation the traction forces, $F_T$, need to be decomposed into their components that lie normal and tangent to the path curvature. This is done by finding the inner and outer slip angles $\alpha_i$ and $\alpha_o$. These are found from the vehicle geometry and the path curvature as

$$\alpha_i = \arctan\left(\frac{H\kappa_G}{2-B\kappa_G}\right) \tag{3.14}$$

$$\alpha_o = \arctan\left(\frac{H\kappa_G}{2+B\kappa_G}\right) \tag{3.15}$$

With those defined the tangential component, $F_t$, and the normal component, $F_n$, can be defined as

$$F_t = F_T\cos(\alpha) \tag{3.16}$$

$$F_n = F_T\sin(\alpha) \tag{3.17}$$

Since the normal force is concurrent at the ICR only the tangential force contributes to the total moment about the ICR. Therefore the total moment about the ICR can be expressed in terms of the left and right tangential traction forces, $F_{l_t}$ and $F_{r_t}$, as

$$M_\Omega = I_{ICR}\dot{\Omega} = 2\left(F_{l_t}\chi_i + F_{r_t}\chi_o\right) - \frac{mg}{2}\left(\cos(\alpha_i)\,\chi_i\mu_i + \cos(\alpha_o)\,\chi_o\mu_o\right) \tag{3.18}$$

Here, the mass moment of inertia about the ICR is represented as $I_{ICR}$. The terms $\mu_i$ and $\mu_o$ are the inner and outer coefficients of friction. These are defined by Pacejka's formula [51] as

$$\mu(\alpha_k) = C_1\sin\left(C_2\tan^{-1}\left(C_3\alpha_k - C_4\left(C_3\alpha_k - \tan^{-1}(C_3\alpha_k)\right)\right)\right) \tag{3.19}$$

This equation is based on the slip angle defined earlier in this paper and road constants defined by the surface conditions, $C_1$ through $C_4$. Although the surface conditions can be characterized by this formula, it is often very hard to find constants for the operating surface, especially as the robot moves and the surface conditions change. Because of the lack of information on the various operating surfaces, the derivations using equation (3.18) will often ignore the forces generated by friction and lump them into an uncertainty term.

### 3.3.2. Dynamic Model

Slip angle constraints are relaxed in this study and then equations (3.12) through (3.19) are used to form

$$I_{ICR}\frac{d\Omega}{dt} = \left(2F_l\cos(\alpha_i) - \frac{mg}{2}\mu(\alpha_i)\right)\chi_i + \left(2F_r\cos(\alpha_o) - \frac{mg}{2}\mu(\alpha_o)\right)\chi_o \tag{3.20}$$

In the previous equation the angular acceleration, $\frac{d\Omega}{dt}$, is found as

$$\frac{d\Omega}{dt} = \frac{1}{B}\left(\frac{dV_r}{dt} - \frac{dV_l}{dt}\right) \tag{3.21}$$

The mass of the robot, *m*, and the acceleration due to gravity, *g*, are both given constants. The equations (3.12) through (3.15) can be written in terms of left and right wheel velocity as

$$\chi_i = \frac{B}{2}\sqrt{\left(\frac{V_l+V_r-\text{sgn}(\Omega)(V_l-V_r)}{\text{sgn}(\Omega)(V_l-V_r)}\right)^2 + \left(\frac{H}{B}\right)^2} \tag{3.22}$$

$$\chi_o = \frac{B}{2}\sqrt{\left(\frac{V_l+V_r+\text{sgn}(\Omega)(V_l-V_r)}{\text{sgn}(\Omega)(V_l-V_r)}\right)^2 + \left(\frac{H}{B}\right)^2} \tag{3.23}$$

$$\alpha_i = \arctan\left(\left(\frac{H}{B}\right)\left(\frac{\text{sgn}(\Omega)(V_l-V_r)}{V_l+V_r-\text{sgn}(\Omega)(V_l-V_r)}\right)\right) \tag{3.24}$$

$$\alpha_o = \arctan\left(\left(\frac{H}{B}\right)\left(\frac{\text{sgn}(\Omega)(V_l-V_r)}{V_l+V_r-\text{sgn}(\Omega)(V_l-V_r)}\right)\right) \tag{3.25}$$

Now the radius of curvature is fully related to the traction force in terms of the left and right wheel velocities by using equations (3.20) through (3.25). It is still necessary to solve for the traction force and relate that to motor torque. For the traction force, it is assumed that the vehicle has an even weight distribution so the traction force, $F_{T(i)}$, of wheel *i* can be related to the angular acceleration of wheel *i*, $\dot{\omega}_{(i)}$, the mass of the robot, *m*, the wheel radius, $r_w$, and the mass of inertia of the wheel about its axis, $I_w$ as

$$F_{T(i)} r_w = \left(I_w + \frac{1}{4} m r_w^2\right) \dot{\omega}_{(i)} \tag{3.26}$$

The traction force for the left and right side of the robot can be derived from equations (3.5) and (3.26) as

$$F_l = F_{T(3)} + F_{T(4)} = 2\left(\frac{I_w}{r_w^2} + \frac{m}{4}\right)\frac{dV_l}{dt} \tag{3.27}$$

$$F_r = F_{T(1)} + F_{T(2)} = 2\left(\frac{I_w}{r_w^2} + \frac{m}{4}\right)\frac{dV_r}{dt} \tag{3.28}$$

The left and right side traction forces can also be related to the wheel torques and ultimately the motor voltages. The individual wheel torque, $T_{(i)}$, is subject to

$$T_{(i)} = F_{T(i)} r_w - T_{loss} \tag{3.29}$$

where $T_{loss}$ accounts for torque losses that are otherwise ignored. The torque on each wheel can also be related to the motor torque, $T_m$ as

$$T_{(i)} = T_m \left(\frac{n_r}{2}\right) \tag{3.30}$$

where $n_r$ is the transmission ratio of the drivetrain. The motor torque is related to the angular velocity of the motor shaft by a motor torque constant, $k_t$, and armature resistance $R_a$ and subject to losses from the back electromagnetic field constant $k_b$. These parameters are provided by the manufacture of the motor in the following relations and they combine with equations (3.2), (3.29) and (3.30) to form

$$F_{T(i)} = \left(\frac{k_t n_r}{2 R_a r_w}\right)\left(V_{DC} - k_b n_r \omega_{(i)}\right) + T_{loss} \tag{3.31}$$

$$F_l = F_{T(3)} + F_{T(4)} = \left(\frac{k_t n_r}{R_a r_w}\right)\left(V_{DC_L} - k_b n_r \omega_l\right) + \Delta_L \tag{3.32}$$

$$F_r = F_{T(1)} + F_{T(2)} = \left(\frac{k_t n_r}{R_a r_w}\right)\left(V_{DC_R} - k_b n_r \omega_r\right) + \Delta_R \tag{3.33}$$

where $\Delta_L$ and $\Delta_R$ lump all the motor and transmission losses on the left and right side respectively. Equations (3.32) and (3.33) can be combined with (3.27) and (3.28) to form the left and right side vehicle accelerations as

$$\frac{dV_l}{dt} = \left(\frac{2k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)V_{DC_L} - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)V_l + \Delta_L \tag{3.34}$$

$$\frac{dV_r}{dt} = \left(\frac{2k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)V_{DC_R} - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)V_r + \Delta_R \tag{3.35}$$

$$\frac{dv_x}{dt} = \frac{1}{2}\left(\frac{dV_r}{dt} + \frac{dV_l}{dt}\right) \tag{3.36}$$

which can be related to the overall vehicle acceleration in equation (3.36) to form

$$\frac{dv_x}{dt} = \left(\frac{k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)(V_{DC_R} + V_{DC_L}) - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)v_x + \Delta_{Loss} \tag{3.37}$$

where all the losses can be summed up as $\Delta_{Loss}$. Along with the individual traction force, this result can be used in the control algorithm.

The angular acceleration is found using equations (3.20) through (3.25). The fully expanded equation yields the following cumbersome nonlinear equation,

$$I_{ICR}\frac{d\Omega}{dt} = \left[\frac{k_t n_r}{R_a r_w}\left(V_{DC_L} - \frac{k_b n_r}{r_w}V_l\right)\sqrt{\frac{1}{1+\left(\frac{H}{B}\right)^2\left(\frac{\operatorname{sgn}(\Omega)(V_l-V_r)}{V_l+V_r-\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2}} - \frac{mg}{2}\mu_i\right] \times \ldots$$

$$\ldots\left[\frac{B}{2}\sqrt{\left(\frac{V_l+V_r-\operatorname{sgn}(\Omega)(V_l-V_r)}{\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2 + \left(\frac{H}{B}\right)^2}\right] + \cdots$$

$$\cdots\left[\frac{k_t n_r}{R_a r_w}\left(V_{DC_R} - \frac{k_b n_r}{r_w}V_r\right)\sqrt{\frac{1}{1+\left(\frac{H}{B}\right)^2\left(\frac{\operatorname{sgn}(\Omega)(V_l-V_r)}{V_l+V_r+\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2}} - \frac{mg}{2}\mu_i\right] \times \ldots$$

$$\left[\frac{B}{2}\sqrt{\left(\frac{V_l+V_r+\operatorname{sgn}(\Omega)(V_l-V_r)}{\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2 + \left(\frac{H}{B}\right)^2}\right] \tag{3.38}$$

This is linearized by noting that the ratio $\frac{H}{B}$ is approximately 1. By assuming that the ratio and all of the radicals associated with the ratio are equal to one the following terms can be approximated as

$$\frac{B}{2}\sqrt{\frac{1}{1+\left(\frac{H}{B}\right)^2\left(\frac{\operatorname{sgn}(\Omega)(V_l-V_r)}{V_l+V_r+\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2}\left(\frac{V_l+V_r-\operatorname{sgn}(\Omega)(V_l-V_r)}{\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2+\left(\frac{H}{B}\right)^2} \approx \frac{2}{\sqrt{2}}\left(\frac{V_0}{\Omega}\right) \tag{3.39}$$

$$\frac{B}{2}\sqrt{\frac{1}{1+\left(\frac{H}{B}\right)^2\left(\frac{\operatorname{sgn}(\Omega)(V_l-V_r)}{V_l+V_r-\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2}\left(\frac{V_l+V_r+\operatorname{sgn}(\Omega)(V_l-V_r)}{\operatorname{sgn}(\Omega)(V_l-V_r)}\right)^2+\left(\frac{H}{B}\right)^2} \approx \frac{2}{\sqrt{2}}\left(\frac{V_0}{\Omega}\right) \tag{3.40}$$

where $V_o$ can be either $V_l$ or $V_r$ depending on $\operatorname{sgn}(\Omega)$. This reduces equation (3.38) to

$$\frac{d\Omega}{dt}=\frac{Bk_tn_r}{2IR_ar_w^2}\left[\left(V_{DC_L}-\frac{k_bn_r}{r_w}V_l\right)-\left(V_{DC_R}-\frac{k_bn_r}{r_w}V_r\right)\right]-\Delta M_\Omega \tag{3.41}$$

where $\Delta M_\Omega$ represents the uncertainties introduced through linearization, $\Delta_e$, and the unknown friction conditions shown in equation .

$$\Delta M_\Omega=\Delta_e+\left[\frac{mg}{2}\mu_i+\frac{mg}{2}\mu_o\right] \tag{3.42}$$

Further simplification leads to

$$\frac{d\Omega}{dt}=\left(\frac{B^2k_bk_tn_r^2}{2IR_ar_w^2}\right)\Omega-\left(\frac{Bk_tn_r}{2IR_ar_w^2}\right)(V_{DC_R}-V_{DC_L})+\Delta M_\Omega \tag{3.43}$$

By using equations (3.34), (3.35), (3.37) and (3.43) the dynamics system is now linearized and can be used in combination with the kinematic equations (3.7) and (3.9) in standard robust and optimal linear control equations. This study will use this new dynamic model to implement some of those control algorithms in later chapters.

# CHAPTER 4. GPS TRILATERATION IMPROVEMENTS

As noted earlier, standard GPS algorithms tend have precision errors that are influenced by the satellite selection algorithms used in calculation of the receiver's position. This study aimed to improve the precision of the position fix by selecting satellite signals of similar strength. Along with this improvement, the study also compared the traditional solution technique of a least-squares algorithm to the BFGS algorithm and Newtonian algorithm. The BFGS and Newtonian algorithms make less approximations and it was hypothesized that this also can improve the precision of the position fix. It is important to note again that this study sought to improve the precision of the position fix, not necessarily the accuracy. The accuracy can be dealt with in a control application while imprecision causes navigation problems. This chapter details the satellite position calculations, the time difference calculations, the estimation algorithms, and the satellite selection algorithm.

## 4.1. Satellite Coordinate Calculation

The calculations for satellite coordinates are based on the ephemeris data and raw time data. The ephemeris data is listed below in Table 4.1 which has been adapted from [18], [52], [53] and [54]. The following derivation is adapted from standard positioning equations found in [18], [52], [53].

Table 4.1: Ephemeris data received from each satellite. *Denotes that data was calculated by the GPS receiver from the standard ephemeris data.

| Parameter | Scale Factor | Units |
|---|---|---|
| **C_rs** | $2^{-5}$ | Meters |
| **M_0** | $2^{-31}$ | Semi-circles |
| **C_uc** | $2^{-29}$ | Radians |
| **e** | $2^{-33}$ | Dimensionless |
| **C_us** | $2^{-29}$ | Radians |
| **sqrt_A** | $2^{-19}$ | $\text{Meters}^{1/2}$ |
| **t_oe** | $2^{4}$ | Seconds |
| **C_ic** | $2^{-29}$ | Radians |
| **OMEGA_0** | $2^{-31}$ | Semi-circles |
| **C_is** | $2^{-29}$ | Radians |
| **i_0** | $2^{-31}$ | Semi-circles |
| **C_rc** | $2^{-5}$ | Meters |
| **OMEGADOT** | $2^{-43}$ | Semi-circles/sec |
| **IDOT** | $2^{-43}$ | Semi-circles/sec |
| **n*** | - | - |
| **r1me2*** | - | - |
| **omega** | - | - |
| $\mathbf{a_{f_0}}$ | - | - |
| $\mathbf{a_{f_1}}$ | - | - |
| $\mathbf{a_{f_2}}$ | - | - |

To start the initial time difference, $t_k$, is calculated as shown below in equation (4.1) where $t$ is the GPS receiver time at the reception of the signal. This is taken from the ephemeris data given by the receiver.

$$t_k = t - t\_oe \tag{4.1}$$

The receiver time, $t$, is

$$t = t_{GPS_k} - \left(a_{f_0} + a_{f_1}(\mathrm{t} - t\_oe) + a_{f_2}(\mathrm{t} - t\_oe)^2 + \Delta t_r + T_{GD}\right) \tag{4.2}$$

and it is based on the atmospheric conditions received from the satellites. This a quadratic equation with a closed loop solution however it is solved numerically using Newton-Raphson's method for root finding. The method applied to equation (4.2) can be seen below in equation (4.3). The final value is then used to find $\mathrm{t_k}$.

$$t^{(n+1)} = t^{(n)} + \frac{t_{GPS_k} - \left(a_{f_0} + a_{f_1}(t^{(n)} - t\_oe) + a_{f_2}(t^{(n)} - t\_oe)^2 + \Delta t_r + T_{GD}\right) - t^{(n)}}{a_{f_1} + 2a_{f_2}(t^{(n)} - t\_oe) + 1} \tag{4.3}$$

Next, the mean anomaly,

$$M = M\_0 + nt_k \tag{4.4}$$

is found by using the ephemeris data and equation (4.1). Then, Kepler's Equation of Eccentric Anomaly,

$$M = E_k - e\, sin(E_k) \tag{4.5}$$

is solved iteratively by using Newton's method as

$$E_k = E_{k-1} - \frac{E_{k-1} - e\, sin(E_{k-1}) - M}{1 - e\, cos(E_{k-1})} \tag{4.6}$$

until

$$|E_k - E_{k-1}| \leq \varepsilon \tag{4.7}$$

Next, the true anomaly is solved by taking the ephemeris data and combining the data with the eccentric anomaly as

$$\upsilon_k = tan^{-1}\left(\frac{(r1me2)\, sin(E_k)}{cos(E_k) - e}\right) \tag{4.8}$$

The true anomaly is then used to find the argument of latitude as

$$\Phi_k = \upsilon_k + omega \tag{4.9}$$

The argument of latitude correction, radius correction and inclination corrections are

$$\delta u_k = (C\_uc)\, cos(2\Phi_k) + (C\_us)\, sin(2\Phi_k) \tag{4.10}$$

$$\delta r_k = (C\_rc)\, cos(2\Phi_k) + (C\_rs)\, sin(2\Phi_k) \tag{4.11}$$

and

$$\delta i_k = (C\_ic)\, cos(2\Phi_k) + (C\_is)\, sin(2\Phi_k) \tag{4.12}$$

respectively. They use the ephemeris data and the result from equation (4.9). Those results are then combined with the ephemeris data and used to find the corrected argument of latitude, the corrected radius, and corrected inclination as

$$u_k = \Phi_k + \delta u_k \tag{4.13}$$

$$r_k = (sqrt\_A)^2(1 - e\,cos(E_k)) + \delta r_k \tag{4.14}$$

and

$$i_k = i\_0 \ + \delta i_k + (IDOT)(t_k) \tag{4.15}$$

respectively. The corrected longitude of ascending node is then calculated as

$$\Omega_K = OMEGA\_0 + (OMEGADOT - \omega_e)t_k - \omega_e t_{oe} \tag{4.16}$$

by combining the ephemeris data, time data and the constant rotation of the earth, $\omega_e = 7.292115E - 5$. Next, the orbital plane coordinates, $X_K'$and $Y_K'$ are found as

$$X_K' = r_k\,cos(u_k) \tag{4.17}$$

$$Y_K' = r_k\,sin(u_k) \tag{4.18}$$

by using the ephemeris data and the previously solved equations. Finally, this information is all combined as

$$\begin{bmatrix} X_K \\ Y_K \\ Z_K \end{bmatrix} = \begin{bmatrix} X_K'\,cos(\Omega_K) - Y_K'\,sin(\Omega_K)\,cos(i_k) \\ X_K'\,sin(\Omega_K) + Y_K'\,cos(\Omega_K)\,cos(i_k) \\ Y_K'\,sin(i_k) \end{bmatrix} \tag{4.19}$$

to find the Cartesian coordinates of the given satellite in an earth centered earth fixed (ECEF) coordinate system. For the purpose of the precision study the coordinates were left in the ECEF coordinates for simplicity. For use in most autonomous vehicle navigations however, the coordinates should be converted into Universal Transverse Mercator (UTM) coordinates for simplicity.

## 4.2. Time Difference Estimation

One of the unknowns in the trilateration algorithm is the time difference between the satellite and the receiver. This time difference can be estimated from the raw measurement data given by the satellite, shown in Table 4.2 [54].

Table 4.2: Raw measurement data received from each satellite

| Parameter | Units |
|---|---|
| **Pseudo Range Integer Number (PRIN)** | Milliseconds |
| **Code phase (CP)** | 1/16th chip |
| **Pseudo Range (PR)** | centimeters |

The time difference is on a scale of milliseconds. Simply subtracting the calculated times of GPS reception, $t_{GPS_K}$, and the satellite transmission time, $t_K$, will not work because these values are typically calculated in seconds and do not have a high enough precision to find a reliable difference on the order of milliseconds. Instead, this value is found in a variety of ways and on the Copernicus II receiver that is used in this study the time is found from the code phase. The estimation of the time difference is

$$t_{est} = c * \left(\frac{PR}{1000} + \frac{CP}{16*1.023*10^6}\right) + \frac{PR}{100} \tag{4.20}$$

The code phase is 1/16th of the course acquisition code, or C/A code, of the chip, which is 1.23 Mbps for this receiver. This means the code phase unit is 61.0948ns. It should be noted that this is the time resolution of the receiver, when converted to a distance resolution by multiplying by the speed of light the resolution is 18.3158 meters and this error is too large for many applications. This is adjusted for by the time delay, $\Delta T$. The time delay is also composed of unknown factors such as Einstein curvature and unknown ionospheric conditions.

## 4.3. Receiver Position Estimation

As noted in Chapter 3, the set of equations made from using the satellite data is not consistent and the solution needs to be estimated. The set of equations can be represented as

$$F(q) = \begin{bmatrix} F(q_1) \\ F(q_2) \\ F(q_3) \\ F(q_4) \end{bmatrix} = \begin{bmatrix} (\mathrm{X}_1 - q_1)^2 + (\mathrm{Y}_1 - q_2)^2 + (\mathrm{Z}_1 - q_3)^2 - c^2(\,t_{est1} - q_4)^2 \\ (\mathrm{X}_2 - q_1)^2 + (\mathrm{Y}_2 - q_2)^2 + (\mathrm{Z}_2 - q_3)^2 - c^2(\,t_{est2} - q_4)^2 \\ (\mathrm{X}_3 - q_1)^2 + (\mathrm{Y}_3 - q_2)^2 + (\mathrm{Z}_3 - q_3)^2 - c^2(\,t_{est3} - q_4)^2 \\ (\mathrm{X}_4 - q_1)^2 + (\mathrm{Y}_4 - q_2)^2 + (\mathrm{Z}_4 - q_3)^2 - c^2(\,t_{est4} - q_4)^2 \end{bmatrix} \tag{4.21}$$

where

$$q = \begin{bmatrix} x \\ y \\ z \\ \Delta T \end{bmatrix} \tag{4.22}$$

and $q$ is sought so

$$F(q) = 0 \tag{4.23}$$

This problem can be approached in different ways with varying degrees of accuracy.

### 4.3.1. Least-Squares Method

The least-squares is a very common numerical method [55]–[58] for performing linear regressions on a set of data and, in the case of this study, finding a state estimation by minimizing a set of equations. As is typical for numerical solutions the least squares method starts with an initial guess, $q_0$, and then uses the Taylor series expansion,

$$F(q) = F(q_0) + \nabla F_q(q_0)\Delta q + \nu \tag{4.24}$$

to find the root of $F(q)$. In the expansion $\nu$ represents the higher order terms that are neglected. In formulating the problem it is convenient to form the notation of

$$\nabla F_q(q_0) = \mathrm{A}(q_0) \tag{4.25}$$

$$\Delta q = (\mathrm{q} - q_0) \tag{4.26}$$

$$b(q_0) = F(q) - F(q_0) \tag{4.27}$$

Then a set of linear equations can be formed as

$$b(q_0) = \mathrm{A}(q_0)\Delta q + \nu \tag{4.28}$$

and $q$ is sought for which

$$\mathrm{A}(q_0)\Delta q = 0 \tag{4.29}$$

This usually cannot be found because of the approximations so the minimization of the square difference is sought as

$$J(q_0) = [\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q]^T[\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q] \tag{4.30}$$

The derivative of (4.30) becomes

$$\delta J(q_0) = \delta[\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q]^T[\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q] + \cdots$$
$$\ldots [\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q]^T\delta[\mathrm{b}(q_0) - \mathrm{A}(q_0)\Delta q] \tag{4.31}$$

where

$$\delta J(q_0) = 0 \tag{4.32}$$

is the minimization of (4.30). Equation (4.32) can be simplified to

$$[\mathrm{A}(q_0)^T\mathrm{b}(q_0) - \mathrm{A}(q_0)^T\mathrm{A}(q_0)\Delta q] = 0 \tag{4.33}$$

Then through algebraic manipulation $\Delta q$ can be solved directly as

$$\Delta q = [\mathrm{A}(q_0)^T\mathrm{A}(q_0)]^{-1}\mathrm{A}(q_0)^T\mathrm{b}(q_0) \tag{4.34}$$

With $\Delta q$ found $q$ is solved for by

$$q_{k+1} = q_k + \Delta q_k \tag{4.35}$$

This process is repeated until

$$\varepsilon > \|\Delta q_k\|_2 \tag{4.36}$$

This forms the typical least square solution formed in most GPS trilateration algorithms. There are errors that come with this solution which can be attributed to the ignored higher terms in the Taylor's series expansion and the numerical approximation of the inverse of a matrix in solving for $\Delta q$.

### 4.3.2. Newtonian Method

One method that seeks to avoid ignoring the higher order terms in the Taylor series expansion [55]–[58] is by finding the unconstrained minimum of

$$q = \arg\min_{q} \; G(q) \tag{4.37}$$

where *G*(*q*) is defined as

$$\nabla G(q) = F(q) \tag{4.38}$$

So *G*(*q*) can be defined directly as

$$G(q) = \textstyle\sum_{k=1}^{4}[\int F_k(q)\, dq_k - \varphi_k(q)] \tag{4.39}$$

The terms $\varphi_S(q)$ uphold the continuity of $G(q)$ such that

$$\frac{\delta^2 G(q)}{\delta q_i \delta q_j} = \frac{\delta^2 G(q)}{\delta q_j \delta q_i} \tag{4.40}$$

The terms $\varphi_S(q)$ that make equation (4.40) true are

$$\varphi(q) = \begin{bmatrix} \varphi_1(q) \\ \varphi_2(q) \\ \varphi_3(q) \\ \varphi_4(q) \end{bmatrix} = \begin{bmatrix} (\mathrm{X}_2 - q_2)^2 q_2 + (\mathrm{X}_3 - q_3)^2 q_3 + (\mathrm{X}_4 - q_4)^2 q_4 \\ (\mathrm{Y}_1 - q_1)^2 q_1 + (\mathrm{Y}_3 - q_3)^2 q_3 + (\mathrm{Y}_4 - q_4)^2 q_4 \\ (\mathrm{Z}_1 - q_1)^2 q_1 + (\mathrm{Z}_2 - q_2)^2 q_2 + (\mathrm{Z}_4 - q_4)^2 q_4 \\ \kappa(t_{est1} - q_1)^2 q_1 + \kappa(t_{est2} - q_2)^2 q_2 + \kappa(t_{est3} - q_3)^2 q_3 \end{bmatrix} \tag{4.41}$$

where $-c^2$ is represented as *κ*. By defining the function *η*(*q*) as

$$\eta(q) = \begin{bmatrix} \eta_1(q) \\ \eta_2(q) \\ \eta_3(q) \\ \eta_4(q) \end{bmatrix} = \frac{1}{3} \begin{bmatrix} (\mathrm{X}_1 - q_1)^2(\mathrm{X}_1 - 2q_1) \\ (\mathrm{Y}_2 - q_2)^2(\mathrm{Y}_2 - 2q_2) \\ (\mathrm{Z}_3 - q_3)^2(\mathrm{Z}_3 - 2q_3) \\ \kappa(t_{est4} - q_4)^2(t_{est4} - 2q_4) \end{bmatrix} \tag{4.42}$$

equation (4.39) can be simplified as

$$G(q) = \textstyle\sum_{k=1}^{4}\{[(X_k - q_1)^2 + (Y_k - q_2)^2 + (Z_k - q_3)^2 - c^2(\, t_{estk} - q_4)^2] q_k \ldots$$

$$\ldots - \varphi_k(q) + \eta_k(q) \tag{4.43}$$

which can be solved numerically.

The most straight forward way to solve the minimization of *G* is to find the root of the gradient, *F*, using Newton's method for root finding. The equation in the case of the functions defined above becomes

$$q_{k+1} = q_k + \frac{J}{H} \tag{4.44}$$

where *J* is the Jacobian and *H* is the Hessian of *G*. The Jacobian and Hessian of *G* are defined as

$$J = \nabla G(q) = \begin{bmatrix} \frac{dG_1}{dx_1} & \cdots & \frac{dG_1}{dx_m} \\ \vdots & \ddots & \vdots \\ \frac{dG_n}{dx_1} & \cdots & \frac{dG_n}{dx_m} \end{bmatrix} = F(q) \tag{4.45}$$

In equation (4.21) the gradient of (4.45) is required so the Hessian needs to be formed as

$$H = \nabla^2 G(q) = \begin{bmatrix} \frac{d^2G_1}{dx_1{}^2} & \cdots & \frac{d^2G_1}{dx_m{}^2} \\ \vdots & \ddots & \vdots \\ \frac{d^2G_n}{dx_1{}^2} & \cdots & \frac{d^2G_n}{dx_m{}^2} \end{bmatrix} = \nabla F(q) = \begin{bmatrix} 2(\mathrm{X}_1 - q_1) \\ 2(\mathrm{Y}_2 - q_2) \\ 2(\mathrm{Z}_3 - q_3) \\ 2\kappa(t_{est4} - q_4) \end{bmatrix} \tag{4.46}$$

Substituting equations (4.45) and (4.46) into (4.44) the following solution to the GPS trilateration becomes

$$q_{k+1} = q_k + \cdots$$

$$\ldots 2 \begin{bmatrix} (\mathrm{X}_1 - q_1) & (\mathrm{X}_1 - q_2) & (\mathrm{X}_1 - q_3) & (\mathrm{X}_1 - q_4) \\ (\mathrm{Y}_2 - q_1) & (\mathrm{Y}_2 - q_2) & (\mathrm{Y}_2 - q_3) & (\mathrm{Y}_2 - q_4) \\ (\mathrm{Z}_3 - q_1) & (\mathrm{Z}_3 - q_2) & (\mathrm{Z}_3 - q_3) & (\mathrm{Z}_3 - q_4) \\ \kappa(t_{est4} - q_1) & \kappa(t_{est4} - q_2) & \kappa(t_{est4} - q_3) & \kappa(t_{est4} - q_4) \end{bmatrix}^{-1} \times \ldots$$

$$\cdots \begin{bmatrix} (\mathrm{X}_1 - q_1)^2 + (\mathrm{Y}_1 - q_2)^2 + (\mathrm{Z}_1 - q_3)^2 - c^2(\,t_{est1} - q_4)^2 \\ (\mathrm{X}_2 - q_1)^2 + (\mathrm{Y}_2 - q_2)^2 + (\mathrm{Z}_2 - q_3)^2 - c^2(\,t_{est2} - q_4)^2 \\ (\mathrm{X}_3 - q_1)^2 + (\mathrm{Y}_3 - q_2)^2 + (\mathrm{Z}_3 - q_3)^2 - c^2(\,t_{est3} - q_4)^2 \\ (\mathrm{X}_4 - q_1)^2 + (\mathrm{Y}_4 - q_2)^2 + (\mathrm{Z}_4 - q_3)^2 - c^2(\,t_{est4} - q_4)^2 \end{bmatrix} \tag{4.47}$$

For this study there are only four equations used in the set of equations used and because of the position of the satellites in relation to the receiver the Hessian of *G* should always be invertible.

### 4.3.3. Broyden-Fletcher-Goldfarb-Shanno Method

In trilateration algorithms that use more than four satellites and in other applications of numerical methods the inverse of the Hessian is not always able to be found directly. Because of this fact and the fact that finding the Hessian can be computationally intense there are a set of algorithms known as quasi-Newtonian methods. These methods seek to estimate the Hessian through iterations. The first quasi-Newtonian approximation was formulated by Davidson, Fletcher and Powell and is known as the DFP [59], [60] method which formed three properties about the Hessian that allowed for a numerical approximation based on those properties. That method was expanded on by Broyden [61], Fletcher [62], Goldfarb [63] and Shanno [64], each of whom expanded the DFP method independently. Their method, the BFGS method, seeks to approximate the inverse of the Hessian to avoid problems with inverting the Hessian should singularities form in the approximation of the Hessian in the DPF method. Formulation of the BFGS has become standard and can be found in many numerical method books [55]–[57].

In this method, the inverse of the Hessian, $\widetilde{H}_k$, is subject to the same set of constraints that help form a method for a numerical approximation. The first is that $\widetilde{H}_k$ must be symmetric and the second is that the gradient of the approximation must be equal to the gradient of the function the current and the previous time step, which is represented as

$$\nabla f_k + \widetilde{H}_k(q_{k-1} - q_k) = \nabla f_{k-1} \tag{4.48}$$

and rearranged to form the constraint

$$\widetilde{H}_k s_{k-1} = y_{k-1} \tag{4.49}$$

where

$$s_{k-1} = (q_{k-1} - q_k) \tag{4.50}$$

and

$$y_{k-1} = \nabla f_k - \nabla f_{k-1} \tag{4.51}$$

The final constraint is that the change in *H* at each time step should be minimized forming

$$\left\|\widetilde{H}_k - \widetilde{H}_{k-1}\right\|_w \tag{4.52}$$

subject to the first to constraints. This minimization is solved by

$$\widetilde{H}_k = (I - \mathrm{y}_{k-1}\rho_{k-1}s_{k-1}^T)\widetilde{H}_{k-1}(I - \mathrm{y}_{k-1}\rho_{k-1}\mathrm{y}_{k-1}^T) + y_{k-1}\rho_{k-1}y_{k-1}^T \tag{4.53}$$

where

$$\rho_{k-1} = (y_{k-1}^T s_{k-1})^{-1} \tag{4.54}$$

This can be simplified and rewritten as

$$\widetilde{H}_{k+1} = \widetilde{H}_k + \frac{[(s_k^T y_k + y_k^T \widetilde{H}_k y_k)(s_k s_k^T)]}{(s_k^T y_k)^2} - \frac{\widetilde{H}_k y_k s_k^T + s_k y_k^T \widetilde{H}_k}{s_k^T y_k} \tag{4.55}$$

Once the inverse of the Hessian is formed it can be used in place of the actual Hessian of *G* in equation (4.47).

## 4.4. Satellite Selection Algorithm

For the satellite selection the signal strengths were monitored and a selection of the most similar signal strengths were selected. The data from the preliminary study showed the signal strengths between 2 and 4 were most common. This gives a good selection criterion for Copernicus II GPS receivers in Fargo, ND. It is important to note that the strength criterion for selection may change based on the region and the capabilities of the receiver. Should this method prove to improve the trilateration precision more studies should be conducted on how to adjust the selection criterion for different regions and receivers.

# CHAPTER 5. ROBUST PATH TRACKING CONTROL

This chapter discusses the background and application of path tracking of the robot. Path tracking requires the combination of the kinematics, dynamics, sensor theory and sensor fusion discussed in the previous chapters to successfully guide the robot along the desired path.

## 5.1. Robot Path Tracking and Localization

For path tracking control it is necessary to define where the robot is in relation to the desired path. In this study the path will be a fixed, predetermined route that is preprogrammed into the robot although it is often desired to update the path with respects to the surroundings using methods such as simultaneous localization and mapping (SLAM). The control theory outlined here can be applied to either a SLAM technique or a preplanned path. Regardless of how the path is defined the path tracking seeks to find where the robot is, where it is supposed to be, and how to return the robot back to the correct path.

### 5.1.1. General Path Tracking and Localization

The general path tracking problem assumes that the robot moves through space with a body frame of $(x,y,z)$ and the inertial frame $(X_F, Y_F, Z_F)$. The body frame is attached to the center of gravity of the robot and it rotates and moves as the robot rotates and moves. The $x$, $y$ and $z$ axis lie on the back and forth, left and right, and up and down motions of the robot respectively with the positive $x$, $y$ and $z$ directions in the forward, right and up axis respectively. The inertial frame is fixed at some point $O(0,0,0)$ and the axis are oriented parallel to the ECEF coordinate system. In general the location and velocity in the inertial frame at time, $t$, is $(X,Y,Z)$ and $(V_X, V_Y, V_Z)$ respectively. Theses sets of data are related to each other as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \int V_X dt \\ \int V_Y dt \\ \int V_Z \, dt \end{bmatrix} \tag{5.1}$$

Since the body frame is attached the robot, the (*x*,*y*,*z*) coordinates will always be equal to zero. The usefulness in the body frame comes in defining the local velocities ($v_x$,$v_y$,$v_z$) to the inertial frame velocities as [65]

$$\begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} = \begin{bmatrix} c\theta_y c\theta_z & c\theta_x s\theta_z - c\theta_z s\theta_x s\theta_y & -s\theta_x s\theta_z - c\theta_x c\theta_z s\theta_y \\ -c\theta_y s\theta_z & c\theta_x c\theta_z + s\theta_x s\theta_y s\theta_z & c\theta_x s\theta_y s\theta_z - c\theta_z s\theta_x \\ s\theta_y & c\theta_y s\theta_x & c\theta_x c\theta_y \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{5.2}$$

where the roll-pitch-yaw (RPY) angles ($\theta_x$, $\theta_y$, $\theta_z$) relative to the inertial coordinate frame. In this equation and the following equations in this chapter $c\theta_i = \cos(\theta_i)$ and $s\theta_i = \sin(\theta_i)$.

The inverse of this formulation is often times more useful. In this case the current position, ($X_G$, $Y_G$, $Z_G$), is known along the known trajectory and the local velocities, ($v_x$, $v_y$, $v_z$), need to be found. So to find the inverse of (5.2) the derivative of the inertial frame position,

$$\begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} = \begin{bmatrix} \frac{dX}{dt} \\ \frac{dY}{dt} \\ \frac{dZ}{dt} \end{bmatrix} \tag{5.3}$$

is used to find

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} c\theta_y c\theta_z & c\theta_x s\theta_z - c\theta_z s\theta_x s\theta_y & -s\theta_x s\theta_z - c\theta_x c\theta_z s\theta_y \\ -c\theta_y s\theta_z & c\theta_x c\theta_z + s\theta_x s\theta_y s\theta_z & c\theta_x s\theta_y s\theta_z - c\theta_z s\theta_x \\ s\theta_y & c\theta_y s\theta_x & c\theta_x c\theta_y \end{bmatrix}^T \begin{bmatrix} \frac{dX}{dt} \\ \frac{dY}{dt} \\ \frac{dZ}{dt} \end{bmatrix} \tag{5.4}$$

With this formulation the body speed can now be related to the desired trajectory rate of change. If the RYP angles are fixed then the body speed is related using just the trajectory rate of change, however it is more common for curvilinear motion to be desired. In this case the body velocity is related to the rate of change of the trajectory and the RYP rates of change as

$$d\theta_x = Y v_z dt - Z v_y dt \tag{5.5}$$

$$d\theta_y = Z v_x dt - X v_z dt \tag{5.6}$$

$$d\theta_z = Xv_y dt - Yv_x dt \tag{5.7}$$

which is a more complex set of differential equations [66], [67]. The information that needs to be known about the robot at all times is the current position along the known trajectory.

There are two main sensors to find the inertial frame position: 3-D IMU and GPS sensors. These are often times used in tandem to create a more accurate position reading of the robot. The 3-D IMU can sense acceleration along three mutually perpendicular axis, $a_x$, $a_y$ and $a_z$. The IMU can also sense the angular acceleration about each axis $\omega_x$, $\omega_y$ and $\omega_z$. When the IMU is fixed to the center of gravity of the robot the information it provides can be related to the RYP angles and the body velocities as

$$\begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} = \begin{bmatrix} \int \omega_x dt \\ \int \omega_y dt \\ \int \omega_z \, dt \end{bmatrix} + \begin{bmatrix} \nu_x \\ \nu_y \\ \nu_z \end{bmatrix} \tag{5.8}$$

and

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \int a_x dt \\ \int a_y dt \\ \int a_z \, dt \end{bmatrix} + \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \end{bmatrix} \tag{5.9}$$

where ($\nu_x, \nu_y, \nu_z$) and ($\mu_x,\mu_y,\mu_z$) are the noise components for the RYP angles and body velocities (5.8) and (5.9) can now be used in equation (5.2) to find the inertial frame velocities and then equation (5.1) to find the inertial frame position.

The GPS receiver calculates the inertial frame position to by taking information from at least four satellites. The details of this calculation are introduced in Chapter 2 and then continued in Chapter 4. Chapter 4 also details the noise associated with these calculations and suggested improvements to the algorithm.

### 5.1.2. Skid Steered Path Tracking on High Friction, Flat Surfaces

The previous section discussed general path tracking in a large 3-D space which makes the equations large and cumbersome. It can be simplified by applying the kinematic and dynamic assumptions outlined in Chapter 3. The first major point in reducing the equations is instead of assuming the inertial frame is translated from the ECEF coordinate system it is assumed that the robot moves on a 2-D surface over a small localized area of earth that can be assumed to be flat such as Universal Transverse Mercator (UTM) zones. Path is then defined in the *X* and *Y* coordinates where

$$\begin{bmatrix} \frac{dX_r}{dt} \\ \frac{dY_r}{dt} \end{bmatrix} = \begin{bmatrix} V_X \\ V_Y \end{bmatrix} \tag{5.10}$$

and the body velocities are

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} c\theta_z & -s\theta_z \\ s\theta_z & c\theta_z \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \frac{dX}{dt} \\ \frac{dY}{dt} \end{bmatrix} \tag{5.11}$$

These are then used to define a curvilinear path in relation to the body velocities as

$$d\theta_z = Xv_y dt - Yv_x dt \tag{5.12}$$

The control parameters in this study are based on the linear velocities of the left and right wheels as $V_l$ and $V_r$ respectively. These can be related to the applied voltage for the left and right motor as shown in Chapter 3. The body velocities in the *y* axis is assumed to be zero because on high friction surfaces lateral sliding is not possible. So the body velocities and the heading angle become

$$v_x = \frac{1}{2}(V_r + V_l) \tag{5.13}$$

$$v_y = 0 \tag{5.14}$$

$$\theta_Z = \int \frac{1}{B}(V_r - V_l)dt \tag{5.15}$$

where $B$ is the axle width of the vehicle.

Like the general formulation, this study used IMU and GPS localization sensors. This study also used wheel encoders to add redundancy to the desired information. In this reduced system the only information required from the IMU are the values of $a_x$ and $\omega_z$. This is then used to find the body velocity in along the $x$ axis and the heading angle as

$$v_x = \int a_x dt + \mu_x \tag{5.16}$$

$$\theta_z = \int \omega_z \, dt + \nu_z \tag{5.17}$$

subject their respective noise components. The GPS sensor provides $X$ and $Y$ coordinates which are translated from the ECEF coordinates found in the position fix to UTM corrdinates.

The encoders are able to find the left and right wheel velocities which can be used to locate the robot using equations (5.13) through (5.15). The data from the encoder is read as left, $N_l$, and right, $N_r$, wheel counts which is translated to left and right linear velocities as

$$V_l = \frac{2\pi r_w}{\Delta t_e N_c} N_l + n_l \tag{5.18}$$

$$V_r = \frac{2\pi r_w}{\Delta t_e N_c} N_r + n_r \tag{5.19}$$

where $N_c$ is the number of counts in one revolution of the wheel and $n_l$ and $n_r$ are the errors associated with wheel slip on the left and right encoders respectively. It is unexpected that the wheels will slip because of the high friction but this still may happen, especially during turning maneuvers.

As discussed in detail in previous chapters, these sensors all have noise associated with them that make any single reading unreliable. This specific selection of sensors are commonly used for the way they can be used together. The errors in the encoders and the IMU will tend to drift without bound although over a short period of time, they are very reliable. The GPS on the other had is imprecise but maintains accuracy over time. The GPS can then be used to keep the overall fusion of the measurement from drifting and the IMU and the encoders can be used to keep the states precise.

## 5.2. Formulating Linear State Space Equations

Many standard control algorithms that can be used to solve for the optimal and robust control of a dynamic systems first require that the system be put into standard state space form. As indicated in Chapter 3, the dynamic model for the vehicle can be represented by

$$\frac{dV_l}{dt} = \left(\frac{2k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)V_{DC_L} - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)V_l + \Delta_L \tag{5.20}$$

$$\frac{dV_r}{dt} = \left(\frac{2k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)V_{DC_R} - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)V_r + \Delta_R \tag{5.21}$$

$$\frac{dv_x}{dt} = \left(\frac{k_t n_r r_w}{R_a(4I_w+mr_w^2)}\right)(V_{DC_R} + V_{DC_L}) - \left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)v_x + \Delta_{Loss} \tag{5.22}$$

$$\frac{d\Omega}{dt} = \left(\frac{B^2 k_b k_t n_r^2}{2I_{ICR}R_a r_w^2}\right)\Omega - \left(\frac{Bk_t n_r}{2I_{ICR}R_a r_w^2}\right)(V_{DC_R} - V_{DC_L}) + \Delta M_\Omega \tag{5.23}$$

and the final kinematic equations are

$$\frac{d\theta_z}{dt} = \Omega \tag{5.24}$$

$$\frac{dX_r}{dt} = v_x\, cos(\theta_z) \tag{5.25}$$

$$\frac{dY_r}{dt} = v_x\, sin(\theta_z) \tag{5.26}$$

If the state vector, $x_k$, is defined as

$$x_k = \begin{bmatrix} V_l \\ V_r \\ v_x \\ \Omega \\ \theta_z \\ X \\ Y \end{bmatrix} \tag{5.27}$$

and the measurement vector obtained from all sensors on the vehicle is

$$y_k = \begin{bmatrix} N_l \\ N_r \\ a_{x1} \\ a_{x2} \\ \omega_{z1} \\ \omega_{z2} \\ X_{GPS} \\ Y_{GPS} \end{bmatrix} \tag{5.28}$$

Then these equations were transformed in discrete time form using Euler approximation as [24]

$$x_{k+1} = A_k x_k + B_k u_k + \Delta \tag{5.29}$$

$$y_k = C_k x_k + \Delta \tag{5.30}$$

where Δ accounts for all the uncertainties and error terms discussed earlier, The control input vector, $u_k$, becomes

$$u_k = \begin{bmatrix} V_{DCl} \\ V_{DCr} \end{bmatrix} \tag{5.31}$$

Using equations (5.20) through (5.26) the state space matrices can be formed through algebraic manipulation. Therefore the system matrix is

$$A_k = \begin{bmatrix} 1-\left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)\Delta T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1-\left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)\Delta T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\left(\frac{2k_b k_t n_r^2}{R_a(4I_w+mr_w^2)}\right)\Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1+\left(\frac{B^2 k_b k_t n_r^2}{2I_{ICR}R_a r_w^2}\right)\Delta T & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta T & 1 & 0 & 0 \\ 0 & 0 & \Delta T\, cos(\theta_z) & 0 & 0 & 1 & 0 \\ 0 & 0 & \Delta T\, sin(\theta_z) & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.32}$$

and the input matrix is

$$B_k = \begin{bmatrix} \frac{-2k_t n_r r_w}{R_a(4I_w+mr_w^2)} & 0 \\ 0 & \frac{-2k_t n_r r_w}{R_a(4I_w+mr_w^2)} \\ \frac{-k_t n_r r_w}{R_a(4I_w+mr_w^2)} & \frac{-k_t n_r r_w}{R_a(4I_w+mr_w^2)} \\ \frac{Bk_t n_r}{2I_{ICR}R_a r_w^2} & \frac{-Bk_t n_r}{2I_{ICR}R_a r_w^2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.33}$$

The measurement process matrix is

$$C_k = \begin{bmatrix} \frac{N_c \Delta t_e}{2\pi r_w} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{N_c \Delta t_e}{2\pi r_w} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-2k_b k_t n_r^2}{R_a(4I_w + m r_w^2)} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-2k_b k_t n_r^2}{R_a(4I_w + m r_w^2)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\Delta t} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.34}$$

and the feedforward matrix is zero. The actual properties of the robot are summarized below in Table 5.1.

Table 5.1: Parameter values

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $k_b$ | $0.154 \frac{\text{V}}{\text{rad/s}}$ | $B$ | 0.86 m, .81 |
| $k_t$ | $0.13 \frac{\text{Nm}}{\text{amp}}$ | $I_{ICR}$ | 26.75 kgm$^2$ |
| $n_r$ | 20 | $\Delta T$ | 0.785 |
| $R_a$ | 0.21 Ω | $\Delta T_e$ | 0.785 |
| $I_w$ | .764 kgm$^2$ | $N_c$ | 1024 |
| $m$ | 230 kg | $r_w$ | 0.2667 m |
| $N_c$ | 4096 | | |

These values can be placed in equations (5.32) through (5.34) to find the numerical values. It is important to note that the system matrix needs to be update with every time step because the heading angle will constantly be changing. With this system in place the control objective is to minimize the tracking error between the robot coordinates and the path coordinates using something similar to a *p*-norm,

$$\varepsilon = \left\| \begin{bmatrix} X \\ Y \end{bmatrix} - \begin{bmatrix} \hat{X} \\ \hat{Y} \end{bmatrix} \right\|_p \tag{5.35}$$

which will be the focus of the following sections.

## 5.3. Background on Linear Quadratic Control Design

When controlling an autonomous application often times there is some parameter that the designer attempts to optimize based on the control input. This could be anything from the energy exerted by the actuators to the efficiency with which the system operates. In path tracking applications such as the one in this study a popular optimization is the find the minimum tracking error. The continuous time state space system is defined as

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{5.36}$$

$$y(t) = Cx(t) + Du(t) \tag{5.37}$$

where the state vector is $x$, and the input, $u$,. The system matrices $A$ represents the system dynamics, $B$ the control input matrix, $C$ represents the sensor fusion matrix and $D$ represents the feed forward control. Correspondingly, the discrete time state space system is defined as

$$x_k = Ax_{k-1} + Bu_k \tag{5.38}$$

$$y_k = Cx_k + Du_k \tag{5.39}$$

For a system to have an optimal control it is required first that the system be controllable, observable and stable. The controllability of the system is the idea that the system should respond to an input by changing all elements of its state vector; practically, controllability is achieved by proper selection and placement of system actuators. An observable system is one where all the states are able to be measured directly or indirectly by one or multiple sensors and is achieved by proper placement and selection of sensors. Normally systems are required to be stable during their operations, i.e. maintains a certain equilibrium state and would return to that state when disturbed. This concept can be seen visually in Figure 5.1 where the ball on the left will return to the bottom of the “cup” if disturbed and the ball on the right will roll off the “hill” when disturbed. Here the left side represents an unstable system and the right side represents a table system. There are many ways to categorize stability but some of more useful conditions are known as the Lyapunov Criterion [68].

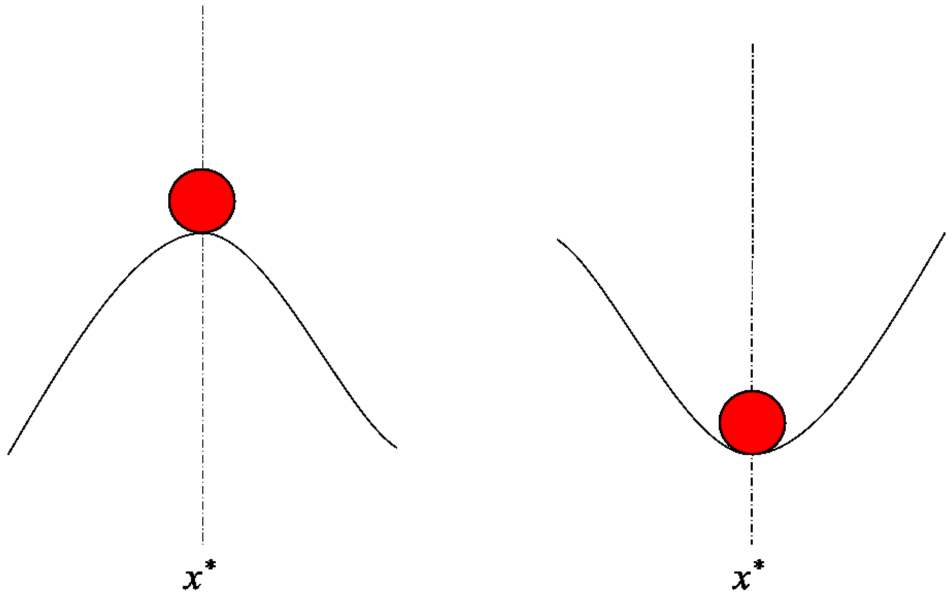


Figure 5.1: System stability

### 5.3.1. Lyapunov Criterion and the Bounded Real Lemma on Continuous Time Systems

Stability of dynamic systems is governed by the popular Lyapunov Criterion, which is a set of equations that describe a stable system based on some scalar function of the state vector $V(x(t))$ known as the Lyapunov function, which is always positive. The Lyapunov criteria for a stable system require the Lyapunov function to be decreasing with time and reaching a zero value at the equilibrium state $x^*$, i.e.,

$$V(x^*) = 0 \tag{5.40}$$

$$V(x(t)) \geq 0, \forall t \geq 0 \tag{5.41}$$

$$\frac{dV(x(t))}{dt} \geq 0, \forall t \geq 0 \tag{5.42}$$

There are many possible ways of defining the Lyapunov function for any given system, however, the most popular for linear systems is the quadratic function, also known as the kinetic energy function

$$V(x) = x^T Q x \tag{5.43}$$

where $Q = Q^T$. The first two criterion can be shown relatively easily and the last criterion requires the derivative of the quadratic function to be taken and is shown as

$$\dot{V}(x) = \dot{x}^T Q x + x^T \left[ Q\dot{x} + \dot{Q}x \right] \tag{5.44}$$

For a homogeneous linear system

$$\dot{x}(t) = Ax(t) \tag{5.45}$$

$$y(t) = Cx(t) \tag{5.46}$$

the decreasing condition of the Lyapunov function (5.42) becomes

$$\dot{V}(x) = [Ax]^T Q x + x^T \left[ QAx + \dot{Q}x \right] \tag{5.47}$$

which implies that

$$A^T Q + QA + \dot{Q} \leq 0 \tag{5.48}$$

If $Q$ is time invariant, which is almost always the case for linear systems, then this condition becomes

$$A^T Q + QA = -P \tag{5.49}$$

or

$$A^T Q + QA + P = 0 \tag{5.50}$$

for some matrix $P=P^T$. The equation, known as the Lyapunov equation can be solved for $Q$ to find a function that fits the Lyapunov Criterion.

Associated with the Lyapunov equations is the algebraic Riccati equation which appears in many applications of linear optimal control. One application of Riccati is in defining conditions for which the output of the system is bounded. The governing principle for this condition is the bounded real lemma, which states that if for the system in (5.36) and (5.37)

$$||y||_\infty < \gamma \tag{5.51}$$

then there exists a nonnegative matrix $P = P^T$ such that

$$A^T P + PA + C^T C + PB(\gamma^2 I - DD^T)^{-1} B^T P \leq 0 \tag{5.52}$$

### 5.3.2. Linear Quadratic Regulator for Continuous Time Systems

The Linear Quadratic Regulator (LQR) forms an optimal control algorithm that seeks to minimize a quadratic cost function based on methods first developed by Kalman in [69], [70] and some of the first applications of these methods were developed in [71], [72]. Quite often the quadratic cost happens to be a Lyapunov functions for the system under study, therefore minimization of such quadratic function would bring the system to its equilibrium and stabilize it. The common quadratic cost function for linear systems is the Hamiltonian

$$H(x, u, \lambda) = \frac{1}{2}(x^T Q x + 2x^T N u + u^T R u) + \lambda^T (Ax + Bu - \dot{x}) \tag{5.53}$$

where $\lambda$ is another variable known as the Lagrange multiplier. Minimization of this function is met by satisfying three sets of equations known as the state equation, $\frac{dH}{d\lambda} = 0$, costate equation $\frac{dH}{dx} = 0$, and stationarity equation $\frac{dH}{du} = 0$. The stationarity equation leads to

$$N^T x + Ru + B^T \lambda = 0 \tag{5.54}$$

which when solved for the optimal input $u_{opt}$ results in

$$u_{opt} = -R^{-1}(N^T x + B^T \lambda) \tag{5.55}$$

The LQR solution is completed by using the method of [73] where the Lagrange multiplier is treated as a linear combination of the state vector as

$$\lambda = Px \tag{5.56}$$

such that

$$u_{opt} = -R^{-1}(N^T + B^T P)x \tag{5.57}$$

or

$$u_{opt} = K_{opt}x \tag{5.58}$$

where the optimal feedback gain is

$$K_{opt} = -R^{-1}(N^T + B^T P) \tag{5.59}$$

The symmetric term *P* used in defining the Lagrange multiplier is determined and from the state and costate equations which together define

$$-\dot{P} = (A - BR^{-1}N^T)^T P + (A - BR^{-1}N^T) - PBR^{-1}B^T P + (Q - NR^{-1}N^T) \tag{5.60}$$

which is the differential Riccati equation. In linear systems, the value of *P* is determined from the steady state solution of the corresponding Algebraic Riccati equation

$$(A - BR^{-1}N^T)^T P + (A - BR^{-1}N^T) - PBR^{-1}B^T P + (Q - NR^{-1}N^T) = 0 \tag{5.61}$$

### 5.3.3. Lyapunov Criterion and Bounded Real Lemma on Discrete Time Systems

The Lyapunov criterion equation (5.42) for continuous time systems transform into discrete time from as

$$V(x_{k-1}) - V(x_k) \geq 0, \forall k \tag{5.62}$$

If the Lyapunov function is defined as in (5.42) then for the homogeneous system (5.44) the condition (5.62) leads to

$$V(x_{k-1}) - V(x_k) = x_{k-1}{}^T Q x_{k-1} - [Ax_{k-1}]^T Q [Ax_{k-1}] \geq 0 \tag{5.63}$$

which requires the presence of $Q = Q^T$ such that

$$Q - A^T Q A \geq 0 \tag{5.64}$$

or

$$A^T Q A - Q \leq 0 \tag{5.65}$$

The Bounded real lemma for discrete time systems can also be shown to require a matrix P such that

$$A^TPA - P + C^TC + (A^TPB + C^TD)(\gamma^2 I - DD^T - B^TPB)^{-1}(B^TPA + D^TC) \leq 0 \qquad (5.66)$$

## 5.3.4. Linear Quadratic Regulator for Discrete Time Systems

Quadratic regulators for discrete time systems strive to minimize that the corresponding Hamilton-Jacobian cost function which is defined as

$$H_k = {x_{k-1}}^T Q x_{k-1} \qquad (5.67)$$

where also

$$H_k = {x_{k-1}}^T P x_{k-1} + {u_k}^T R u_k + {x_k}^T Q x_k \qquad (5.68)$$

such that the control problem is to find u that satisfies

$${x_{k-1}}^T Q x_{k-1} = {x_{k-1}}^T P x_{k-1} + {u_k}^T R u_k + {x_k}^T Q x_k \qquad (5.69)$$

By using equation (5.38) we find that

$${x_{k-1}}^T Q x_{k-1} = {x_{k-1}}^T P x_{k-1} + {u_k}^T R u_k + [A x_{k-1} + B u_k]^T Q [A x_{k-1} + B u_k] \qquad (5.70)$$

which results in an optimal control of

$$u_{opt} = -(R + B^TPB)^{-1} B^T Q A x_{k-1}$$

Where $Q = Q^T$ is determined by using this optimal controller in equation (5.70) to result in the discrete time Ricatti equation

$$Q = P + A^TQ\mathrm{A} - \mathrm{A}^TQ\mathrm{B}(R + B^TPB)^{-1}B^TQA$$

## 5.4. Control Formulation the Skid Steered Vehicle

The control algorithm used in this study is a form of the Linear Quadratic Gaussian (LQG) controller. As stated earlier this controller assumes the validity of the separation principle between the measurement and dynamic uncertainties. The two main parts to this controller is the formulation of a Kalman filter to estimate the state of the robot and then a formulation of Linear

Quadratic Regulator (LQR) controller which uses the estimated state to find an optimal input vector based on the overall position error.

For both the Kalman filter design and the control design it is useful to further define the uncertainty terms present in the system. The Euler approximations of the standard state space can be written to include the dynamic and measurement uncertainties as

$$x_k = [A_k + \Delta_A]x_k + [B_k + \Delta_B]u_k + w_k \tag{5.71}$$

$$y_k = [C_k + \Delta_C]x_k + v_k \tag{5.72}$$

The terms $\Delta_A, \Delta_B,$ and $\Delta_C$ represent the uncertainties caused by the linearization and simplification of the dynamic and kinematic equations. The state vector can then be expressed in terms of the nominal value, $x_{n_k}$ , from the modeled dynamics and the uncertain value, $\Delta x_k$, from the ignored dynamics as

$$x_k = \ x_{n_k} + \Delta x_k \tag{5.73}$$

which satisfy

$$\Delta x_{k+1} = \Delta_A \Delta x_k + \Delta_B u_k \tag{5.74}$$

$$\Delta y_k = \Delta_C \Delta x_k \tag{5.75}$$

or simply

$$\begin{bmatrix} \Delta x_{k+1} \\ \Delta y_k \end{bmatrix} = \begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ u_k \end{bmatrix} \tag{5.76}$$

The continual effects of the uncertainties will be kept in bound as long as

$$\varepsilon = \left\| \begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} \right\|_\infty < 1 \tag{5.77}$$

The uncertainty matrix above can also be represented as

$$\begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} = \begin{bmatrix} A_x \\ A_Y \end{bmatrix} W[B_1 \quad B_2] \tag{5.78}$$

where $A_x, A_Y, B_1$ and $B_2$ are constant structural matrices and

$$W^T W \leq 1 \tag{5.79}$$

### 5.4.1. Control Function

As described earlier the control parameters being sought are the voltages that need to be applied to the left and the right motor. The required voltage is determined through the control algorithm that seeks to keep vehicle's center of gravity (*X, Y*) follow the desired path ($\hat{X}$, $\hat{Y}$) at a set speed $V_G$. If the vehicle's current center of gravity is at (*X*, *Y*) and its current orientation is at $\theta$ , the control problem is solved by the constrained minimization of the tracking error, $\varepsilon$ [24]. The tracking error is defined by the Euclidean distance

$$\varepsilon^2 = \left(\hat{X} - X\right)^2 + \left(\hat{Y} - Y\right)^2 + \left(tan^{-1}\left(\frac{dY}{dX}\right) - \theta\right)^2 \tag{5.80}$$

subject to the requirements

$$\left(\frac{dX}{dt}\right)^2 + \left(\frac{dY}{dt}\right)^2 = V^2 \tag{5.81}$$

This constrained minimization is a nonlinear problem with no closed loop solution. However, there is a numerical solution to the problem. The navigation data will be sampled at regular time intervals, $\Delta T$ and the motion will be determined through straight line segments $\Delta L$ defined as

$$\Delta L = \sqrt{\Delta X^2 + \Delta Y^2} \tag{5.82}$$

At any given time interval, *k*, the next position is determined by

$$X_{k+1} = X_k + \Delta X_k \tag{5.83}$$

$$Y_{k+1} = Y_k + \Delta Y_k \tag{5.84}$$

As $\Delta L_{max}$ decreases the motion of the vehicle along the desired path will become smoother. To account for boundary cases equation (5.82) can be turned into an inequality which can be used to find $\Delta X$ and $\Delta Y$.

$$\sqrt{\Delta {X_k}^2 + \Delta {Y_k}^2} \leq \Delta L_{\max} \tag{5.85}$$

The fixed velocity *V* determines the $\Delta L_{max}$ and $\Delta T$ by

$$V_k = \frac{\Delta L_{\max}}{\Delta T} \tag{5.86}$$

With a fixed $\Delta T$, the desired $\Delta X_k$ and $\Delta Y_k$ can be established from the kinematic relations as

$$\Delta X_k = V_k \cos(\theta_k)\Delta T \tag{5.87}$$

$$\Delta Y_k = V_k \sin(\theta_k)\Delta T \tag{5.88}$$

Both $\hat{V}_k$ and $\hat{\theta}_k$ depend on the current values of $V_r$ and $V_l$.

$$\theta_k = \tan^{-1}\left(\frac{\hat{Y}_k - Y_k}{\hat{X}_k - X_k}\right) = \tan^{-1}\left(\frac{\Delta Y_k}{\Delta X_k}\right) \tag{5.89}$$

Equation (5.89) simplifies the problem by showing that $\theta_k$ can be defined in just terms of the current position ($\Delta X_k$, $\Delta Y_k$) and the next desired position ($\hat{X}_k$, $\hat{Y}_k$).

In Figure 5.2 the desired path is shown with respect to the vehicle's current position ($X_k$, $Y_k$), previous position ($X_{k-1}, Y_{k-1}$) and intended position ($\hat{X}_k$, $\hat{Y}_k$). The circle represents the radius of $\Delta L_{max}$ and the points where the circle intersects the desired path are (*X*, *Y*) and (*X"*, *Y"*). To decide which point of intersection should be chosen for time *k*+1 the distance between each point of intersection and the position at time *k*-1 should be determined. Whichever point is further from the previous position should be chosen as the next point.

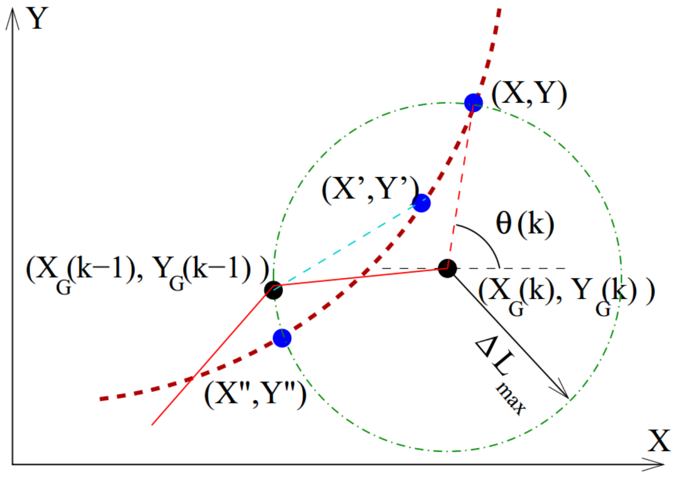


Figure 5.2: Path tracking

The current desired orientation $\theta_k$ is known and the old orientation $\theta_{k-1}$ is known. With those values the change in orientation *Δθ<sub></sub>*$_k$ can be found and related to the velocities $V_r$ and $V_l$ by

$$\Delta\theta_k = \theta_k - \theta_{k-1} = (V_r - V_l)\frac{\Delta T}{B} \tag{5.90}$$

The coordinate changes can also be found using the left and right wheel velocities as

$$\Delta X_k = (V_r + V_l)\frac{\Delta T}{2}\cos(\theta_k) \tag{5.91}$$

$$\Delta Y_k = (V_r + V_l)\frac{\Delta T}{2}\sin(\theta_k) \tag{5.92}$$

By combining equations (5.90) through (5.92) the left and right velocities at time *k* can be found as

$$V_{r_k} = \frac{\Delta X_k}{\Delta T}\cos(\theta_k) + \frac{\Delta Y_k}{\Delta T}\sin(\theta_k) + \frac{\mathrm{B}}{2\Delta T}\Delta\theta_k \tag{5.93}$$

$$V_{l_k} = \frac{\Delta X_k}{\Delta T}\cos(\theta_k) + \frac{\Delta Y_k}{\Delta T}\sin(\theta_k) - \frac{\mathrm{B}}{2\Delta T}\Delta\theta_k \tag{5.94}$$

Now that the left and right wheel velocities have been determined, the required wheel torques can be calculated.

### 5.4.2. Control Solution

With the state estimation from the Kalman filter and the control constraints established the control solution can be formulated based on waypoint selection. If the path at time interval *k* is known as

$$p_k = \begin{bmatrix} \hat{\theta}_k \\ \hat{X}_k \\ \hat{Y}_k \end{bmatrix} \tag{5.95}$$

then the robot movement can be defined as

$$\begin{bmatrix} \Delta\theta_k \\ \Delta X_k \\ \Delta Y_k \end{bmatrix} = p_k - E_1 x_k \tag{5.96}$$

where

$$E_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.97}$$

where the change along the *X* and *Y* axis is subject to

$$\Delta\theta_k = \tan^{-1}\left(\frac{\Delta Y_k}{\Delta X_k}\right) - \theta_k \tag{5.98}$$

It is desired to find the controller gain matrix $\Lambda_k$ such that the controller

$$u_k = \Lambda_k x_k \tag{5.99}$$

minimizes the 2-norm tracking error

$$\varepsilon = \left\| \begin{bmatrix} \hat{\theta}_{k+1} \\ \hat{X}_{k+1} \\ \hat{Y}_{k+1} \end{bmatrix} - \begin{bmatrix} \theta_{k+1} \\ X_{k+1} \\ Y_{k+1} \end{bmatrix} \right\|_2, \forall k \tag{5.100}$$

If the dynamic system without disturbances is represented as

$$x_{k+1} = [A_k + \Delta_A]x_k + [B_k + \Delta_B]u_k \tag{5.101}$$

then objective function becomes

$$\varepsilon = \| p_k - E_1([A_k + \Delta_A]x_k + [B_k + \Delta_B]u_k \|_2 \tag{5.102}$$

The constraints (5.90) through (5.92) can be defined as

$$\Delta\theta_k = 2(V_r - V_l)\frac{\Delta T}{B} - \Omega\Delta T \tag{5.103}$$

$$\Delta X_k = (V_r + V_l)\frac{\Delta T}{2}\cos(\tilde{\theta}_k) - v_x \Delta T \cos(\tilde{\theta}_k) \tag{5.104}$$

$$\Delta Y_k = (V_r + V_l)\frac{\Delta T}{2}\sin(\tilde{\theta}_k) - v_x \Delta T \sin(\tilde{\theta}_k) \tag{5.105}$$

where

$$\tilde{\theta}_k = \tan^{-1}\left(\frac{\Delta Y_k}{\Delta X_k}\right) \tag{5.106}$$

Therefore, the robot motion can now be expressed as

$$\begin{bmatrix} \Delta\theta_k \\ \Delta X_k \\ \Delta Y_k \end{bmatrix} = E_2 x_k \tag{5.107}$$

where

$$E_2 = \begin{bmatrix} -2\frac{\Delta T}{B} & -2\frac{\Delta T}{B} & 0 & -\Delta T & \frac{\Delta\theta_k}{\theta_k} & 0 & 0 \\ \Delta T\cos(\tilde{\theta}_k) & \Delta T\cos(\tilde{\theta}_k) & -\Delta T\cos(\tilde{\theta}_k) & 0 & 0 & \frac{\Delta X_k}{X_k} & 0 \\ \Delta T\sin(\tilde{\theta}_k) & \Delta T\sin(\tilde{\theta}_k) & -\Delta T\sin(\tilde{\theta}_k) & 0 & 0 & 0 & \frac{\Delta Y_k}{Y_k} \end{bmatrix} \qquad (5.108)$$

By combing the robot motion defined in equation (5.96) with the constraint defined in (5.85) the path vector can be expressed in terms of the state vectors as

$$p_k = (E_1 + E_2)x_k \qquad (5.109)$$

which can be used in to reformulate the cost function as

$$\varepsilon = \|(E_2 + E_1[I - A_k - A_xWB_1])x_k - (E_1[B_k + A_xWB_2])u_k\|_2 \qquad (5.110)$$

The cost function can now be formed as a quadratic Lyapunov function. To find the optimal control input based on the cost function a dynamic optimization is performed as described in Section 6.3. The discrete form of the quadratic function can be shown as

$$\varepsilon = \sum_k x_k{}^T\Theta_{\xi\xi}x_k + u_k{}^T\Theta_{\mathrm{uu}}u_k - 2x_k{}^T\Theta_{\xi\mathrm{u}}u_k \qquad (5.111)$$

where

$$\Theta_{\xi\xi} = (E_2 + E_1[I - A_k - A_xWB_1])^T(E_2 + E_1[I - F(k) - A_xWB_1]) \qquad (5.112)$$

$$\Theta_{\mathrm{uu}} = (E_1[B_k + A_xWB_2])^T(E_1[B_k + A_xWB_2]) \qquad (5.113)$$

$$\Theta_{\xi\mathrm{u}} = (E_2 + E_1[I - A_k - A_xWB_1])^T(E_1[B_k + A_xWB_2]) \qquad (5.114)$$

The solution is well known and can be found in [74], [75] as

$$u_k = -\Lambda_k x_k \qquad (5.115)$$

where

$$\Lambda_k = -\left(B_k{}^T\mathrm{S}_kB_k + \Theta_{\mathrm{uu}}\right)^{-1}\left(A_k{}^T\mathrm{S}_kB_k + \Theta_{\xi\mathrm{u}}\right)^T \qquad (5.116)$$

and $S_k$ is the recursive solution the Riccati equation

$$\mathrm{S}_k = A_k{}^T\mathrm{S}_{k-1}A_k + \Theta_{\xi\xi} - \ldots$$

$$\ldots\left(A_k{}^T \mathrm{S}_{k-1} B_k + \Theta_{\xi\mathrm{u}}\right)\left(B_k{}^T \mathrm{S}_{k-1} B_k + \Theta_{\mathrm{uu}}\right)^{-1}\left(A_k{}^T \mathrm{S}_{k-1} B_k + \Theta_{\xi\mathrm{u}}\right) \tag{5.117}$$

This control assumes that the state vector is known. The state estimation through filtering and fusion is described in the next chapter.

# CHAPTER 6. SENSOR PROCESSING: FILTERING AND FUSION

It is widely known that sensor measurement data can never be completely trusted because of their inherent noise. Autonomous applications need to be especially aware of these imperfections in the sensor data because errors in this data can be detrimental to the control algorithm. These errors come in two main forms, accuracy error and precision errors and there are different solutions to reduce each of them. Filtering can help increase the precision of the sensors, while the accuracy may be improved by using multiple sensors that are fused together. Often times the filtering and fusion of sensor can be done at the same time where noise from each single sensor is filtered out and the results from multiple sensors are fused together all in one large algorithm. The methods of filtering noise from a single sensor and fusing multiple sensors are often related as it will become apparent in later sections of this chapter. A widely accepted definition of sensor fusion as given by Joint Directors of Laboratories is [76]:

*"A multi-level process dealing with the association, correlation, combination of data and information from single and multiple sources to achieve refined position, identify estimates and complete and timely assessments of situations, threats and their significance."*

A good categorization of what is obtained by sensor fusion is also in [77], which categorizes the goals of sensor fusion as either creating more information making it more robust or just introducing complimentary information. There are many more ways to break down different levels and categories of sensor fusion such as in [78] but the focus of this chapter is to outline techniques used for state estimation and fusion of sensor based on the probability of information being accurate.

## 6.1. Fusion Methods

Sensor fusion can be categorized by how they deal with noise. Traditionally sensor fusion methods were based on the probability that a sensor reading was true. Newer studies found other methods to deal with problems that arose in the probabilistic methods such as what happens when two sets of information contradict each other.

### 6.1.1. Non-Probabilistic Fusion Methods

One of the main non-probabilistic methods for sensor fusion is fuzzy logic which is based on fuzzy sets of information. The basic theory behind fuzzy logic is that instead of information being part of a set or excluded from a set, it can have a degree of membership to a set of information [65]. Fuzzy logic is especially practical in instances where competing information sets contradict each other. Another popular non-probabilistic fusion technique is based on the Dempster-Shafer theory developed by Dempster in 1930 [79], [80]. This theory which provides a framework for reasoning using imprecise data, has found more applications in robotics when decisions need to be made upon a robot's state [65]. Since the state cannot always be determined with complete certainty the Dempster-Shafer theory allows for ambiguity in the state of the robot to be trusted (belief theory.) Essentially, this approach is similar to fuzzy logic approach and it requires the designer to develop a set of rules that define the level of ambiguity in the data and how different pieces of partial information are combined.

### 6.1.2. Probabilistic Fusion Methods

Many methods of sensor fusion assume some knowledge in the probability of a sensor's information being accurate, and use that information to estimate the state of the object. This process of state estimation based on probability is usually founded on Bayes law, which states the probability of receiving the set of information from a given sensor based on the noise distribution of that sensor. This process is described well in [81] where sensors are combined into a “Bayesian team.” This Bayesian team can be described as sensors with competing information which can be rectified by using the bargaining problem described by [82] for economic information and extended to sensors by [9] and [10].

One of the most fundamental state estimation models was developed by Kalman in in the sixties [85]. In this model the assumption is made that the noise of the sensors are known to be Gaussian and have a zero mean. It is also assumed that dynamics of the model are completely linear. With this information the Bayesian equations have an optimal analytical solution which is known as the Kalman filter; these filters are widely used in control applications because of their ability to filter and fuse information together effectively.

The Kalman Filter makes many assumptions about the system however, one of the main ones being that the system is linear. However, non-linear systems can be linearized by using a first order Taylor series expansion resulting in the so called Extended Kalman Filter (EKF) [86]. Linearization of a non-linear model can be computationally expensive, and also introduces inherent errors, or “scents,” that are carried throughout the state estimation. This linearization can be eliminated by sampling data and finding the true mean and covariance of a sensor, which can then be used directly through the non-linear system to find the analytical solution to the Bayesian state estimation. This filter is known as an unscented Kalman filter and was first developed in [87].

All of the extensions of the Kalman filter assume the noise is known to have a Gaussian distribution but this is not always the case. In the particle filter shown in [88], the noise is not known. However, the noise distribution is calculated by taking a sufficiently large sample size. This distribution is then used in the Bayesian equations to solve for the state estimation. This method is useful for unknown noise in sensors or non-Gaussian distributions. It can also handle the non-linearities that the extensions to the Kalman filter seek to address.

## 6.2. The Kalman Filter Algorithm

As shown in Chapter 3 this research developed a linear model of the skid steered vehicle, therefore, the Kalman filter is ideal for this application if all sensor noises are assumed to be Gaussian. This section will discuss the process of state estimation and sensor fusion using a Kalman filtering.

Linear dynamical systems are normally defined as

$$\dot{x} = Ax + B(u + w) \tag{6.1}$$

$$y = Cx + D(u + w) + v \tag{6.2}$$

where $x$, is the state vector; the input signal applied to the actuators, $u$, is subject to disturbances in the actuators, $w$, and the sensor measurements, $y$, are subject noise corruptions $v$. Control applications require knowledge of the state vector from the measurements, $y$, therefore the state is estimated using the measurement $y$, in a way that reduces the effects of noise and disturbances. The process noise and the measurement noise both are assumed to have normal distributions whose probability is subject to the process noise covariance and the measurement noise covariance respectively seen in the equations below.

$$p(w) \sim \mathcal{N}(0, Q) \tag{6.3}$$

$$p(v) \sim \mathcal{N}(0, R) \tag{6.4}$$

By using the Certainty Equivalence Principle [89] it can be shown that the optimal control law for a noise-free deterministic problem is the same as the optimal control law for a noisy stochastic control problem. Then the exact linear system represented above can be represented as the estimated noise-free linear system

$$\dot{\tilde{x}} = A\tilde{x} + Bu \tag{6.5}$$

$$y = C\tilde{x} + Du \tag{6.6}$$

where $\tilde{x}$ is the estimation of the state such that the estimation error, $x_e$. Where

$$x_e = x - \tilde{x} \tag{6.7}$$

is optimally minimized. The estimated system shown in equations (6.5) and (6.6) is then used to determine the control input that yields optimal performance.

Practical microprocessor based applications use the discrete form of the system as

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{6.8}$$

$$y_k = Cx_k + v_k \tag{6.9}$$

where at the current time instant *k*, the state vector is $x_k$, and the measurement is $y_{k.}$ The measurement and process noises are $v_k$ and $w_k$ respectively. The Kalaman filter can be developed to estimate the state vector $\hat{x}_k$ iteratively as where the initial time step state estimation is

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{6.10}$$

and the current state estimation is the linear combination

$$\hat{x}_k = \hat{x}_k^- + K(y_k - C\hat{x}_k^-) \tag{6.11}$$

Where *K*, is known as a Kalman gain. The initial and updated errors are calculated as

$$e_k^- = x_k - \hat{x}_k^- \tag{6.12}$$

and

$$e_k = x_k - \hat{x}_k \tag{6.13}$$

respectively with the initial and updated error covariances become

$$P_k^- = \mathcal{E}[e_k^- e_k^{-T}] = AP_{k-1}A^T + Q \tag{6.14}$$

and

$$P_k = \mathcal{E}[e_k e_k^T] = (I - KC)P_k^- \tag{6.15}$$

where $\mathcal{E}$ is the normal expectation. It is seen that $P_k^-$, the initial error, is a a solution of Lyapunov stability equations, and the Kalman gain, *K*, is computed from these two equations as,

$$K = P_k^- C^T (C P_k^- C^T + R)^{-1} \tag{6.16}$$

This set of equations forms the centralized discretized Kalman filter. The initial state estimation and covariance are found using equations (6.10) and (6.14) respectively. These initial estimates are then used to find the Kalman gain in equation (6.16), which determines the updated the state and covariances in equations (6.14) and (6.15) respectively. The filter then runs onto the next time step and is able to complete real time filtering and fusion.

## 6.3. Decentralized Kalman Fusion

Some complex sensor systems are decentralized to prevent failure as seen in [90]–[92]. In decentralized Kalman fusion *i* separate sensor systems are fused together independently where

$$x_k^{(i)} = A^{(i)} x_{k-1}^{(i)} + B^{(i)} u_{k-1}^{(i)} + w_{k-1}^{(i)} \tag{6.17}$$

$$y_k = C^{(i)} x_k^{(i)} + v_k^{(i)} \tag{6.18}$$

In a set of four sensor systems the previous equations become

$$\begin{bmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \end{bmatrix} = \begin{bmatrix} A^{(1)} & 0 & 0 & 0 \\ 0 & A^{(2)} & 0 & 0 \\ 0 & 0 & A^{(3)} & 0 \\ 0 & 0 & 0 & A^{(4)} \end{bmatrix} \begin{bmatrix} x_{k-1}^{(1)} \\ x_{k-1}^{(2)} \\ x_{k-1}^{(3)} \\ x_{k-1}^{(4)} \end{bmatrix} + \cdots$$

$$\cdots \begin{bmatrix} B^{(1)} & 0 & 0 & 0 \\ 0 & B^{(2)} & 0 & 0 \\ 0 & 0 & B^{(3)} & 0 \\ 0 & 0 & 0 & B^{(4)} \end{bmatrix} \begin{bmatrix} u_{k-1}^{(1)} \\ u_{k-1}^{(2)} \\ u_{k-1}^{(3)} \\ u_{k-1}^{(4)} \end{bmatrix} + \begin{bmatrix} w_{k-1}^{(1)} & 0 & 0 & 0 \\ 0 & w_{k-1}^{(2)} & 0 & 0 \\ 0 & 0 & w_{k-1}^{(3)} & 0 \\ 0 & 0 & 0 & w_{k-1}^{(4)} \end{bmatrix} \tag{6.19}$$

and

$$\begin{bmatrix} y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \end{bmatrix} = \begin{bmatrix} C^{(1)} & 0 & 0 & 0 \\ 0 & C^{(2)} & 0 & 0 \\ 0 & 0 & C^{(3)} & 0 \\ 0 & 0 & 0 & C^{(4)} \end{bmatrix} \begin{bmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \\ x_k^{(4)} \end{bmatrix} + \begin{bmatrix} v_k^{(1)} \\ v_k^{(2)} \\ v_k^{(3)} \\ v_k^{(4)} \end{bmatrix} \tag{6.20}$$

The initial state estimate time equations then become

$$\begin{bmatrix} \hat{x}_k^{-(1)} \\ \hat{x}_k^{-(2)} \\ \hat{x}_k^{-(3)} \\ \hat{x}_k^{-(4)} \end{bmatrix} = \begin{bmatrix} A^{(1)} & 0 & 0 & 0 \\ 0 & A^{(2)} & 0 & 0 \\ 0 & 0 & A^{(3)} & 0 \\ 0 & 0 & 0 & A^{(4)} \end{bmatrix} \begin{bmatrix} \hat{x}_{k-1}^{(1)} \\ \hat{x}_{k-1}^{(2)} \\ \hat{x}_{k-1}^{(3)} \\ \hat{x}_{k-1}^{(4)} \end{bmatrix} + \begin{bmatrix} B^{(1)} & 0 & 0 & 0 \\ 0 & B^{(2)} & 0 & 0 \\ 0 & 0 & B^{(3)} & 0 \\ 0 & 0 & 0 & B^{(4)} \end{bmatrix} \begin{bmatrix} u_{k-1}^{(1)} \\ u_{k-1}^{(2)} \\ u_{k-1}^{(3)} \\ u_{k-1}^{(4)} \end{bmatrix} \tag{6.21}$$

$$\begin{bmatrix} P_k^{-(1)} & 0 & 0 & 0 \\ 0 & P_k^{-(2)} & 0 & 0 \\ 0 & 0 & P_k^{-(3)} & 0 \\ 0 & 0 & 0 & P_k^{-(4)} \end{bmatrix} = \begin{bmatrix} A^{(1)} & 0 & 0 & 0 \\ 0 & A^{(2)} & 0 & 0 \\ 0 & 0 & A^{(3)} & 0 \\ 0 & 0 & 0 & A^{(4)} \end{bmatrix} \times \ldots$$

$$\ldots \begin{bmatrix} P_{k-1}^{(1)} & 0 & 0 & 0 \\ 0 & P_{k-1}^{(2)} & 0 & 0 \\ 0 & 0 & P_{k-1}^{(3)} & 0 \\ 0 & 0 & 0 & P_{k-1}^{(4)} \end{bmatrix} \begin{bmatrix} A^{(1)} & 0 & 0 & 0 \\ 0 & A^{(2)} & 0 & 0 \\ 0 & 0 & A^{(3)} & 0 \\ 0 & 0 & 0 & A^{(4)} \end{bmatrix}^T + \begin{bmatrix} Q^{(1)} \\ Q^{(2)} \\ Q^{(3)} \\ Q^{(4)} \end{bmatrix} \tag{6.22}$$

The Kalman gain is then found where

$$\begin{bmatrix} K^{(1)} \\ K^{(2)} \\ K^{(3)} \\ K^{(4)} \end{bmatrix} = \begin{bmatrix} P_k^{-(1)} & 0 & 0 & 0 \\ 0 & P_k^{-(2)} & 0 & 0 \\ 0 & 0 & P_k^{-(3)} & 0 \\ 0 & 0 & 0 & P_k^{-(4)} \end{bmatrix} \begin{bmatrix} C^{(1)} & 0 & 0 & 0 \\ 0 & C^{(2)} & 0 & 0 \\ 0 & 0 & C^{(3)} & 0 \\ 0 & 0 & 0 & C^{(4)} \end{bmatrix}^T \times \ldots$$

$$\ldots \left( \begin{bmatrix} C^{(1)} & 0 & 0 & 0 \\ 0 & C^{(2)} & 0 & 0 \\ 0 & 0 & C^{(3)} & 0 \\ 0 & 0 & 0 & C^{(4)} \end{bmatrix} \begin{bmatrix} P_k^{-(1)} & 0 & 0 & 0 \\ 0 & P_k^{-(2)} & 0 & 0 \\ 0 & 0 & P_k^{-(3)} & 0 \\ 0 & 0 & 0 & P_k^{-(4)} \end{bmatrix} \ldots \right.$$

$$\dots\times\begin{bmatrix}C^{(1)} & 0 & 0 & 0\\ 0 & C^{(2)} & 0 & 0\\ 0 & 0 & C^{(3)} & 0\\ 0 & 0 & 0 & C^{(4)}\end{bmatrix}^T + \begin{bmatrix}R^{(1)}\\ R^{(2)}\\ R^{(3)}\\ R^{(4)}\end{bmatrix}\right)^{-1} \tag{6.23}$$

which then used to update the state equations as

$$\cdot\begin{bmatrix}P_k^{(1)} & 0 & 0 & 0\\ 0 & P_k^{(2)} & 0 & 0\\ 0 & 0 & P_k^{(3)} & 0\\ 0 & 0 & 0 & P_k^{(4)}\end{bmatrix} = \left(\begin{bmatrix}I^{(1)} & 0 & 0 & 0\\ 0 & I^{(2)} & 0 & 0\\ 0 & 0 & I^{(3)} & 0\\ 0 & 0 & 0 & I^{(4)}\end{bmatrix} - \begin{bmatrix}K^{(1)} & 0 & 0 & 0\\ 0 & K^{(2)} & 0 & 0\\ 0 & 0 & K^{(3)} & 0\\ 0 & 0 & 0 & K^{(4)}\end{bmatrix}\dots\right.$$

$$\left.\dots\times\begin{bmatrix}C^{(1)} & 0 & 0 & 0\\ 0 & C^{(2)} & 0 & 0\\ 0 & 0 & C^{(3)} & 0\\ 0 & 0 & 0 & C^{(4)}\end{bmatrix}\right)\begin{bmatrix}P_k^{-(1)} & 0 & 0 & 0\\ 0 & P_k^{-(2)} & 0 & 0\\ 0 & 0 & P_k^{-(3)} & 0\\ 0 & 0 & 0 & P_k^{-(4)}\end{bmatrix} \tag{6.24}$$

$$\begin{bmatrix}\hat{x}_k^{(1)}\\ \hat{x}_k^{(2)}\\ \hat{x}_k^{(3)}\\ \hat{x}_k^{(4)}\end{bmatrix} = \begin{bmatrix}\hat{x}_k^{-(1)}\\ \hat{x}_k^{-(2)}\\ \hat{x}_k^{-(3)}\\ \hat{x}_k^{-(4)}\end{bmatrix} + \begin{bmatrix}K^{(1)}\\ K^{(2)}\\ K^{(3)}\\ K^{(4)}\end{bmatrix}\left(\begin{bmatrix}y_k^{(1)}\\ y_k^{(2)}\\ y_k^{(3)}\\ y_k^{(4)}\end{bmatrix} - \begin{bmatrix}C^{(1)} & 0 & 0 & 0\\ 0 & C^{(2)} & 0 & 0\\ 0 & 0 & C^{(3)} & 0\\ 0 & 0 & 0 & C^{(4)}\end{bmatrix}\begin{bmatrix}\hat{x}_k^{-(1)}\\ \hat{x}_k^{-(2)}\\ \hat{x}_k^{-(3)}\\ \hat{x}_k^{-(4)}\end{bmatrix}\right) \tag{6.25}$$

Decentralized Kalman fusion has the advantage that if one sensor system fails the other systems can be effective. The main drawback is the computational cost in making such large matrices.

### 6.3.1. Robust Kalman Filtering

The standard Kalman filter solutions discussed above require the model of the system to be known and be as accurate as possible. The problem in modelling most dynamic systems is that certain assumptions must be made to simplify the modelling process, but such assumptions also introduce certain amounts of inaccuracies that can affect the performance of the filter. In the presence of model uncertainties, a robust Kalman filter is required to perform all the filtering and fusion operations subject to some limits on the uncertainty size. Robust Kalman filtering has been studied extensively and many solutions with many different unique applications have been well studied in [93]–[96]. At base of all the filters they seek to find a state estimate, $\tilde{x}$, based on the measurement vector, $y_k$. Many solution seen in the literature can be computationally expensive and due to the limited computational power of the microcontrollers used in this study the solutions with lower computational costs were considered. Specifically this study considers an uncertain system

$$x_k = [A_k + \Delta_A]x_k + [B_k + \Delta_B]u_k + w_k \tag{6.26}$$

$$y_k = [C_k + \Delta_C]x_k + v_k \tag{6.27}$$

The terms $\Delta_A, \Delta_B$, and $\Delta_C$ represent the uncertainties caused by the linearization and simplification of the dynamic and kinematic equations. The state vector can then be expressed in terms of the nominal value, $x_{n_k}$ , from the modeled dynamics and the uncertain value, $\Delta x_k$, from the ignored dynamics as

$$x_k = x_{n_k} + \Delta x_k \tag{6.28}$$

which satisfy

$$\Delta x_{k+1} = \Delta_A \Delta x_k + \Delta_B u_k \tag{6.29}$$

$$\Delta y_k = \Delta_C \Delta x_k \tag{6.30}$$

or simply

$$\begin{bmatrix} \Delta x_{k+1} \\ \Delta y_k \end{bmatrix} = \begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ u_k \end{bmatrix} \tag{6.31}$$

The continual effects of the uncertainties will be kept in bound as long as

$$\varepsilon = \left\| \begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} \right\|_\infty < 1 \tag{6.32}$$

The uncertainty matrix above can also be represented as

$$\begin{bmatrix} \Delta_A & \Delta_B \\ \Delta_C & 0 \end{bmatrix} = \begin{bmatrix} A_x \\ A_Y \end{bmatrix} W [B_1 \quad B_2] \tag{6.33}$$

where $A_x, A_Y, B_1$ and $B_2$ are constant structural matrices and

$$W^T W \leq 1 \tag{6.34}$$

which are sought to make a robust filter of the form

$$\hat{x}_{k+1} = \Phi_k \hat{x}_k + \Gamma_k y_k \tag{6.35}$$

which is related to the solutions seen in [95], [96]. This solution finds the next state, $\hat{x}_{k+1}$ estimation by combing the current state estimation, $\hat{x}_k$, and current measurements, $y_k$. The estimator matrix $\Phi_k$ is used to manipulate the current state and, $\Gamma_k$, is used to optimally fuse the measurements to the state vector. Both these matrices are computed from the solution to the filter.

The objective of the filter is to bind the covariance of the estimator error such that

$$\mathcal{E}[e_k^T e_k] \leq \gamma \tag{6.36}$$

where $\gamma$ is some real constant and

$$e_k = x_k - \hat{x}_k \tag{6.37}$$

The estimator error can be expressed as

$$e_{k+1} = x_{k+1} - \hat{x}_{k+1} \tag{6.38}$$

$$= \{[A_k + \Delta_A]x_k + [B_k + \Delta_B]u_k + w_k\} - \{\Phi_k \hat{x}_k + \Gamma_k([C_k + \Delta_H]x_k + v_k)\} \tag{6.39}$$

$$= \{[A_k + \Delta_A] - \Gamma_k[C_k + \Delta_C] - \Phi_k\}x_k - [B_k + \Delta_B]u_k + \Phi_k e_k + w_k - \Gamma_k v_k \tag{6.40}$$

$$= \Phi_k e_k + \left[\tilde{A}_k + \tilde{E}_k W B_1\right]x_k + [B_k + \tilde{E}_k W B_2]u_k + w_k - \Gamma_k v_k \tag{6.41}$$

where

$$\tilde{A}_k = A_k - \Gamma_k C_k - \Phi_k \tag{6.42}$$

$$\tilde{E}_k = A_x - \Gamma_k A_{\mathrm{Y}} \tag{6.43}$$

For the filter to work the estimator needs to be stable. If the state estimation and error evolution from time interval *k* to *k*+1 is represented in the absence of a control input as

$$\begin{bmatrix} x_{k+1} \\ e_{k+1} \end{bmatrix} = \begin{bmatrix} A_k + A_x W B_1 & 0 \\ \tilde{A}_k + \tilde{E}_k W B_1 & \Phi_k \end{bmatrix} \begin{bmatrix} x_k \\ e_k \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & -\Gamma_k \end{bmatrix} \begin{bmatrix} w_k \\ v_k \end{bmatrix} \tag{6.44}$$

$$e_k = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ e_k \end{bmatrix} \tag{6.45}$$

Then the estimator is only stable when the solution $X = X^T \leq 0$ to the discrete Lyapunov equation

$$\begin{bmatrix} A_k + A_x W B_1 & 0 \\ \tilde{A}_k + \tilde{E}_k W B_1 & \Phi_k \end{bmatrix}^T X \begin{bmatrix} A_k + A_x W B_1 & 0 \\ \tilde{A}_k + \tilde{E}_k W B_1 & \Phi_k \end{bmatrix} - X \leq 0 \tag{6.46}$$

exists for all allowable uncertainties [74], [97]. Details of the Lyapunov equations are introduced in Chapter 6. From the bounded real lemma it can be shown that the evolution of estimator errors can be bounded by $\gamma$>0 such that

$$\varepsilon = \|e_k\|_\infty < \gamma \tag{6.47}$$

if the discrete algebraic Riccati equation

$$\begin{bmatrix} A_k & 0 \\ \tilde{A}_k & \Phi_k \end{bmatrix}^T Z \begin{bmatrix} A_k & 0 \\ \tilde{A}_k & \Phi_k \end{bmatrix} - Z + \left( \begin{bmatrix} A_k & 0 \\ \tilde{A}_k & \Phi_k \end{bmatrix}^T Z \begin{bmatrix} A_x W B_1 \\ \tilde{E}_k W B_1 \end{bmatrix} \right) \ldots$$

$$\ldots \times \left( \gamma^2 I - \begin{bmatrix} A_x W B_1 \\ \tilde{E}_k W B_1 \end{bmatrix}^T Z \begin{bmatrix} A_x W B_1 \\ \tilde{E}_k W B_1 \end{bmatrix} \right)^{-1} \left( \begin{bmatrix} A_k & 0 \\ \tilde{A}_k & \Phi_k \end{bmatrix}^T \begin{bmatrix} A_x W B_1 \\ \tilde{E}_k W B_1 \end{bmatrix} \right) \leq 0 \tag{6.48}$$

has a solution $Z = Z^T$ for any $\gamma$>0 where

$$\left( \gamma^2 I - \begin{bmatrix} A_x W B_1 \\ \tilde{A}_k W B_1 \end{bmatrix}^T Z \begin{bmatrix} A_x W B_1 \\ \tilde{A}_k W B_1 \end{bmatrix} \right) \leq 0 \tag{6.49}$$

#### 6.3.1.1. Specific Formulation

Initially the solution to the Kalman filter presented above was solved by partitioning the solutions to the Lyapunov and Riccati equations, *X* and *Z* as

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix}, Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{12}^T & Z_{22} \end{bmatrix} \tag{6.50}$$

The fully expanded equation based on the partition could be simplified by using the positive definite covariances *Q* and *R* and would result in two decoupled discrete Riccati equations that can be formed recursively as similar to the ones in [97]–[99] as

$$\Sigma_{k+1} = A_k \Sigma_k {A_k}^T + A_k \Sigma_k B_1^T (\gamma^2 I - B_1 \Sigma_k B_1^T)^{-1} B_1 \Sigma_k {A_k}^T + \frac{1}{\gamma} A_x A_x^T + Q \tag{6.51}$$

$$\Upsilon_{k+1} = A_k \Xi_k {A_k}^T + M_k \mathrm{N}_k^{-1} M_k^T + + \frac{1}{\gamma} A_x A_x^T + Q \tag{6.52}$$

where

$$\Xi_k = (\Upsilon_k^{-1} - \gamma B_1^T \mathrm{B}_1)^{-1} \tag{6.53}$$

$$\mathrm{M}_k = R + \frac{1}{\gamma} A_\Upsilon A_\Upsilon^T + A_k \Xi_k A_k^T \tag{6.54}$$

$$\mathrm{N}_k = A_k \Xi_k {C_k}^T + \frac{1}{\gamma} A_x A_\Upsilon^T \tag{6.55}$$

The Kalman matrices only depend on the solution to the second Riccati equation and are defined as

$$\Gamma_k = M_k N_k^{-1} \tag{6.56}$$

$$\Phi_k = [A_k - \Gamma_k C_k] \left[ I + \frac{1}{\gamma} \Upsilon_k B_1^T (I - \gamma B_1 \Upsilon_k B_1^T) \right] \tag{6.57}$$

The designer is left to find covariances $Q$ and $R$ as well as the uncertainty bounds $A_x, A_\Upsilon, B_1$ and $\gamma$. If there is no uncertainty in the linearization of the system, or $A_x = A_y = B_1 = 0$, this reduces to standard Kalman filter.

## 6.4. Combined Robust State Estimation and Robust Path Tracking Control

In implementation, as shown in the appended code, the robust estimator outlined above and the controller, shown in Chapter 5, are combined in seven steps:

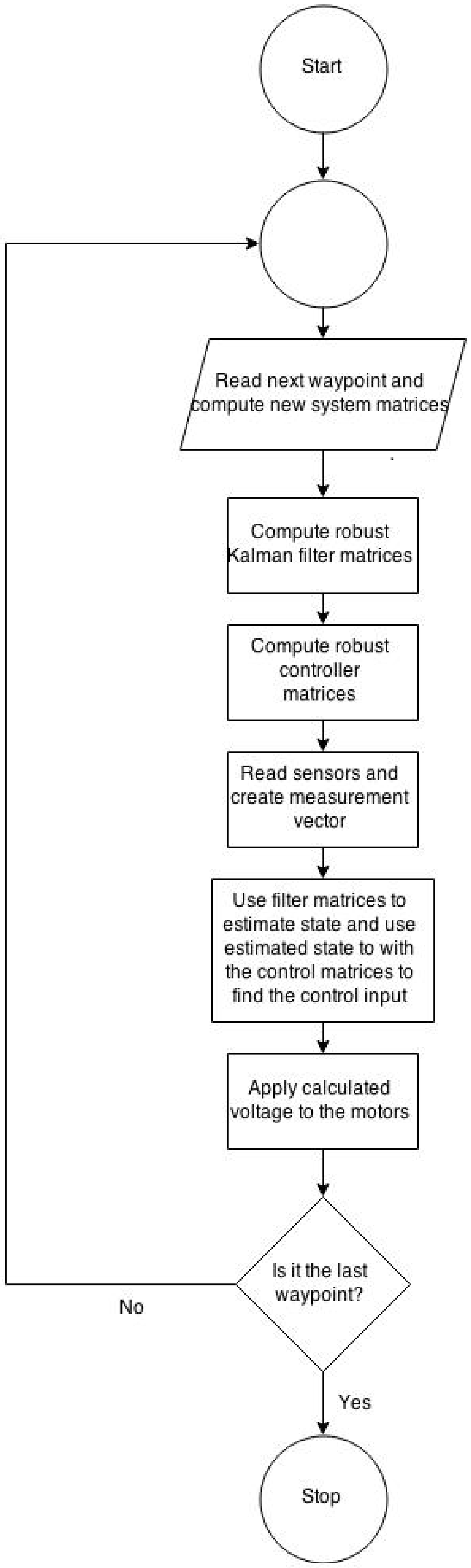

Start
Read next waypoint and compute new system matrices
Compute robust Kalman filter matrices
Compute robust controller matrices
Read sensors and create measurement vector
Use filter matrices to estimate state and use estimated state to with the control matrices to find the control input
Apply calculated voltage to the motors
Is it the last waypoint?
No
Yes
Stop

Figure 6.1: Control flow chart

# CHAPTER 7. EXPERIMENTAL AND SIMULATION RESULTS

This chapter presents the results from the improved GPS algorithm, simulations of the dynamic model and experimentation of the combined control algorithm and filter.

## 7.1. GPS Algorithm

Chapter 3 explained the errors in the GPS systems that are amplified by the satellite selection used to find a position fix. Chapter 4 then goes into detail on the calculations used in the Copernicus II receiver for a satellite's ECEF coordinates. Chapter 4 then gave a background on the least-squares algorithm commonly used to estimate the system of equations used in GPS trilaterations and explains other numerical methods that may improve the solution for the estimation to the set of equations. This chapter details the selection method and used and the results for that selection method in the least squares, Newton's, and the BFGS methods for estimation of the solution. These are all compared to the coordinates given by the Copernicus II receiver.

### 7.1.1. GPS Improvement Methods

As noted in Chapter 3 most of the signals for the satellites have a strength that in the range of 2 to 4. In this experiment the satellite selection will be limited to the signal strengths between 2 and 4 in the least squares, Newton's, and BFGS methods for estimating a solution to a set of equations. It is hypothesized that by limiting the selection of satellites to a strength of 2 to 4 the signal strength will become more precise even if the solution becomes less accurate. It is also hypothesized that Newton's and the BFGS methods will improve the precision of the trilateration because they ignore less terms leading to less approximation. The program developed to test the GPS algorithm is in APPENDIX A.

### 7.1.2. GPS Results

While attempting to achieve a position fix from a custom selection of satellite signals it was noted that the calculated satellite positions were incorrect. The satellite positions were calculated as seen in Chapter 4 from the data sent from the Copernicus II receiver and compared to the expected position based on the two line element (TLE) files downloaded by [100] and then converted to satellite positions by [101]. A sample of the expected position versus the actual position can be seen in Table 7.1.

Table 7.1: Satellite ECEF coordinates

| | | Calculated from Receiver | | | Calculated from TLE | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | X | Y | Z | X | Y | Z | Error | *Uk* |
| PRN 4 | 8:44 AM | -15459101 | 1969198 | 21176367 | -15436979 | 2120922 | 21179065 | 7.71 | 1.504 |
| | 8:48 AM | -15483504 | 1265138 | 21217767 | -15467267 | 1455204 | 21218911 | 15.02 | 1.541 |
| | 8:53 AM | -15532920 | 217814 | 21226803 | -15523218 | 534195 | 21229892 | 145.25 | 1.595 |
| PRN 14 | 8:44 AM | -10158587 | 11132505 | -21640717 | -11427077 | -19492172 | 14315491 | 321.62 | -1.491 |
| | 8:48 AM | -9522484 | 11613871 | -21674408 | -11040759 | -19290712 | 14885974 | 315.46 | -1.515 |
| | 8:53 AM | -7698498 | 12837871 | -21702095 | -10478473 | -19006647 | 15645628 | 304.05 | -1.581 |

The position calculations from the TLE files should not be considered entirely accurate because time used to calculate the TLE position will always be slightly different than the actual time the GPS receiver actually received the satellite information for the satellite calculation. The TLE positions can however provide an estimate of the general vicinity the satellite should be located. As can be seen in Table 7.1, sometimes the two calculated positions for a given satellite are within 5% even, many times the satellite positions are extremely different.

## 7.2. Control and Robust Filter Performance Results Obtained on the Developed Model

These results will be broken into two sections: results based on model and control simulation only, without using the developed robust filter, and experimental results on the vehicle combining all components

### 7.2.1. Simulation Results

A MATLAB code used in simulating the developed model using the developed controller and the robust filter is shown in APPENIX B. Since there were no sensors to read, the robust filter essentially runs without producing anything. So these results will only show the performance of the vehicle and of the controller only. Three types of paths were evaluated as shown in Table 7.2 and the performance was evaluated based on the average root-mean-squared (rms) tracking error per step,

$$e_{rms} = \frac{\sqrt{\Sigma[(X_r - x_p)^2 + (Y_r - y_p)^2]}}{N_{STEPS}} \tag{7.1}$$

Where $(X_r, Y_r)$ are the simulated coordinates tracked by the robot and $(x_p, y_p)$ are the desired path coordinates

Table 7.2: Simulation performance

| Path | RMS tracking error per step (m) |
|---|---|
| Semicircle | 0.8512 |
| Zigzag | 0.4824 |
| Sinusoid | 1.4228 |

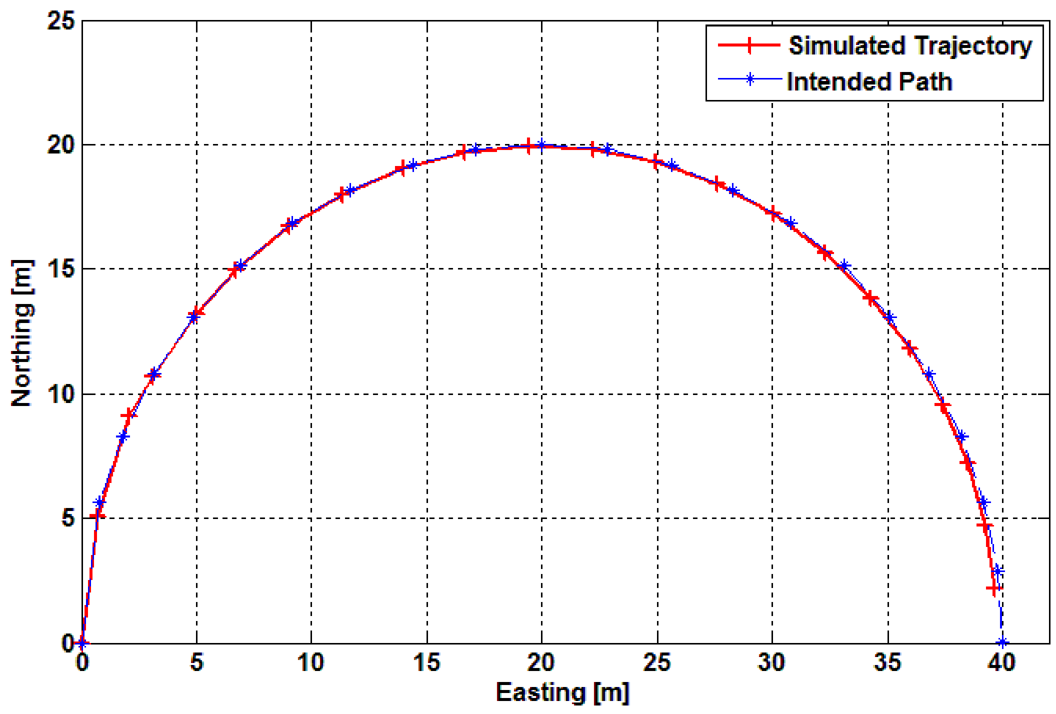


Figure 7.1: Semicircle path

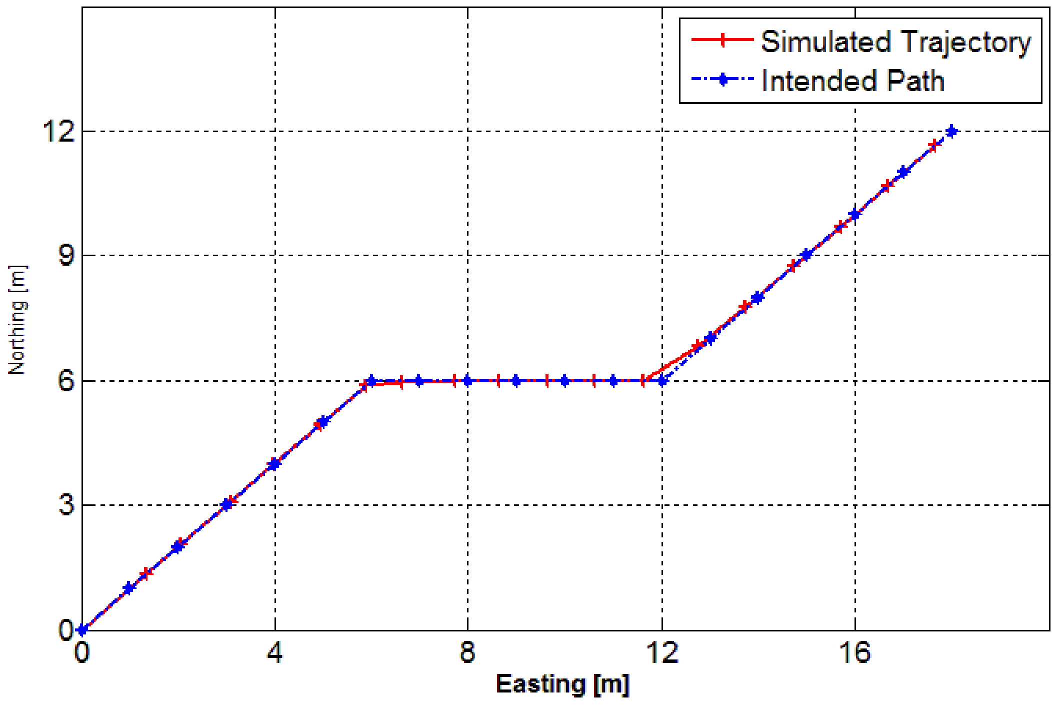


Figure 7.2: Zigzag path

Although the numerical simulation results for the sinusoid path shows failure to turn from decreasing Northing to increasing Northing as shown in Figure 7.4, the simulation was still able to fairly capture the behavior within three quarters of the cycle before that failure happened. For the part where the path was tracked sufficiently tracked as shown in Figure 7.4, the average rms tracking error was within the robot size. Failure to change the northing direction from negative to positive is not attributed to the model or the control itself but rather the way trigonometric functions were handled in the numerical simulator.

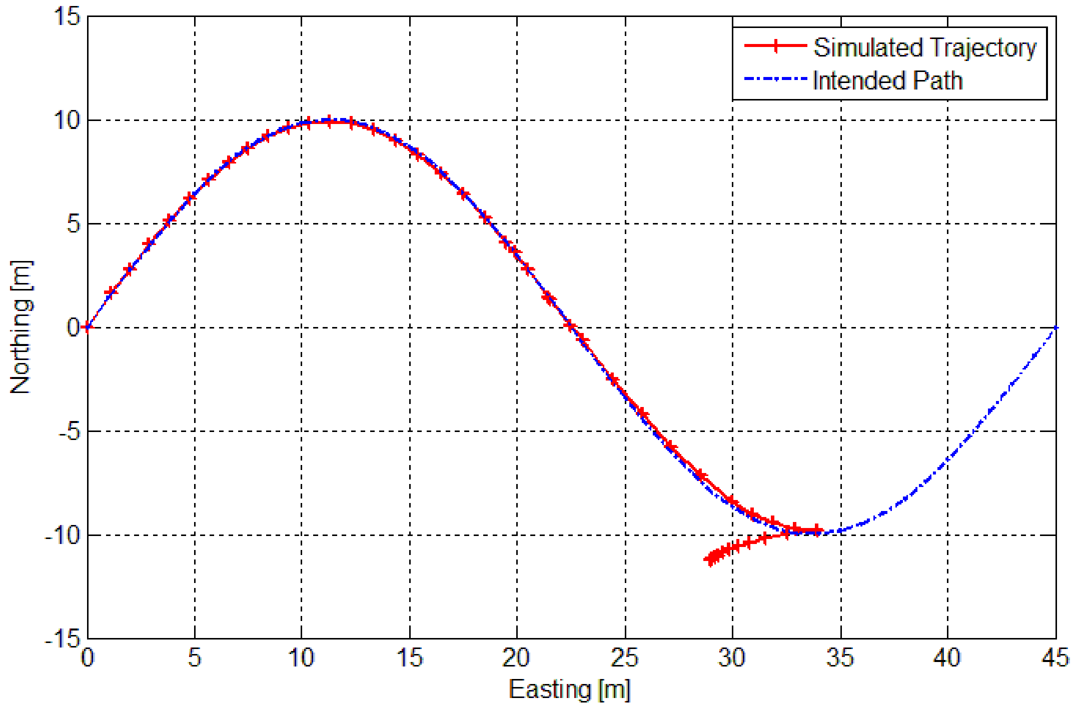


Figure 7.3: Full sinusoid path

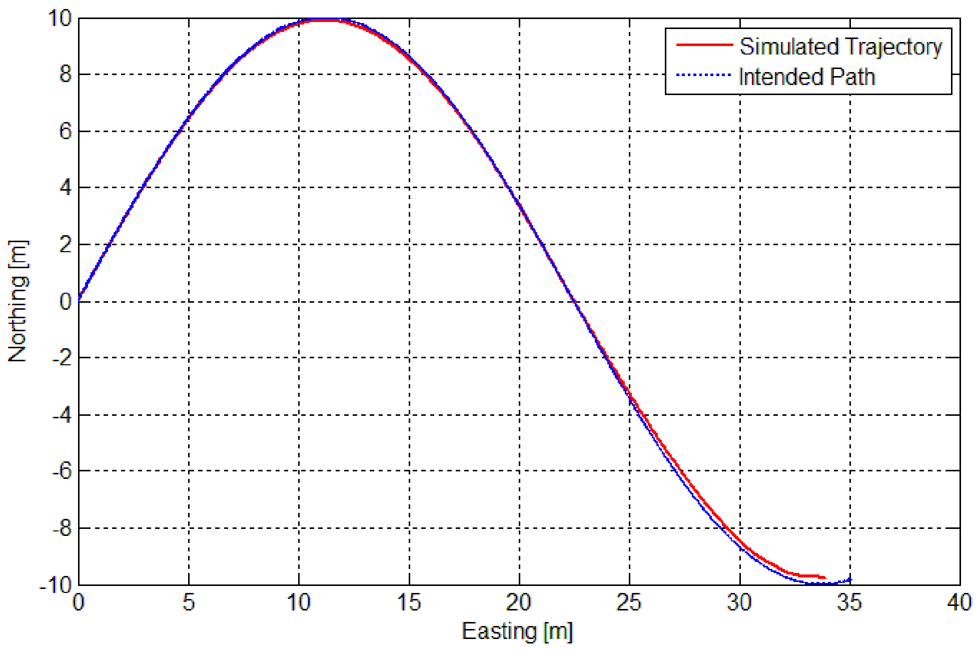


Figure 7.4: Partial sinusoid path

### 7.2.2. Control and Filter Experimentation

The control experiments were conducted on THUNDAR the autonomous snowplow which was designed in [102] and is pictured below.

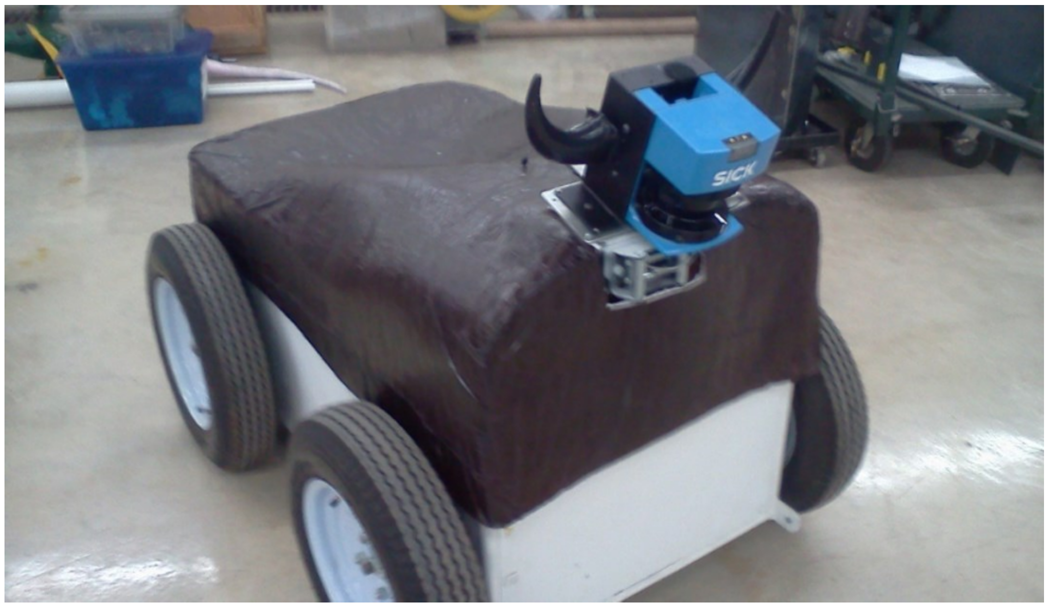


Figure 7.5: THUNDAR the autonomous snow plow

### 7.2.3. Experimental Vehicle

The vehicle was designed to operate with the left side wheels and the right side wheels to be controlled independently as seen in Figure 7.6. Each side is driven by a Motenergy ME0708 DC motor which is capable of supplying 4.5 KW at 24V.

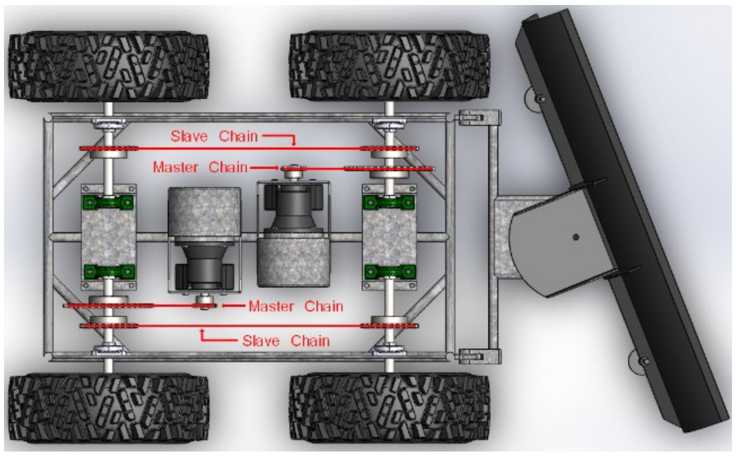


Figure 7.6: Vehicle drivetrain [102]

Each motor has a total of a 20:1 speed reduction to each wheel through a Baldor GCF5X02BB 5:1 gear reduction and then an additional 4:1 speed reduction through chains and gears as shown in Figure 7.7.

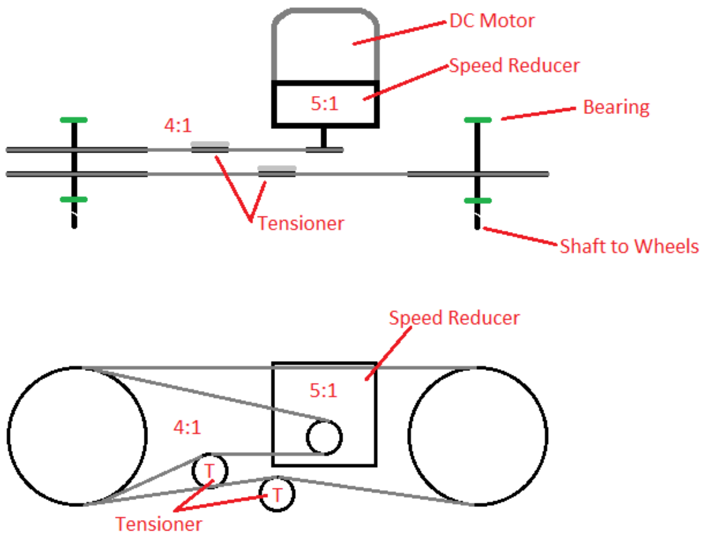


Figure 7.7: Motor reduction [102]

The circuitry is split into two main sections, one part which drives the high voltage DC motors and one low voltage circuit which contains the sensors and the microcontrollers necessary to conduct the control algorithm.

On the high voltage side a RoboteQ VDC2450 motor driver takes PWM signals from the low voltage circuit and converts it to the high voltage PWM necessary to drive the 24V wheel DC motors. A RoboteQ LDC2203C motor driver is drive the 12V winch and linear actuator used on the snowplow blade based on signals from the low voltage circuit. The high voltage circuit contains two safety kill switches shown in Figure 7.8.

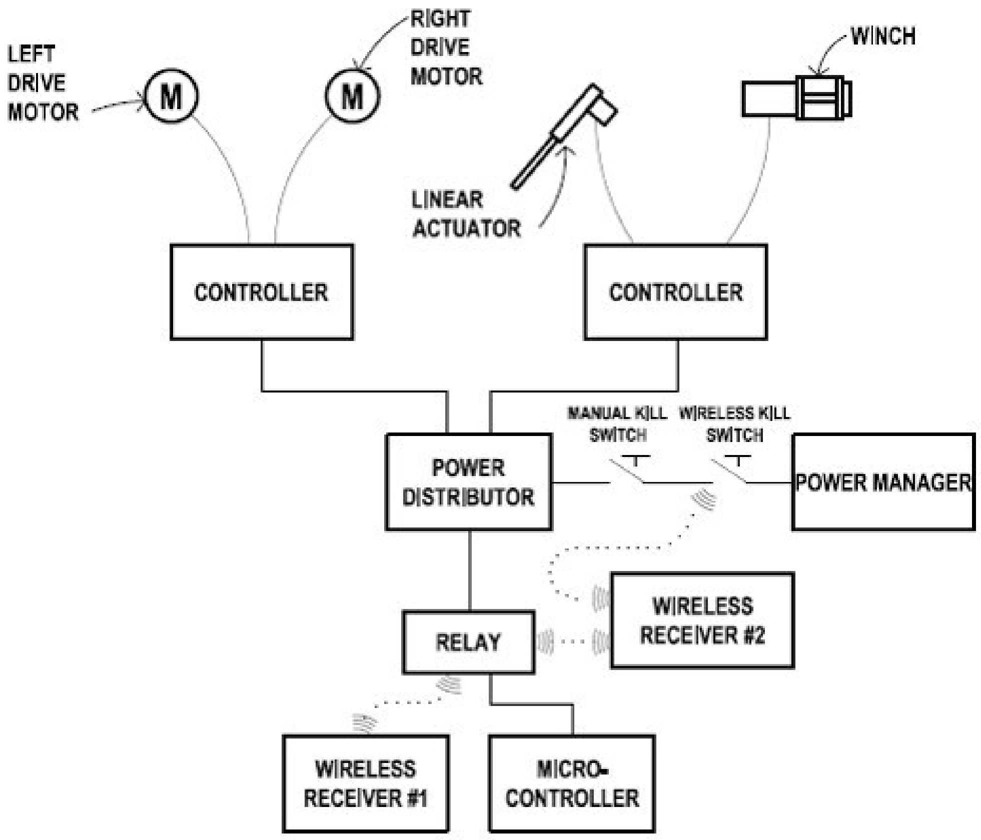


Figure 7.8: High voltage circuit [102]

There is one manual safety switch and one remote safety switch. The high voltage side also contains the remote switch which sends a 12V signal to the low voltage board to switch between autonomous and remote control modes.

The low voltage side consists of a circuit board which was developed for this study and is shown in Figure 7.9 and pictured in Figure 7.10. Further detail of the circuit is shown in the APPENDIX C.

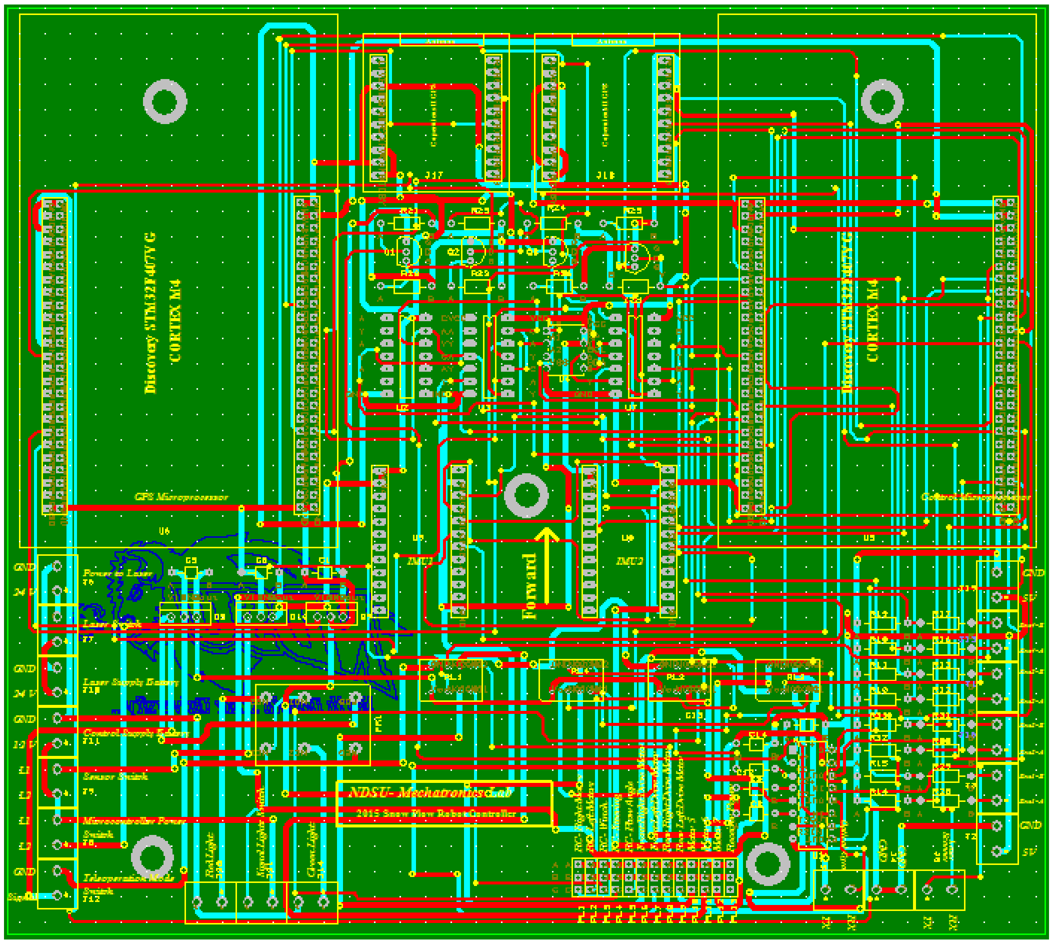


Figure 7.9: Circuit board diagram

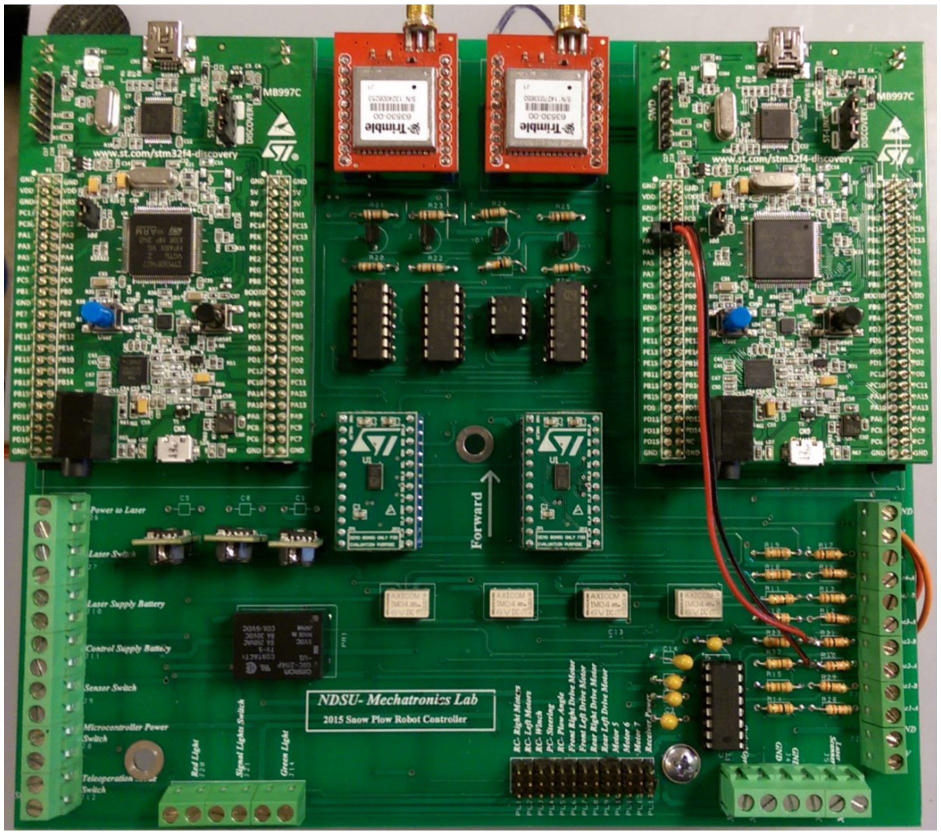


Figure 7.10: Picture of circuit board

The sensors used on the board and which measurements they correspond to are shown below in Table 7.3.

Table 7.3: Measurement vector and corresponding sensors

| Measurement Vector | Sensor |
|---|---|
| $N_l$ | Yumo E6B2-CWZ3E Encoder |
| $N_r$ | Yumo E6B2-CWZ3E Encoder |
| $a_{x1}$ | STEVAL-MKI123V1 IMU |
| $a_{x2}$ | STEVAL-MKI123V1 IMU |
| $\omega_{z1}$ | STEVAL-MKI123V1 IMU |
| $\omega_{z2}$ | STEVAL-MKI123V1 IMU |
| $X$ | Copernicus II GPS |
| $Y$ | Copernicus II GPS |

The sensors used correspond the sensor measurements required for the filtering in Chapter 5 and the control algorithm in Chapter 6. It is important to note that the encoders are only connect to the circuit board, the encoders themselves are mounted directly to one wheel on each side of the robot. There are two Coretex M4 microcontrollers on board. One is used to compute the custom GPS algorithm and the other is used to process the other sensors and carry out the control program. The code used for the control program is shown in APPENDIX D.

### 7.2.4. Experimental Results

In experimentation the control algorithm was not able to reliably track the desired path of the robot because most of the time the controller would saturate and fail to respond to the desired path waypoints. The robot was experimented with several times with different definitions of waypoints, but all the time the system would fail. The research took time to reexamine the whole control system and found that while the controller part (equations (5.112) through (5.117)) had no problems, i.e. if it got the proper state vector, then it would respond well as desired. The problem was found in the filter that was formulated in Chapter 5. This filter exhibited weak convergence in calculating the Ricatti equation (equations (6.35) and (6.53) through (6.57) and often would result in singular matrices that would force the estimated state vector to be infinity. A sample of the state vector, raw sensor measurements and control vector can be seen in Table 7.4, Table 7.5, and Table 7.6 respectively.

Table 7.4: Raw measurement vector

| Time(ms) | Ax1 | Gz1 | Ax2 | Gz2 | X | Y | L Enc | R Enc |
|---|---|---|---|---|---|---|---|---|
| 28092.29 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -2.6 | 3 | 0 | 0 |
| 30173.63 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -2.9 | 4 | 0 | 0 |
| 31688.04 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 32734.11 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 33772.35 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 34810.46 | -0.00508 | 0.0000 | -9160.0 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 36360.1 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 37399.13 | -0.00762 | 0.0012 | -9160.0 | 0.0000 | -3.8 | 5 | 0 | 0 |
| 38438.26 | -0.01271 | 0.0000 | -8910.2 | 0.0000 | -6.5 | 5 | 0 | 0 |
| 40034.88 | 0.00000 | 0.0000 | 0.0 | 0.0000 | -6.5 | 5 | 0 | 0 |
| 41651.49 | -0.01017 | 0.0000 | -8660.4 | 0.0000 | -6.5 | 5 | 0 | 0 |
| 42699.6 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -6.5 | 5 | 0 | 0 |
| 43745.71 | 84.22366 | 0.0300 | -5354.0 | 0.0312 | -4.8 | 6.5 | 358 | 511 |
| 45022.32 | 0.00000 | 0.0000 | 83.3 | 0.0000 | -4.8 | 6.5 | 859 | 828 |
| 46272.16 | -75.5806 | -75.446 | 721.0 | -75.142 | -4.8 | 6.5 | -255 | 198 |
| 47380.48 | -0.65565 | 0.0012 | -13.7 | 0.0012 | -4.8 | 6.5 | -189 | 153 |
| 48916.06 | 0.00000 | 0.0000 | -82.6 | 0.0000 | -4.8 | 6.5 | 1188 | 1158 |
| 49957.26 | -0.00508 | 0.0336 | -4714.0 | 0.0240 | -4.8 | 6.5 | 1896 | 2084 |
| 51279.8 | 0.00000 | 0.0000 | 0.7 | 0.0000 | -4.8 | 6.5 | 905 | 850 |
| 52552.92 | 0.65311 | 0.0012 | -83.3 | 0.0012 | 2.0 | 2.5 | -217 | 157 |
| 53810.78 | 0.00000 | 0.0012 | 5.9 | 0.0012 | 2.0 | 2.5 | -233 | 155 |
| 55151.99 | 21.47649 | 0.0504 | -1869.8 | 0.0492 | 2.0 | 2.5 | 5 | 0 |
| 56197.21 | 0.00000 | 0.0000 | -83.3 | 0.0000 | 2.0 | 2.5 | 0 | 0 |
| 57242.67 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -18.1 | -8 | 0 | 0 |
| 58828.03 | 0.00000 | 0.0036 | -9409.9 | 0.0000 | -17.9 | -6.5 | 0 | 0 |
| 60451.8 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -17.9 | -6.5 | 0 | 0 |
| 62078.71 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -17.9 | -6.5 | 0 | 0 |
| 63123.75 | 0.00000 | 0.0000 | -83.3 | 0.0000 | -19.2 | -7.5 | 0 | 0 |

Table 7.5: Estimated state vector

| Time(ms) | vl | vr | vx | Ω | θ | XE | YE |
|---|---|---|---|---|---|---|---|
| 28092.29 | 0.0000 | 0.0000 | -0.000077 | 0 | 0 | 2.6249 | -2.9999 |
| 30173.63 | 0.0000 | 0.0000 | -0.000006 | 697.134 | 349.3932 | -57.5 | -168.958 |
| 31688.04 | 0.0000 | 0.0000 | -0.000062 | 0 | 0 | 3.8125 | -5.0001 |
| 32734.11 | 0.0000 | 0.0000 | -0.000027 | 0 | -8E-06 | 3.8125 | -4.99996 |
| 33772.35 | 0.0000 | 0.0000 | -0.000078 | 0 | 0 | 3.8124 | -4.9999 |
| 34810.46 | 0.0000 | 0.0000 | 0.059323 | 251.026 | -9885.08 | -3443 | 50277.79 |
| 36360.1 | 0.0000 | 0.0000 | -0.004451 | 0 | -6.4E-05 | 3.812 | -4.99259 |
| 37399.13 | 0.0000 | 0.0000 | -0.000046 | -8E-06 | -3E-06 | 3.8124 | -4.99995 |
| 38438.26 | 0.0000 | 0.0000 | -3.64E+08 | 1.8E+09 | 5.31E+08 | -4E+08 | 8.43E+08 |
| 40034.88 | 0.0000 | 0.0000 | 53851340 | 110.501 | 786488.6 | 4E+07 | -8E+07 |
| 41651.49 | 0.0000 | 0.0000 | -7244002 | -3E+15 | -3.2E+14 | -7E+14 | -5.6E+14 |
| 42699.6 | 0.0000 | 0.0000 | -77852240 | 4.9E+09 | -2.2E+09 | -1E+09 | 3.38E+08 |
| 43745.71 | 0.0172 | 0.0196 | 2.462E+12 | -1E+18 | -5.2E+17 | -5E+16 | 3.74E+17 |
| 45022.32 | 0.0687 | 0.0709 | -1.16E+15 | -2E+19 | -5.1E+18 | 2E+18 | -2.4E+17 |
| 46272.16 | 0.0503 | 0.0646 | 1.428E+09 | -1E+12 | 2.24E+12 | 1E+12 | 8.3E+11 |
| 47380.48 | -0.027 | 0.0219 | 23049.697 | -593.75 | 682.9127 | 26099 | 15915.79 |
| 48916.06 | 0.0180 | 0.0348 | -47165.89 | 1.4E+13 | 6.98E+12 | 4E+12 | -1E+13 |
| 49957.26 | 0.0912 | 0.1003 | -7.98E+11 | 1.1E+18 | 7.41E+17 | 3E+17 | 2.41E+17 |
| 51279.8 | 0.0633 | 0.0675 | -3.43E+11 | -8E+17 | -5.4E+17 | 6E+12 | 3E+17 |
| 52552.92 | 0.0526 | 0.0626 | -1.4E+18 | -1E+18 | 1.05E+18 | 2E+19 | -1.1E+18 |
| 53810.78 | -0.023 | 0.0168 | -1.06E+17 | -5E+21 | -6.7E+21 | 1E+21 | -2.8E+21 |
| 55151.99 | -0.010 | 0.0068 | -1.03E+16 | -4E+10 | -1.5E+14 | -6E+15 | 1.59E+16 |
| 56197.21 | 0.0003 | 0.0000 | 4.615E+12 | -2E+19 | -4E+19 | 2E+18 | -2E+18 |
| 57242.67 | 0.0000 | 0.0000 | 6.872E+09 | -608999 | 2.96E+08 | 9E+09 | -7.7E+09 |
| 58828.03 | 0.0000 | 0.0000 | -2.5E+08 | -33395 | -3655715 | -3E+08 | 2.68E+08 |
| 60451.8 | 0.0000 | 0.0000 | 47632308 | -576843 | 548882.4 | 6E+07 | -5.1E+07 |
| 62078.71 | 0.0000 | 0.0000 | 29255236 | 3.7E+14 | -1.4E+14 | 9E+13 | 1.88E+13 |
| 63123.75 | 0.0000 | 0.0000 | -11687678 | -2149.1 | -167909 | -2E+07 | -4359124 |

Table 7.6: Control output and algorithm convergence

| | Control Output | | Conrol Conv | | Filter Conv | |
|---|---|---|---|---|---|---|
| **Time(ms)** | **U1** | **U2** | **Itt_c** | **Er_c** | **Itt_f** | **Er_f** |
| 28092.29 | 0.43593 | 0.36253 | 3 | 1.844721 | 75 | 16587706368 |
| 30173.63 | 211.486 | -213.32 | 3 | 1.816249 | 75 | 1.58049E+11 |
| 31688.04 | 1.#QNAN0 | 1.#QNAN0 | 563 | 1.#QNAN0 | 75 | 18879340544 |
| 32734.11 | 0.68747 | 0.57421 | 3 | 1.82911 | 75 | 3.385E+11 |
| 33772.35 | 0.68747 | 0.57419 | 3 | 1.829115 | 75 | 5.94185E+11 |
| 34810.46 | -7311.7 | -316.52 | 3 | 1.829116 | 75 | 2.32962E+13 |
| 36360.1 | 1.#QNAN0 | 1.#QNAN0 | 600 | 1.#QNAN0 | 75 | 1703359616 |
| 37399.13 | 0.6878 | 0.57374 | 3 | 8.250623 | 75 | 4.37957E+13 |
| 38438.26 | 3.9E+08 | -5E+08 | 3 | 1.829115 | 75 | 7.54349E+11 |
| 40034.88 | 1.#QNAN0 | 1.#QNAN0 | 654 | 1.#QNAN0 | 75 | 54068789248 |
| 41651.49 | 1.#QNAN0 | 1.#QNAN0 | 682 | 1.#QNAN0 | 75 | 3.33201E+12 |
| 42699.6 | 1.#QNAN0 | 1.#QNAN0 | 8 | 1.#QNAN0 | 75 | 12194299904 |
| 43745.71 | 1.#QNAN0 | 1.#QNAN0 | 12 | 1.#QNAN0 | 75 | 4420327424 |
| 45022.32 | -1.#INF00 | 1.#INF00 | 304 | 1.#QNAN0 | 75 | 15596531712 |
| 46272.16 | -1.#INF00 | 1.#INF00 | 276 | 1.#QNAN0 | 75 | 2.07642E+12 |
| 47380.48 | 1.#QNAN0 | 1.#QNAN0 | 80 | 1.#QNAN0 | 75 | 38445314048 |
| 48916.06 | 1.#QNAN0 | 1.#QNAN0 | 585 | 1.#QNAN0 | 75 | 1927557760 |
| 49957.26 | 1.#QNAN0 | 1.#QNAN0 | 8 | 1.#QNAN0 | 75 | 21354545152 |
| 51279.8 | -1.#INF00 | 1.#INF00 | 366 | 1.#QNAN0 | 75 | 25197451264 |
| 52552.92 | -1.#INF00 | 1.#INF00 | 303 | 1.#QNAN0 | 75 | 2.13941E+12 |
| 53810.78 | -1.#INF00 | 1.#INF00 | 283 | 1.#QNAN0 | 75 | 4845914112 |
| 55151.99 | -1.#INF00 | 1.#INF00 | 392 | 1.#QNAN0 | 75 | 29447045120 |
| 56197.21 | 1.#QNAN0 | 1.#QNAN0 | 7 | 1.#QNAN0 | 75 | 1.2458E+18 |
| 57242.67 | 1.#QNAN0 | 1.#QNAN0 | 8 | 1.#QNAN0 | 75 | 1.43464E+11 |
| 58828.03 | 1.#QNAN0 | 1.#QNAN0 | 647 | 1.#QNAN0 | 75 | 12911873024 |
| 60451.8 | 1.#QNAN0 | 1.#QNAN0 | 684 | 1.#QNAN0 | 75 | 205733248 |
| 62078.71 | 1.#QNAN0 | 1.#QNAN0 | 693 | 1.#QNAN0 | 75 | 26019204 |
| 63123.75 | 1.#QNAN0 | 1.#QNAN0 | 8 | 1.#QNAN0 | 75 | 33034330112 |

It is important to note that the raw sensor measurements seen here are not directly used in the as the measurement vector. In the control program a real time operating system was used and the control function was called every 0.785 seconds. This time period was too large for reliable IMU to be taken so the IMU was sampled at a higher rate and then the average of all the readings were used as the acceleration and the angular velocity readings. The IMU was sampled approximately 112 times for every cycle of the control function. It should also be noted that the second accelerometer was ignored because of bad readings. The measurement vector then simply read the first accelerometer twice. Also, the raw UTM coordinates were very large so the coordinate system was adjusted to the starting point of the control path. Reasons for the weak convergence of the filter are discussed in the conclusions chapter.

# CHAPTER 8. CONCLUSIONS AND RECOMMENDATIONS

This chapter outlines the conclusions based on the results in the previous chapter and make recommendations for future studies.

## 8.1. GPS Algorithm

The results for the GPS algorithm show that certain angles used to calculate the satellite position change over time and overtime can change which quadrant the angle lies in. When switching past multiples of $\frac{\pi}{2}$ either the cosine or sine function will change in positivity which can change the validity of the equations in used. This behavior is highlighted by satellite 4 in the GPS results which starts with a good satellite position. When the corrected argument of latitude, $u_k$, switches from less than to greater than $\frac{\pi}{2}$ between time steps 2 and 3. While it is conceivable to wait for four satellite positions to be calculated correctly this research aims to study the accuracy of the position fix over a long period of time and this would not be practical.

Because the changing angles can change the validity of the equations it was concluded that the information sent from the Copernicus II receiver either needs to be bounded within a range or a correct reference position needs to be used. This was seen for sure in the corrected argument of latitude but it is likely true for other angles used in calculation as well. The information in the Copernicus II manuals did not indicate any special references in the information the receivers calculated but they may still be there. Attempts to gather information from Copernicus II proved difficult and time was not sufficient to find the correct ranges and references to always find an accurate satellite position so this portion of the research was tabled for now. In future research it is suggest a receiver with better documentation on gathered information be used.

## 8.2. Dynamic Model Simulations

The model simulations performed very well and showed that the dynamic model was sufficiently able to capture the dynamics of the vehicle. Through the path tracking algorithm the robot was able to maintain the course within 3/4th of the overall length which means part of the vehicle should have intersected the course at all times. There was however a problem with the numerics of the control algorithm where the robot lost its way while completing the sinusoidal path. This is likely due to the change in the direction of the robot and does not show a problem with the dynamic model which was created.

## 8.3. Control and Filter Experimentation

In controlling the robot experimentally the filter was unable to converge strongly and this created errors in the state which the robot was not able to overcome during experimentation. This could be due to the system matrix terms, which are dependent on the heading angle, turning to zero during the system update which could cause the system to become unobservable and possibly uncontrollable. Specifically these terms relate to the $X$ and $Y$ coordinates of the robot. In practice the uncontrollability would be overcome because the robot would be running initially and it wouldn't be able control the $X$ and $Y$ coordinates and it would still be controlling the velocities. The robot would then move past the point where it is uncontrollable and keep on going. The observability, however, makes it so the state estimation has an error and this could cause the system to crash. There are other factors as well such as not having accurate knowledge of the robot's physical parameters and not having accurate representations of the noise characteristics of the sensors. It is recommended that in future studies the system matrix is monitored and if it becomes unobservable a small value is used instead of zero to maintain observability. It is also recommended that the robot and sensor parameters are accurately measured.

## 8.4. Concluding Remarks

In conclusion this study had four main goals. The first goal of creating a model well suited for high friction surfaces was completed and verified through simulation results. The second goal was to characterize and reduce errors in low cost GPS and IMU units. This was partially completed by being able to characterize the errors inherent in low cost GPS units. The work for reducing the error has yet to be completed because the satellite positions were only sometimes able to be found. This goal was also partially completed by the formulation of the robust Kalman filter. The filter had issues with observability but initial formulations were tested and recommendations for improvements were made. The third goal was also partially completed because the control algorithm was formulated and was able to converge if the state was known, however it was not able to be tested full because the state vector was not able to be found through the Kalman filter. The final goal of experimentation was started and it was found that the initial formulation of the Kalman filter was preventing the experiments from succeeding. In future studies the model and control algorithm should work if the model is made to work in the Kalman filter.

## REFERENCES

[1] V. Godon and N. Godon, “Fargo , North Dakota Climate.” National Weather Service Eastern North Dakota, Grand Forks, North Dakota, 2002.

[2] G. Rutherford, N. Marcy, and A. Mills, “The Hazard Screening Report: Yard and Garden Report,” Consumer Product Safety Commission, 2003.

[3] A. Mandow and J. Martinez, “Experimental kinematics for wheeled skid-steer mobile robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1222–1227.

[4] J. Yi, H. Wang, and J. Zhang, “Kinematic modeling and analysis of skid-steered mobile robots with applications to low-cost inertial-measurement-unit-based motion estimation,” *IEEE Trans. Robot.*, vol. 25, no. 5, pp. 1087–1097, 2009.

[5] J. Pentzer, S. Brennan, and K. Reichard, “On-Line Estimation of Vehicle Motion and Power Model Parameters for Skid-Steer Robot Energy Use Prediction,” *Proc. 2014 Am. Control Conf.*, pp. 2786–2791, 2014.

[6] V. Nazari and M. Naraghi, “Sliding Mode Fuzzy Control of a Skid Steer Mobile Robot for Path Following,” in *IEEE/RSJ International Conference on Control, Automation, Robotics, and Vision*, 2008, no. December, pp. 549–554.

[7] R. E. Colyer and J. T. Economou, “Soft modelling and fuzzy logic control of wheeled skid-steer electric vehicles with steering prioritisation,” *Int. J. Approx. Reason.*, vol. 22, no. 1–2, pp. 31–52, 1999.

[8] Z. Yu, H. Jibin, L. Xueyuan, L. Shupeng, and G. Jing, “A linear lateral dynamic model of skid steered wheeled vehicle,” *2013 IEEE Intell. Veh. Symp.*, no. 4, pp. 964–969, Jun. 2013.

[9] G. Shuang and N. Cheung, “Skid Steering in 4-Wheel-Drive Electric Vehicle,” *IEEE Power Electron. Drive Syst.*, pp. 1548–1553, 2007.

[10] Z. Jian, W. Shuang-shuang, L. Hua, and L. Bin, “The sliding mode control based on extended state observer for skid steering of 4-wheel-drive electric vehicle,” *2012 2nd Int. Conf. Consum. Electron. Commun. Networks*, vol. 2, pp. 2195–2200, Apr. 2012.

[11] W. Yu and O. Chuy, “Analysis and experimental verification for dynamic modeling of a skid-steered wheeled vehicle,” *IEEE Trans. Robot.*, vol. 26, no. 2, pp. 340–353, 2010.

[12] W. Yu and O. Chuy, “Dynamic modeling of a skid-steered wheeled vehicle with experimental verification,” in *IEEE/RJS International Conference on Intelligent Robots and Systems*, 2009, pp. 4212–4219.

[13] A. R. Everett, “Sensors for Mobile Robots : Theory,” *IEEE Trans. Robot. Autom.*, vol. 12, pp. 922–923, 1996.

[14] SICK AG, “Technical Documentation LMS200/211/221/291 Laser Measurement Systems,” 2006.

[15] Hagisonic Co. Ltd, “Hagisonic StarGazer Robot Localization System,” 2014.

[16] S. Pace, G. Frost, I. Lachow, D. Frelinger, and D. Fossum, *The global positioning system: assessing national policies*. Santa Monica, CA, 1995.

[17] T. H. Witte and a M. Wilson, “Accuracy of non-differential GPS for the determination of speed over ground.,” *J. Biomech.*, vol. 37, no. 12, pp. 1891–8, Dec. 2004.

[18] J. S. Subirana, J. M. J. Zornoza, and M. Hernández-Pajares, *GNSS Data Processing Volume I: Fundamentals and Algorithms*, vol. I. Noordwijk, Netherlands: ESA Communications, 2013.

[19] C. Pinana-Diaz, “GPS multipath detection and exclusion with elevation-enhanced maps,” in *International IEEE Conference on Intelligent Transportation Systems*, 2011, pp. 19–24.

[20] J. Meguro, T. Murata, J. Takiguchi, Y. Amano, and T. Hashizume, “GPS Multipath Mitigation for Urban Area Using Omnidirectional Infrared Camera,” *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 1, pp. 22–30, Mar. 2009.

[21] T. Kos, I. Markezic, and J. Pokrajcic, “Effects of multipath reception on GPS positioning performance,” in *International Symposium ELMAR*, 2010, pp. 399–402.

[22] E. Wang, “Research on improving accuracy of GPS positioning based on particle filter,” in *IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, 2013, vol. 1, pp. 1167–1171.

[23] S. Yamaguchi and T. Tanaka, “GPS standard positioning using Kalman filter,” in *SICE-ICASE, 2006. International Joint Conference*, 2006, no. 3, pp. 1351–1354.

[24] A. Narvesen and M. Selekwa, “Reduction of GPS Noise for Precision Control of Robot Navigation in Confined Areas,” in *International Mechanical Engineering Conference*, 2014.

[25] M. Bruch, G. Gilbreath, J. Muelhauser, and J. Lum, “Accurate waypoint navigation using nondifferential GPS.” Space and Naval Warfare Sytems Center, San Diego, California, 2002.

[26] J. Borenstein and L. Feng, “Gyrodometry: a new method for combining data from gyros and odometry in mobile robots,” *Int. Conf. Robot. Autom.*, vol. 1, no. April, pp. 423–428, 1996.

[27] R. Bogue, “MEMS sensors: past, present and future,” *Sens. Rev.*, vol. 27, pp. 7–13, 2007.

[28] C. Acar and A. M. Shkel, “Experimental evaluation and comparative analysis of commercial variable-capacitance MEMS accelerometers,” *J. Micromechanics Microengineering*, vol. 13, pp. 634–645, 2003.

[29] W. Ang and S. Khoo, “Physical model of a MEMS accelerometer for low-g motion tracking applications,” in *International Conference on Robotics and Automation*, 2004, no. April, pp. 1345–1351.

[30] H. Hyvönen, “Thermomechanical and Mechanical Characterization of a 3-Axial MEMS Gyroscope,” M.S. Thesis, Dept. Elect. Eng., Aalto University, Helsinky, 2011.

[31] L. Dachuan, C. Xiaozhu, C. Jian, L. Longtao, Z. Qiancheng, Y. Zhenchuan, and Y. Guizhen, “Research on temperature dependent characteristics and compensation methods for digital gyroscope,” *Proc. 3rd Int. Conf. Sens. Technol. ICST 2008*, no. 4, pp. 273–277, 2008.

[32] M. Graham and G. J. Martin, “Gyroscopes may cease spinning,” *IEEE Spectr.*, vol. 23, pp. 48–53, 1986.

[33] R. B. Northrop, *Introduction to Instrumentation and Measurements*, Second Edi. Boca Raton, FL: Taylor & Francis, 2005.

[34] V. Vali and R. W. Shorthill, “Fiber ring interferometer.,” *Appl. Opt.*, vol. 15, no. 5, pp. 1099–1100, 1976.

[35] B. Shamah, “Steering and control of a passively articulated robot,” *Proc. SPIE*, vol. 4571, pp. 96–107, 2001.

[36] R. Wang, H. Zhang, J. Wang, F. Yan, and N. Chen, “Robust lateral motion control of four-wheel independently actuated electric vehicles with tire force saturation consideration,” *J. Franklin Inst.*, vol. 352, no. 2, pp. 645–668, 2015.

[37] M. F. Selekwa and J. R. Nistler, “Path tracking control of four wheel independently steered ground robotic vehicles,” *IEEE Conf. Decis. Control Eur. Control Conf.*, pp. 6355–6360, Dec. 2011.

[38] M. Woods and J. Katupitiya, “Modelling of a 4WS4WD vehicle and its control for path tracking,” *IEEE Symp. Comput. Intell. Control Autom.*, pp. 155–162, 2013.

[39] R. Oftadeh, “A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels,” in *IEEE/RJS International Conference on Intelligent Robots and Systems*, 2013, pp. 4845–4851.

[40] S. Sakai and Y. Hori, “Advantage of electric motor for anti-skid control of electric vehicle,” *Eur. Power Electron. J.*, vol. 11, no. 4, pp. 26–32, 2001.

[41] M. S. Saidonr, H. Desa, and M. N. Rudzuan, “A differential steering control with proportional controller for an autonomous mobile robot,” *Proc. - 2011 IEEE 7th Int. Colloq. Signal Process. Its Appl. CSPA 2011*, pp. 90–94, 2011.

[42] Y. Z. Y. Zhang, D. H. D. Hong, J. H. Chung, and S. a. Velinsky, “Dynamic model based robust tracking control of a differentially steered wheeled mobile robot,” in *Proceedings of the American Control Conference*, 1998, pp. 850–855.

[43] J. Wong and C. Chiang, “A general theory for skid steering of tracked vehicles on firm ground,” *Proc. Inst. Mech. Eng.*, vol. 215, no. D, pp. 343–355, 2001.

[44] S. Jayasuriya, “Modeling and motion stability analysis of skid-steered mobile robots,” *2009 IEEE Int. Conf. Robot. Autom.*, pp. 4112–4117, May 2009.

[45] J. Aslam, S.-Y. Qin, and M. A. Alvi, “Fuzzy sliding mode control algorithm for a four-wheel skid steer vehicle,” *J. Mech. Sci. Technol.*, vol. 28, no. 8, pp. 3301–3310, 2014.

[46] E. Mohammadpour and M. Naraghi, “Robust Adaptive Stabilization of Skid Steer Wheeled Mobile Robots Considering Slipping Effects,” *Adv. Robot.*, vol. 25, no. 1–2, pp. 205–227, 2011.

[47] E. Mohammadpour, M. Naraghi, and M. Gudarzi, “Posture stabilization of skid steer wheeled mobile robots,” *2010 IEEE Conf. Robot. Autom. Mechatronics, RAM 2010*, pp. 163–169, 2010.

[48] A. Narvesen and M. Selekwa, “Dynamics and Control of four wheeled differentiall steered UGVs,” in *International Mechanical Engineering Conference*, 2014, pp. 1–6.

[49] Motenergy Inc., “Motenergy ME0708 Performance Curves,” 2008.

[50] J. R. Movellan, “DC Motors,” 2010. [Online]. Available: http://mplab.ucsd.edu/tutorials/dc.pdf.

[51] H. B. Pacejka and E. Bakker, “The magic formula tyre model,” *Veh. Syst. Dyn.*, vol. 21, no. S1, pp. 1–18, 1992.

[52] ARINC Research Corporation, “Navstar GPS Space Segment / Navigation User Interfaces,” El Segundo, CA, Tech. Rep. IS-GPS-200 Rev C, 2000.

[53] GPS NAVSTAR, “Global Positioning System Standard Positioning Service Signal Specification,” 1995.

[54] Trimble Navigation Ltd., “Copernicus ® II GPS Receiver Reference Manual,” 2009.

[55] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 4th ed. Hoboken, New Jersey: Wiley, 2013.

[56] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes: The Art of Scientific Computing*. New York, New York: Cabridge University Press, 2002.

[57] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. New York, New York: Springer, 2008.

[58] D. Kincaid and W. Cheney, *Numerical Analysis*, vol. 506. Belmont, California: Brooks/Cole Publishing Company, 1991.

[59] B. R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," no. 1, 1960.

[60] W. C. Davidon, "Variance algorithm for minimization," *Comput. J.*, vol. 10, no. 4, pp. 406–410, 1968.

[61] C. G. Broyden, "The Convergence of a Class of Double-rank Minization Algorithms," *J. Math. its Appl.*, vol. 6, pp. 76–90, 1970.

[62] R. Fletcher, "A new approach to variable metric algorithms," *Comput. J.*, vol. 13, no. 3, pp. 317–322, 1970.

[63] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Math. Comput.*, vol. 24, no. 109, pp. 23–23, 1970.

[64] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. Comput.*, vol. 24, no. 111, pp. 647–647, 1970.

[65] B. Siciliano and O. Khatib, *Handbook of Robotics*, no. February 2007. Würzburg, Germany: Springer, 2008.

[66] D. T. Greenwood, *Advanced Dynamics*. New York, New York: Cambridge University Press, 2003.

[67] A. A. Shabana, *Dynamics of Multibody Systems*, Third. New York, New York: Cambridge University Press, 2005.

[68] A. M. Lyapunov, *General Problem of the Stability Of Motion*. New York, New York: Taylor & Francis, 1992.

[69] R. E. R. Kalman, "Contributions to the theory of optimal control," *Bol. Soc. Mat. Mex.*, vol. 5, no. 2, pp. 102–119, 1960.

[70] R. E. Kalman, "When Is a Linear Control System Optimal?," *J. Fluids Eng.*, vol. 86, no. 1, pp. 51–60, Mar. 1964.

[71] W. Levine and M. Athans, "On the determination of the optimal constant output feedback gains for linear multivariable systems," *IEEE Trans. Automat. Contr.*, vol. 15, no. 1, pp. 44–48, 1970.

[72] W. Levine, T. Johnson, and M. Athans, "Optimal limited state variable feedback controllers for linear systems," *IEEE Trans. Automat. Contr.*, vol. 16, no. 6, 1971.

[73] A. E. Bryson, *Applied Optimal Control: Optimization, Estimation and Control*. New York, New York: Taylor & Francis, 1975.

[74] K. Zhou, J. C. Doyle, and K. Glover, "Robust and Optimal Control," 2008.

[75] T. Glad and L. Ljung, *Control Theory*, 1st ed. New York, New York: Taylor & Francis, 2000.

[76] F. E. White, "Data Fusion Lexicon." Joint Directors of Laboratories, Washingto, DC, USA, Tech. Rep. 20100621258, 1991.

[77] T. Henderson, M. Dekhil, R. Kessler, and M. Griss, "Sensor fusion," *Control Problems in Robotics and Automation*, vol. 230. pp. 193–207, 1998.

[78] F. Castanedo, "A review of data fusion techniques.," *Sci. World J.*, vol. 2013, pp. 1–19, 2013.

[79] A. P. Dempster, "New Methods for Reasoning Towards Posterior Distribution Based on Sample Data," *Ann. Math. Stat.*, vol. 37, no. 2, pp. 355–374, 1966.

[80] A. P. Dempster, "Upper and Lower Probabilitites Induced by a Multivalued Mapping," *Ann. Math. Stat.*, vol. 38, no. 2, pp. 325–339, 1967.

[81] H. F. Durrant-Whyte, "Sensor Models and Multisensor Integration," *Int. J. Rob. Res.*, vol. 7, pp. 97–113, 1988.

[82] J. F. Nash, "The Bargaining Problem," *Econometrica*, vol. 28, no. 2, pp. 155–162, 1950.

[83] K.-C. Chu, "Team decision theory and information structures in optimal control problems--Part II," *IEEE Trans. Automat. Contr.*, vol. 17, no. 1, 1972.

[84] Y.-C. Ho and K.-C. C. Chu, "Team decision theory and information structures in optimal control problems--Part I," *IEEE Trans. Automat. Contr.*, vol. 17, no. 1, pp. 15–22, 1972.

[85] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82. pp. 35–45, 1960.

[86] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

[87] S. J. Julier and J. K. Uhlmann, “A New Extension of the Kalman Filter to Nonlinear Systems,” *AeroSense’97*, vol. 3, pp. 182–193, 1997.

[88] N. J. Gordon, D. J. Salmond, and a F. M. Smith, “Novel-Approach to Nonlinear Non-Gaussian Bayesian State Estimation,” *IEEE Proceedings-F Radar Signal Process.*, vol. 140, pp. 107–113, 1993.

[89] Wm. M. Wonham, “On the separation theorem of stochastic control,” *SIAM J. Control*, vol. 6, no. 2, pp. 312–326, 1968.

[90] X. Qing, H. R. Karimi, Y. Niu, and X. Wang, “Decentralized unscented Kalman filter based on a consensus algorithm for multi-area dynamic state estimation in power systems,” *Int. J. Electr. Power Energy Syst.*, vol. 65, pp. 26–33, 2015.

[91] N. Assimakis, M. Adam, M. Koziri, S. Voliotis, and K. Asimakis, “Optimal decentralized Kalman filter and Lainiotis filter,” *Digit. Signal Process. A Rev. J.*, vol. 23, no. 1, pp. 442–452, 2013.

[92] A. Ahmad, M. Gani, and F. Yang, “Decentralized robust Kalman filtering for uncertain stochastic systems over heterogeneous sensor networks,” *Signal Processing*, vol. 88, no. 8, pp. 1919–1928, 2008.

[93] L. Xie, Y. C. Soh, and C. E. de Souza, “Robust Kalman filtering for uncertain discrete-time systems,” *IEEE Trans. Automat. Contr.*, vol. 39, no. 6, pp. 1310–1314, 1994.

[94] L. El Ghaoui and G. Calafiore, “Robust Filtering for Discrtete-Time Systems with Bounded Noise and Parametric Uncertainty,” vol. 46, no. 7, pp. 1084–1089, 2001.

[95] R. F. Souto and J. Y. Ishihara, “Enhanced robust Kalman predictor for discrete-time systems with uncertain correlated noises,” *Proc. IEEE Conf. Decis. Control*, pp. 1660–1665, 2008.

[96] X. Zhu, Y. C. Soh, and L. Xie, “Robust Kalman filter design,” *Proc. 39th IEEE Conf. Decis. Control (Cat. No.00CH37187)*, vol. 4, pp. 3–8, 2000.

[97] C. E. de Souza, M. Fu, and L. Xie, “Hinf Analysis and Synthesis of Discrete-Time Systems with Time-Varying Uncertainty,” *IEEE Trans. Autom. Contr.*, vol. 38, no. 3, pp. 1058–1061, 1993.

[98] P. Vaidyanathan, “The discrete-time bounded-real lemma in digital filtering,” *IEEE Trans. Circuits Syst.*, vol. 32, no. 9, pp. 918–924, 1985.

[99] H. K. Wimmer, “Extensions of the bounded real lemma of discrete-time systems,” *Int. J. Control*, vol. 73, no. 14, pp. 1322–1328, 2000.

[100] D. S. Kelso, “TLERetriever3.” Center for Space Standards and Innovation, Colorado Springs, Colorado, 2015.

[101] V. Okan, “GPS 2.4.” Hochschule Darmstadt University of Applied Sciences, Dieburg, Germany, 1996.

[102] M. Canton, C. Feldner, P. Nelson, M. Stousland, M. Selekwa, and X. Wang, “Autonomous Snowplow 2013,” 2013.

# APPENDIX A. IMPROVED GPS TRILATERATION CODE

```
#include <math.h>
#include <unistd.h>
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_pwr.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_usart.h"
typedef struct ret_t
{
    uint8_t Data[500];
    uint8_t Check;
} ret_t;

typedef struct inv
{
    double binv[4][4];
} inv;

typedef struct orbit_t
{
      double M[12];
      double Ek[12];
      double Thk[12];
      double ik[12];
      double uk[12];
} orbit_t;

orbit_t orbital;
typedef struct sat_dat_t
{


      //GPS Fix Data
      float X;
      float Y;
      float Z;
      float X_ECEF;
      float Y_ECEF;
      float Z_ECEF;
      float Time;
      //Sat strength data
      uint8_t NoSat;
      uint8_t Sat[12];
      float Level[12];
      //Sat health data
      uint8_t nSVs;
      uint8_t PRN[12];
      //Ephemeris Data
      float t_ephem[12];
      int IODC[12];
      float T_GD[12];
      float t_oc[12];
      float a_f2[12];
      float a_f1[12];
```

```
	float a_f0[12];
	uint8_t IODE[12];
	float C_rs[12];
	float delta_n[12];
	float C_uc[12];
	float C_us[12];
	float t_oe[12];
	float C_ic[12];
	float C_is[12];
	float C_rc[12];
	float OMEGADOT[12];
	float IDOT[12];

	double e[12];
	double sqrt_A[12];
	double M_0[12];
	double OMEGA_0[12];
	double i_0[12];
	double omega[12];
	double nn[12];
	double r1me2[12];


	//Raw Data
	double TOM[12];
	uint8_t TDiff[12];
	float codephase[12];
	float SigLev[12];
	float Doppler[12];
	float delta_tr[12];
	float TOW;

	//GC1
	uint16_t GC1weekn;
	uint GC1TOWms;
	int GC1Fractional;
	int GC1Altitude;
	uint8_t GC1SatID[12];
	uint8_t GC1Sig[12];
	uint16_t GC1Aqu[12];
	uint GC1Psuedo[12];
	int GC1RangeRate[12];
} sat_dat_t;

sat_dat_t Sat_Dat;
void myUSART_Init(void);
uint8_t myUSART_GetByte(void);
ret_t myUSART_TrapByte(uint8_t Trap1,uint8_t Trap2);//If normal packet
Trap2==0;
void myUSART_SendByte(uint8_t data );
void Configure_CopernicusGPS(void);
void THUNDAR_GPS(void);
void Satellite_Fix_A(void);
inv invmatrix(double a[4][4]);
void Position_Fix_A(void);
void Position_Fix_B(void);
void Position_Fix_C(void);
```

```
//Calculated Variables
double X_PA[4];
double Y_PA[4];
double Z_PA[4];
double T_PA[4];
double RR[4];
double PR[4];
double tk[4];
double R[4];
double X_r_BFGS;
double Y_r_BFGS;
double Z_r_BFGS;
double X_r_LS;
double Y_r_LS;
double Z_r_LS;
double X_r_NR;
double Y_r_NR;
double Z_r_NR;

//Temporary data
uint8_t ct;
float test[40];
double testd[15];
float qq[10]={1,1,1,1,1,1,1,1,1,1};



//=========================================================================
void myUSART_Init(void)
{    GPIO_InitTypeDef  GPIO_InitStructure;
       USART_InitTypeDef USART_InitStructure;

       RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOD, &GPIO_InitStructure);

        // Connect UART pins to GPIO
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_USART3);   // TX
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_USART3);   // RX



        // Initialize USART
        USART_InitStructure.USART_BaudRate = 38400;  //Default baud rate
        USART_InitStructure.USART_WordLength = USART_WordLength_8b;  // 8
bits
        USART_InitStructure.USART_StopBits = USART_StopBits_1; // 1 stop bit
        USART_InitStructure.USART_Parity = USART_Parity_No; //no parity
        USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; //No hardware flow control
        USART_InitStructure.USART_Mode =  USART_Mode_Rx | USART_Mode_Tx;
        /* Configure USART */
        USART_Init(USART3, &USART_InitStructure);
        /* Enable the USART */
        USART_Cmd(USART3, ENABLE);
}


//============================================================================
uint8_t myUSART_GetByte(void)
{ uint8_t x;
  while (USART_GetFlagStatus(USART3, USART_FLAG_RXNE) == RESET)
  {
            x++;
            if(x>500)
            {break;}
  }
  return (uint8_t) USART_ReceiveData(USART3);
}
//============================================================================
ret_t myUSART_TrapByte(uint8_t Trap1,uint8_t Trap2)
{
uint8_t i;
uint8_t k;
uint8_t n;
uint8_t t;
uint8_t TrapByte;
uint8_t nmax;
i=0;
ct=0;
t=0;
ret_t DataArray={0};
//Initial
Bits================================================================
LOOPX:
      if(ct>200){DataArray.Check=1; goto END;}
      else {DataArray.Check=0;}
      ct++;
    n=0;
    TrapByte=myUSART_GetByte();
    if(TrapByte!=0x10) goto LOOPX;
    DataArray.Data[n]=TrapByte;
    n++;
    TrapByte=myUSART_GetByte();
    if(TrapByte!=Trap1) goto LOOPX; //Check for intended packet
    DataArray.Data[n]=TrapByte;
    //n++;
    if(Trap2!=0)                   //Check if normal or super packet
    {
        //Super Packet Data Header===
        TrapByte=myUSART_GetByte();
        if(TrapByte!=Trap2) goto LOOPX;
        n++;
        DataArray.Data[n]=TrapByte;
    }
```

```
LOOPX2:
    n++;
    DataArray.Data[n]=myUSART_GetByte();
    if (DataArray.Data[n]==0x10)
        {
            i++;                    //Count intermediate 0x10 bytes
        }
    if ((DataArray.Data[n]==0x03) && (DataArray.Data[n-1]==0x10) &&
((i&1)==1)) //Check for legal end of data transmission
        {
            goto LOOPY;
        }
        else
        {
            if (n>=179)
            {DataArray.Check=1; goto END;}
            else
            {goto LOOPX2;}
        }
    //Check for Byte Stuffing
    LOOPY:
    nmax=n;
    n=0;
    for (n=1;n<nmax;n++)
    {
        if ((DataArray.Data[n-
1]==DataArray.Data[n])&&(DataArray.Data[n]==0x10)&&(i>1))
        {
            nmax--;
            i=i-2;//subtract 2 on the counter for both 10s
            for(k=n;k<nmax;k++) DataArray.Data[k-1]=DataArray.Data[k];
        }
    }
    if (i!=1)
    {
      DataArray.Check=1;
    }
    END:
    return(DataArray);
}

//=============================================================================
void myUSART_SendByte(uint8_t data )
{      uint8_t x;
x=0;
      while (USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET)
      {
            x++;
            if(x>500)
            {break;}
      }
      USART_SendData(USART3, data);
}

//=============================================================================

void Configure_CopernicusGPS(void)
```

```
{
      uint8_t m;
      uint8_t n;
      uint8_t
NavigConfig[]={0x10,0xBB,0x00,0x00,0x01,0x02,0x00,0x3E,0x33,0x33,0x33,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0
x03}; //Navigation dynamics
      uint8_t
TurnOFFPortB[]={0x10,0xBC,0x01,0x06,0x06,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0
x10,0x03}; //Port B
      uint8_t SetIOOptions[]={0x10,0x35,0x25,0x02,0x41,0x01,0x10,0x03}; //IO
Options OK
      uint8_t SetDatum[]={0x10,0x8E,0x15,0x00,0x00,0x10,0x03}; //Datum WGS-84
      uint8_t NoAuto[]={0x10,0x8E,0x20,0x00,0x10,0x03}; //No Auto outputing
ECEF and LLA
      uint8_t NoUTM1[]= {0x10,0x8E,0x17,0x00,0x10,0x03}; //No Auto outputing
UTM Single
      uint8_t NoUTM2[]= {0x10,0x8E,0x18,0x00,0x10,0x03}; //No Auto outputing
UTM Double
      uint8_t NoAuto2[]={0x10,0x8E,0x21,0x00,0x10,0x03}; //No Auto Extra Info
      uint8_t NoAuto3[]={0x10,0x8E,0x23,0x00,0x10,0x03}; //No Auto Extra Info
      uint8_t NoAuto4[]={0x10,0x8E,0x2A,0x00,0x10,0x03}; //No Auto Extra Info
      uint8_t NoAuto5[]={0x10,0x8E,0x2B,0x00,0x10,0x03}; //No Auto Extra Info
      myUSART_Init();          //Set GPS to USART Channel 2, 3,and 4

            m=sizeof(NavigConfig)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NavigConfig[n]);
            //Configure navigation to Sea Dynamics at 10 degrees level mask
            }


            m=sizeof(TurnOFFPortB)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
              myUSART_SendByte(TurnOFFPortB[n]);//Turn Off the unused Port B
            }

            m=sizeof(SetIOOptions)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(SetIOOptions[n]);
            //Configure I/O Options XYZ-ECEF, pps OFF, Superpackets ON
            }


            m=sizeof(SetDatum)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(SetDatum[n]);
            //Set Datum to NAR-C (i.e., NAD 83 CONUS)
            }


            m=sizeof(NoAuto)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoAuto[n]);
            //No automatic ECEF and LLA outputting
            }


            m=sizeof(NoUTM1)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoUTM1[n]);
            //No automatic UTM-Single outputting
            }


            m=sizeof(NoUTM2)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoUTM2[n]);
            //No automatic UTM-Double outputting
            }


            m=sizeof(NoAuto2)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoAuto2[n]);
            //No automatic UTM-Double outputting
            }


            m=sizeof(NoAuto3)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoAuto3[n]);
            //No automatic UTM-Double outputting
            }


            m=sizeof(NoAuto4)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoAuto4[n]);
            //No automatic UTM-Double outputting
            }


            m=sizeof(NoAuto5)/sizeof(uint8_t);
            for(n=0;n<m;n++)
            {
                  myUSART_SendByte(NoAuto5[n]);
            //No automatic UTM-Double outputting
            }

}
```

```
//===========================================================================
void THUNDAR_GPS(void)
      {
//Declare Variables
      uint32_t i1;
      uint32_t i2;
          union
            {
                    uint64_t HexValue;
                    float FloatValue;
             }TempData;
          union
              {
               uint8_t byte[8];
                     double DoubleValue;
            }TempDouble;
          ret_t   DataArray;
            uint8_t res;
          uint32_t i;
            uint8_t j;
            uint8_t k;
            uint8_t m;
            uint8_t n;
            uint8_t GetSatDat[]={0x10,0x27,0x10,0x03}; //Satellites used
            uint8_t GetTOW[]={0x10,0x21,0x10,0x03}; //Satellites used
            uint8_t GetUsedSat[]={0x10,0x24,0x10,0x03};//Data on Satellites
            uint8_t GetGPSData[]={0x10,0x8E,0x17,0x10,0x03};//Get Coordinates
            uint8_t GetIon[]={0x10,0x38,0x01,0x06,0x1A,0x10,0x03};//Get Ion
            uint8_t
GetIon10[]={0x10,0x38,0x01,0x06,0x10,0x10,0x10,0x03};//Get Ephemiris 0x10
            uint8_t GetRaw[]={0x10,0x3A,0x00,0x10,0x03};//Get Raw Data
            uint8_t GetRaw10[]={0x10,0x3A,0x10,0x10,0x10,0x03};//Get Raw Data
            uint8_t GetChannel1[]={0x8E,0x2A,0x10,0x03};
            uint8_t GetAl[]={0x10,0x38,0x01,0x02,0x1A,0x10,0x03};
            //Get Ephemiris
            uint8_t GetAl10[]={0x10,0x38,0x01,0x02,0x10,0x10,0x10,0x03};
            //Get Ephemiris 0x10
      //=====================================================================
            //Read Data
            ct=0;
            res=0;
            res=0;
            Retry0:
            res++;
            for(i=0;i<180;i++)
            {
                  DataArray.Data[i]=0;
            }
            DataArray=myUSART_TrapByte(0x42,0);
            if(DataArray.Check==1 && res<50){goto Retry0;}
            ct=0;
            res=0;

      TempData.HexValue=(DataArray.Data[2]<<24)+(DataArray.Data[3]<<16)+(Data
Array.Data[4]<<8)+(DataArray.Data[5]); //GPSData[8] was GPSData[9]
            Sat_Dat.X_ECEF=TempData.FloatValue;
```

```
	TempData.HexValue=(DataArray.Data[6]<<24)+(DataArray.Data[7]<<16)+(Data
Array.Data[8]<<8)+(DataArray.Data[9]);
		Sat_Dat.Y_ECEF=TempData.FloatValue;

	TempData.HexValue=(DataArray.Data[10]<<24)+(DataArray.Data[11]<<16)+(Da
taArray.Data[12]<<8)+(DataArray.Data[13]);
		Sat_Dat.Z_ECEF=TempData.FloatValue;


		ct=0;
		res=0;
		res=0;
		Retry1:
		res++;
		for(i=0;i<180;i++)
		{
			DataArray.Data[i]=0;
		}
		m=sizeof(GetGPSData)/sizeof(uint8_t);
			for(n=0;n<=m;n++)
			{
				myUSART_SendByte(GetGPSData[n]);
			}
		DataArray=myUSART_TrapByte(0x8F,0x17);
		if(DataArray.Check==1 && res<50){goto Retry1;}
		ct=0;
		res=0;

	TempData.HexValue=(DataArray.Data[6]<<24)+(DataArray.Data[7]<<16)+(Data
Array.Data[8]<<8)+(DataArray.Data[9]); //GPSData[8] was GPSData[9]
		Sat_Dat.Y=TempData.FloatValue;

	TempData.HexValue=(DataArray.Data[10]<<24)+(DataArray.Data[11]<<16)+(Da
taArray.Data[12]<<8)+(DataArray.Data[13]);
		Sat_Dat.X=TempData.FloatValue;

	TempData.HexValue=(DataArray.Data[14]<<24)+(DataArray.Data[15]<<16)+(Da
taArray.Data[16]<<8)+(DataArray.Data[17]);
		Sat_Dat.Z=TempData.FloatValue;

	TempData.HexValue=(DataArray.Data[22]<<24)+(DataArray.Data[23]<<16)+(Da
taArray.Data[24]<<8)+(DataArray.Data[25]);
		Sat_Dat.Time=TempData.FloatValue;

		//Get Satellite Data
		Retry2:
		res++;
		for(i=0;i<50;i++)
		{
			DataArray.Data[i]=0;
		}
		m=sizeof(GetSatDat)/sizeof(uint8_t);
		for(n=0;n<=m;n++)
		{
			myUSART_SendByte(GetSatDat[n]);
		}
```

```
            DataArray=myUSART_TrapByte(0x47,0);
            if(DataArray.Check==1 && res<50){goto Retry2;}
            Sat_Dat.NoSat=DataArray.Data[2];
            ct=0;
            res=0;
            for(i=0;i<Sat_Dat.NoSat;i++)
            {
                  Sat_Dat.Sat[i]=DataArray.Data[3+i*5];

      TempData.HexValue=(DataArray.Data[4+i*5]<<24)+(DataArray.Data[5+i*5]<<1
6)+(DataArray.Data[6+i*5]<<8)+(DataArray.Data[7+i*5]);
                  Sat_Dat.Level[i]=TempData.FloatValue;
            }
            for(i=Sat_Dat.NoSat;i<12;i++)
            {
                  Sat_Dat.Sat[i]=0;
                  Sat_Dat.Level[i]=0;
            }

            //Get Used Satellites
            Retry3:
            res++;
            for(i=0;i<180;i++)
            {
                  DataArray.Data[i]=0;
            }
            m=sizeof(GetUsedSat)/sizeof(uint8_t);
            for(n=0;n<=m;n++)
            {
                  myUSART_SendByte(GetUsedSat[n]);
            }
            DataArray=myUSART_TrapByte(0x6D,0);
            if(DataArray.Check==1 && res<50){goto Retry3;}
            ct=0;
            res=0;
            Sat_Dat.nSVs=(DataArray.Data[2]&0xF);

            for(i=0;i<12;i++)
            {
                  Sat_Dat.PRN[i]=0;
            }
            for(i=0;i<Sat_Dat.nSVs;i++)
            {
                  Sat_Dat.PRN[i]=DataArray.Data[19+i];
            }

            //Get Ephemeris Data and Raw Data
            k=Sat_Dat.NoSat;
            for(i=0;i<k;i++)
            {
                  Retry4:
                  res++;
                  for(j=0;j<180;j++)
                  {
                        DataArray.Data[j]=0;
                  }
                  if(Sat_Dat.Sat[i]==0x10)
```

```
                        {
                              m=sizeof(GetIon10)/sizeof(uint8_t);
                              for(n=0;n<=m;n++)
                              {
                                    myUSART_SendByte(GetIon10[n]);
                              }
                        }
                        else
                        {
                              GetIon[4]=Sat_Dat.Sat[i];
                              m=sizeof(GetIon)/sizeof(uint8_t);
                              for(n=0;n<=m;n++)
                              {
                                    myUSART_SendByte(GetIon[n]);
                              }
                        }
                        DataArray=myUSART_TrapByte(0x58,0);
                        if(DataArray.Check==1 && res<50){goto Retry4;}

                        res=0;
                        ct=0;

            TempData.HexValue=(DataArray.Data[7]<<24)+(DataArray.Data[8]<<16)+(Data
Array.Data[9]<<8)+(DataArray.Data[10]);
                        Sat_Dat.t_ephem[i]=TempData.FloatValue;

            Sat_Dat.IODC[i]=(DataArray.Data[17]<<8)+(DataArray.Data[18]);

            TempData.HexValue=(DataArray.Data[19]<<24)+(DataArray.Data[20]<<16)+(Da
taArray.Data[21]<<8)+(DataArray.Data[22]);
                        Sat_Dat.T_GD[i]=TempData.FloatValue;

            TempData.HexValue=(DataArray.Data[23]<<24)+(DataArray.Data[24]<<16)+(Da
taArray.Data[25]<<8)+(DataArray.Data[26]);
                        Sat_Dat.t_oc[i]=TempData.FloatValue;

            TempData.HexValue=(DataArray.Data[27]<<24)+(DataArray.Data[28]<<16)+(Da
taArray.Data[29]<<8)+(DataArray.Data[30]);
                        Sat_Dat.a_f2[i]=TempData.FloatValue;

            TempData.HexValue=(DataArray.Data[31]<<24)+(DataArray.Data[32]<<16)+(Da
taArray.Data[33]<<8)+(DataArray.Data[34]);
                        Sat_Dat.a_f1[i]=TempData.FloatValue;

            TempData.HexValue=(DataArray.Data[35]<<24)+(DataArray.Data[36]<<16)+(Da
taArray.Data[37]<<8)+(DataArray.Data[38]);
                        Sat_Dat.a_f0[i]=TempData.FloatValue;
                        Sat_Dat.IODE[i]=DataArray.Data[43];

            TempData.HexValue=(DataArray.Data[45]<<24)+(DataArray.Data[46]<<16)+(Da
taArray.Data[47]<<8)+(DataArray.Data[48]);
                        Sat_Dat.C_rs[i]=TempData.FloatValue;

            TempData.HexValue=(DataArray.Data[49]<<24)+(DataArray.Data[50]<<16)+(Da
taArray.Data[51]<<8)+(DataArray.Data[52]);
                        Sat_Dat.delta_n[i]=TempData.FloatValue;
                        for(j=0;j<8;j++)
```

```
                  {
                        TempDouble.byte[7-j]= DataArray.Data[53+j];
                  }
                  Sat_Dat.M_0[i]=TempDouble.DoubleValue;
                  orbital.M[i]=TempDouble.DoubleValue;
      TempData.HexValue=(DataArray.Data[61]<<24)+(DataArray.Data[62]<<16)+(Da
taArray.Data[63]<<8)+(DataArray.Data[64]);
                  Sat_Dat.C_uc[i]=TempData.FloatValue;
                  for(j=0;j<8;j++)
                  {
                        TempDouble.byte[7-j]= DataArray.Data[65+j];
                  }
                  Sat_Dat.e[i]=TempDouble.DoubleValue;

      TempData.HexValue=(DataArray.Data[73]<<24)+(DataArray.Data[74]<<16)+(Da
taArray.Data[75]<<8)+(DataArray.Data[76]);
                  Sat_Dat.C_us[i]=TempData.FloatValue;
                  for(j=0;j<8;j++)
                  {
                        TempDouble.byte[7-j]= DataArray.Data[77+j];
                  }
                  Sat_Dat.sqrt_A[i]=TempDouble.DoubleValue;

      TempData.HexValue=(DataArray.Data[85]<<24)+(DataArray.Data[86]<<16)+(Da
taArray.Data[87]<<8)+(DataArray.Data[88]);
                  Sat_Dat.t_oe[i]=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[89]<<24)+(DataArray.Data[90]<<16)+(Da
taArray.Data[91]<<8)+(DataArray.Data[92]);
                  Sat_Dat.C_ic[i]=TempData.FloatValue;
                  for(j=0;j<8;j++)
                  {
                        TempDouble.byte[7-j]= DataArray.Data[93+j];
                  }
                  Sat_Dat.OMEGA_0[i]=TempDouble.DoubleValue;
                  if(Sat_Dat.OMEGA_0[i]<0)
                  {
                        Sat_Dat.OMEGA_0[i]=Sat_Dat.OMEGA_0[i]+M_PI;
                  }
                  orbital.Thk[i]=TempDouble.DoubleValue;

      TempData.HexValue=(DataArray.Data[101]<<24)+(DataArray.Data[102]<<16)+(
DataArray.Data[103]<<8)+(DataArray.Data[104]);
                  Sat_Dat.C_is[i]=TempData.FloatValue;
                  for(j=0;j<8;j++)
                  {
                        TempDouble.byte[7-j]= DataArray.Data[105+j];
                  }
                  Sat_Dat.i_0[i]=TempDouble.DoubleValue;

      TempData.HexValue=(DataArray.Data[113]<<24)+(DataArray.Data[114]<<16)+(
DataArray.Data[115]<<8)+(DataArray.Data[116]);
                  Sat_Dat.C_rc[i]=TempData.FloatValue;
                  for(j=0;j<8;j++)
                  {
                        TempDouble.byte[7-j]= DataArray.Data[117+j];
                  }
```

```
                    Sat_Dat.omega[i]=TempDouble.DoubleValue;
                    orbital.uk[i]=TempDouble.DoubleValue;

        TempData.HexValue=(DataArray.Data[125]<<24)+(DataArray.Data[126]<<16)+(
DataArray.Data[127]<<8)+(DataArray.Data[128]);
                    Sat_Dat.OMEGADOT[i]=TempData.FloatValue;

        TempData.HexValue=(DataArray.Data[129]<<24)+(DataArray.Data[130]<<16)+(
DataArray.Data[131]<<8)+(DataArray.Data[132]);
                    Sat_Dat.IDOT[i]=TempData.FloatValue;
                    for(j=0;j<8;j++)
                    {
                          TempDouble.byte[7-j]= DataArray.Data[141+j];
                    }
                    Sat_Dat.nn[i]=TempDouble.DoubleValue;
                    for(j=0;j<8;j++)
                    {
                          TempDouble.byte[7-j]= DataArray.Data[149+j];
                    }
                    Sat_Dat.r1me2[i]=TempDouble.DoubleValue;



                    Retry5:
                    res++;
                    for(j=0;j<180;j++)
                    {
                          DataArray.Data[j]=0;
                    }
                    if(Sat_Dat.Sat[i]==0x10)
                    {
                          m=sizeof(GetRaw10)/sizeof(uint8_t);
                          for(n=0;n<=m;n++)
                          {
                                myUSART_SendByte(GetRaw10[n]);
                          }
                    }
                    else
                    {
                          GetRaw[2]=Sat_Dat.Sat[i];
                          m=sizeof(GetRaw)/sizeof(uint8_t);
                          for(n=0;n<=m;n++)
                          {
                                myUSART_SendByte(GetRaw[n]);
                          }
                    }
                    DataArray=myUSART_TrapByte(0x5A,0);
                    if(DataArray.Check==1 && res<50){goto Retry5;}
                    ct=0;
                    res=0;
                    for(j=0;j<8;j++)
                    {
                          TempDouble.byte[7-j]= DataArray.Data[19+j];
                    }
                    Sat_Dat.TOM[i]=TempDouble.DoubleValue;
                    Sat_Dat.TDiff[i]=DataArray.Data[6];
```

```
      TempData.HexValue=(DataArray.Data[11]<<24)+(DataArray.Data[12]<<16)+(Da
taArray.Data[13]<<8)+(DataArray.Data[14]);
                  Sat_Dat.codephase[i]=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[7]<<24)+(DataArray.Data[8]<<16)+(Data
Array.Data[9]<<8)+(DataArray.Data[10]);
                  Sat_Dat.SigLev[i]=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[15]<<24)+(DataArray.Data[16]<<16)+(Da
taArray.Data[17]<<8)+(DataArray.Data[18]);
                  Sat_Dat.Doppler[i]=TempData.FloatValue;
            }


            for(i=k;i<12;i++)
            {
                  Sat_Dat.t_ephem[i]=0;
                  Sat_Dat.IODC[i]=0;
                  Sat_Dat.t_oc[i]=0;
                  Sat_Dat.a_f2[i]=0;
                  Sat_Dat.a_f1[i]=0;
                  Sat_Dat.a_f0[i]=0;
                  Sat_Dat.IODE[i]=0;
                  Sat_Dat.C_rs[i]=0;
                  Sat_Dat.delta_n[i]=0;
                  Sat_Dat.M_0[i]=0;
                  Sat_Dat.C_uc[i]=0;
                  Sat_Dat.e[i]=0;
                  Sat_Dat.C_us[i]=0;
                  Sat_Dat.sqrt_A[i]=0;
                  Sat_Dat.t_oe[i]=0;
                  Sat_Dat.C_ic[i]=0;
                  Sat_Dat.OMEGA_0[i]=0;
                  Sat_Dat.C_is[i]=0;
                  Sat_Dat.i_0[i]=0;
                  Sat_Dat.C_rc[i]=0;
                  Sat_Dat.omega[i]=0;
                  Sat_Dat.OMEGADOT[i]=0;
                  Sat_Dat.IDOT[i]=0;
                  Sat_Dat.nn[i]=0;
                  Sat_Dat.T_GD[i]=0;
                  Sat_Dat.TOM[i]=0;
            }

            //Get Time of Week
            Retry7:
            res++;
            for(i=0;i<180;i++)
            {
                  DataArray.Data[i]=0;
            }
            m=sizeof(GetTOW)/sizeof(uint8_t);
            for(n=0;n<=m;n++)
            {
                  myUSART_SendByte(GetTOW[n]);
            }
            DataArray=myUSART_TrapByte(0x41,0);
            if(DataArray.Check==1 && res<50){goto Retry7;}
            ct=0;
            res=0;

      TempData.HexValue=(DataArray.Data[2]<<24)+(DataArray.Data[3]<<16)+(Data
Array.Data[4]<<8)+(DataArray.Data[5]);
            Sat_Dat.TOW=TempData.FloatValue;
            //Get Time of Week
            Retry8:
            res++;
            for(i=0;i<500;i++)
            {
                  DataArray.Data[i]=0;
            }
            m=sizeof(GetChannel1)/sizeof(uint8_t);
            for(n=0;n<=m;n++)
            {
                  myUSART_SendByte(GetChannel1[n]);
            }
            DataArray=myUSART_TrapByte(0x8F,0x2A);
            if(DataArray.Check==1 && res<50){goto Retry8;}
            ct=0;
            res=0;
            TempData.HexValue=(DataArray.Data[5]<<8)+(DataArray.Data[6]);
            Sat_Dat.GC1weekn=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[7]<<24)+(DataArray.Data[8]<<16)+(Data
Array.Data[9]<<8)+(DataArray.Data[10]);
            Sat_Dat.GC1TOWms=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[11]<<24)+(DataArray.Data[12]<<16)+(Da
taArray.Data[13]<<8)+(DataArray.Data[14]);
            Sat_Dat.GC1Fractional=TempData.FloatValue;

      TempData.HexValue=(DataArray.Data[15]<<24)+(DataArray.Data[16]<<16)+(Da
taArray.Data[17]<<8)+(DataArray.Data[18]);
            Sat_Dat.GC1Altitude=TempData.FloatValue;
            for(i=0;i<12;i++)
            {
                  Sat_Dat.GC1SatID[i]=DataArray.Data[22+i*12];
                  Sat_Dat.GC1Sig[i]=(DataArray.Data[23+i*12]);

      Sat_Dat.GC1Aqu[i]=(DataArray.Data[24+i*12]<<8)+(DataArray.Data[25+i*12]
);

      Sat_Dat.GC1Psuedo[i]=(DataArray.Data[26+i*12]<<24)+(DataArray.Data[27+i
*12]<<16)+(DataArray.Data[28+i*12]<<8)+(DataArray.Data[29+i*12]);

      Sat_Dat.GC1RangeRate[i]=(DataArray.Data[30+i*12]<<24)+(DataArray.Data[3
1+i*12]<<16)+(DataArray.Data[32+i*12]<<8)+(DataArray.Data[33+i*12]);
            }
      }
//=========================================================================

void Satellite_Fix_A(void)
{
      uint8_t i;
      int8_t lvlchk;
      uint8_t k;
      uint8_t ss[4];
      uint8_t ss2[4];
      float eps;
      double ET;
      double ET_old;
    double Tmpt[5];
      double we;  double M;   double Ek;  double vk;  double thk; double drk;
      double duk;      double dik; double rk; double uk; double ik; double
OMk; double Xpr; double Ypr;
      double F=-4.442807633E-10; double fc=1575.42E6;
      eps=1E-5;
      we=7.2921151467E-5;

      float terror=1;
      double told;
      k=0;
      Restart:
      //Level Check
      for (i=0;i<4;i++)
      {
            lvlchk=-50;
            while(lvlchk<0)
            {
                  ss2[i]=k;
                  if ((Sat_Dat.GC1Sig[ss2[i]]>0) &&
(Sat_Dat.GC1Psuedo[ss2[i]]>0))
                  {
                        lvlchk=1;
                  }
                  k++;
            }
      }

      //Satellite Alignment
      k=0;
      for (i=0;i<12;i++)
      {
            if (Sat_Dat.Sat[i]==Sat_Dat.GC1SatID[ss2[k]])
            {
                  ss[k]=i;
                  k++;
            }
      }
      for (k=0;k<4;k++)
      {
            test[10+k]=Sat_Dat.Sat[ss[k]];
            test[14+k]=Sat_Dat.GC1SatID[ss2[k]];
      }
      float c =299792458;
//TIME

      for (i=0;i<4;i++)
      {
            terror=1;

            //Reset Variables

      M=0;Ek=0;vk=0;thk=0;drk=0;duk=0;dik=0;rk=0;uk=0;ik=0;OMk=0;Xpr=0;Ypr=0;
            Tmpt[i]=Sat_Dat.TOM[ss[i]]; //Check initializing
            Tmpt[i]=Sat_Dat.TOM[ss[i]];
            RR[i]=Sat_Dat.GC1Psuedo[ss2[i]]/100;
            tk[i]=Sat_Dat.TOM[ss[i]]-Sat_Dat.t_oe[ss[i]];
            if (tk[i]<-302400)
            {
                  tk[i]=tk[i]+604800;
            }
            else if (tk[i]>302400)
            {
                  tk[i]=tk[i]-604800;
            }
            M=((Sat_Dat.M_0[ss[i]])+(Sat_Dat.nn[ss[i]])*tk[i]);
            testd[i+4]=Sat_Dat.nn[ss[i]];
            ET=M; ET_old=.1;
            while(fabsf(ET-ET_old)>eps)
            {
                  ET_old=ET;
                  ET=ET_old-(ET_old-(Sat_Dat.e[ss[i]])*sin(ET_old)-M)/(1-
(Sat_Dat.e[ss[i]])*cos(ET_old));
            }
            Ek=ET;
            testd[i]=-F*Sat_Dat.e[ss[i]]*sin(Ek)*Sat_Dat.sqrt_A[ss[i]];

            vk=atan((sqrt(1-
Sat_Dat.e[ss[i]]*Sat_Dat.e[ss[i]])*sin(Ek)))/(cos(Ek)-Sat_Dat.e[ss[i]]);
            thk=vk+((Sat_Dat.omega[ss[i]]));

      drk=Sat_Dat.C_rc[ss[i]]*cos(2*thk)+Sat_Dat.C_rs[ss[i]]*sin(2*thk);

      duk=Sat_Dat.C_uc[ss[i]]*cos(2*thk)+Sat_Dat.C_us[ss[i]]*sin(2*thk);

      dik=Sat_Dat.C_ic[ss[i]]*cos(2*thk)+Sat_Dat.C_is[ss[i]]*sin(2*thk);
            rk=((Sat_Dat.sqrt_A[ss[i]]*Sat_Dat.sqrt_A[ss[i]]))*(1-
(Sat_Dat.e[ss[i]])*cos(Ek))+drk;
            uk=thk+duk;
            ik=(Sat_Dat.i_0[ss[i]])+dik+Sat_Dat.IDOT[ss[i]]*tk[i];
            OMk=(Sat_Dat.OMEGA_0[ss[i]])+(Sat_Dat.OMEGADOT[ss[i]]-we)*tk[i]-
we*Sat_Dat.t_oe[ss[i]];
            Xpr=rk*cos(uk);
            Ypr=rk*sin(uk);

            //Final Calculation
            X_PA[i]=(Xpr*cos(OMk)-Ypr*sin(OMk)*cos(ik));
            Y_PA[i]=(Xpr*sin(OMk)+Ypr*cos(OMk)*cos(ik));
            Z_PA[i]=(Ypr*sin(ik));
            T_PA[i]=Sat_Dat.codephase[ss[i]]/(16*1.023E6);
            if(X_PA[i]==0||Y_PA[i]==0||Z_PA[i]==0)
            {
                  goto Restart;
            }
      }
```

```
}
inv invmatrix(double a[4][4])
{
      uint8_t i;uint8_t j;uint8_t k;double s;uint8_t L;
      inv inv;
      double binv[4][4];
      double t;
      for (i=0;i<4;i++)
      {
            for (j=0;j<4;j++)
            {
                  if (i==j)
                  {
                        binv[i][j]=1;
                  }
                  else
                  {
                        binv[i][j]=0;
                  }
            }
      }

      for (j=0;j<4;j++)
      {
            for (i=j;i<4;i++)
            {
                  if (a[i][j]!=0)
                  {
                        for (k=0;k<4;k++)
                        {
                              s=a[j][k];
                              a[j][k]=a[i][k];
                              a[i][k]=s;
                              s=binv[j][k];
                              binv[j][k]=binv[i][k];
                              binv[i][k]=s;
                        }
                        t=1/a[j][j];
                        for (k=0;k<4;k++)
                        {
                              a[j][k]=t*a[j][k];
                              binv[j][k]=t*binv[j][k];
                        }
                        for (L=0;L<4;L++)
                        {
                              if(L!=j)
                              {
                                    t=-a[L][j];
                                    for (k=0;k<4;k++)
                                    {
                                          a[L][k]=a[L][k]+t*a[j][k];
                                          binv[L][k]=binv[L][k]+t*binv[j][k];
                                    }
                              }
                        }
                  }
```

```
			}
		}
		for (i=0;i<4;i++)
		{
			for (j=0;j<4;j++)
			{
				inv.binv[i][j]=binv[i][j];
			}
		}
		return(inv);
}


	///////////////////////////////////////////////////////////////////////
	//==============BFGS Algorithm=======================================//
	///////////////////////////////////////////////////////////////////////

void Position_Fix_A(void)
{
	float c =299792458;
	uint8_t j;
	uint8_t i;
	uint8_t k;
	uint8_t m;
	double h [8][8];
	double hh [8][8];
	double h_old [8][8];
	double F_old [8]={0,0,0,0,0,0,0,0,0,0};

	double Ff [8];
	double Fd[8];
	double Lagrangian;
	double Lagrangian_d;
	double Eta[8];
	double Etad[8];
	double q[8];

	double d [8];
	double s [8]={1,1,1,1,1,1,1,1,1,1};
	double g [8]={1,1,1,1,1,1,1,1,1,1};
	double gam;

	double sTg; double gTH[8]; double gTHg; double SST[8][8]; double gST
[8][8]; double SgT [8][8];

	double Q;
	double Accuracy;
	double CC;
	double XS[4];
	double YS[4];
	double ZS[4];
	double GG=1;
	double kap;
	double Gkf=100;
	//=====================initialize Inverse Hessian=============
	for (i=0;i<8;i++)
	{
```

```
            for (j=0;j<8;j++)
            {
                  if (i==j)
                  {
                        h[i][j]=0.1;//Free to change the initial value
                  }
                  else
                  {
                        h[i][j]=0;
                  }
            }
      }
      for (k=0;k<4;k++)
      {
            XS[k]=X_PA[k]/GG;
            YS[k]=Y_PA[k]/GG;
            ZS[k]=Z_PA[k]/GG;
            R[k]=RR[k]/GG;
      }
      //========================Functional Initialization=========
      q[0]=-(2000+517036)/GG;
      q[1]=-(2000+4335706)/GG;
      q[2]=(-2000+4633982)/GG;
      q[3]=10/GG;
      q[4]=1;
      q[5]=1;
      q[6]=1;
      q[7]=1;
      Eta[0]=-(XS[0]-q[0])*(XS[0]-q[0])*(XS[0]+2*q[0])/3;
      Eta[1]=-(YS[1]-q[1])*(YS[1]-q[1])*(YS[1]+2*q[1])/3;
      Eta[2]=-(ZS[2]-q[2])*(ZS[2]-q[2])*(ZS[2]+2*q[2])/3;
      Eta[3]=(R[3]-q[3])*(R[3]-q[3])*(R[3]+2*q[3])/3;
      Eta[4]=0;
      Eta[5]=0;
      Eta[6]=0;
      Eta[7]=0;

      for(k=0;k<4;k++)
      {
          Ff[k]=(XS[k]-q[0])*(XS[k]-q[0])+(YS[k]-q[1])*(YS[k]-q[1])+(ZS[k]-
q[2])*(ZS[k]-q[2])-(R[k]+q[3])*(R[k]+q[3]);
      }
      Ff[4]=(XS[1]-q[0])*q[1]+(XS[2]-q[0])*q[2]+(XS[3]-q[0])*q[3];
      Ff[5]=(YS[0]-q[1])*q[0]+(YS[2]-q[1])*q[2]+(YS[3]-q[1])*q[3];
      Ff[6]=(ZS[0]-q[2])*q[0]+(ZS[1]-q[2])*q[1]+(ZS[3]-q[2])*q[3];
      Ff[7]=-(R[0]+q[3])*q[0]-(R[1]+q[3])*q[1]-(R[2]+q[3])*q[2];

      //==================Minimization========================================
      Accuracy=1e-15;
      float st;
      uint iter=1;
      for(st=0;st<iter;st++)
      {
           //======== Cost functional G(q) ============================
          Lagrangian=0;
          for (i=0;i<8;i++)
          {
              Lagrangian=Lagrangian+(Ff[i]*q[i]+Eta[i]);
          }
          //========== d(n) ============================================
          for(i=0;i<8;i++)
          {
              d[i]=0;
              for(j=0;j<8;j++)
              {
                  d[i]=d[i]-h[i][j]*Ff[j];
              }
          }
          //=========================== gamma ===========================
          Gkf=0;
          for (i=0;i<8;i++)
          {
             Gkf= Gkf+Ff[i]*Ff[i];
          }
          gam=1;

          //======================== s(n), q(n) ============================
          for(k=0;k<8;k++)
          {
              s[k]=gam*d[k];
              q[k]=q[k]+gam*d[k];
          }

          //============================ f(n) and g(n) ===================
          for(k=0;k<8;k++)
          {
              F_old[k]=Ff[k];
          }
          Eta[0]=-(XS[0]-q[0])*(XS[0]-q[0])*(XS[0]+2*q[0])/3;
          Eta[1]=-(YS[1]-q[1])*(YS[1]-q[1])*(YS[1]+2*q[1])/3;
          Eta[2]=-(ZS[2]-q[2])*(ZS[2]-q[2])*(ZS[2]+2*q[2])/3;
          Eta[3]=(R[3]-q[3])*(R[3]-q[3])*(R[3]+2*q[3])/3;
          Eta[4]=0;
          Eta[5]=0;
          Eta[6]=0;
          Eta[7]=0;
          for(k=0;k<4;k++)
          {
              Ff[k]=(XS[k]-q[0])*(XS[k]-q[0])+(YS[k]-q[1])*(YS[k]-
q[1])+(ZS[k]-q[2])*(ZS[k]-q[2])-(R[k]+q[3])*(R[k]+q[3]);
          }
          Ff[4]=(XS[1]-q[0])*q[1]+(XS[2]-q[0])*q[2]+(XS[3]-q[0])*q[3];
          Ff[5]=(YS[0]-q[1])*q[0]+(YS[2]-q[1])*q[2]+(YS[3]-q[1])*q[3];
          Ff[6]=(ZS[0]-q[2])*q[0]+(ZS[1]-q[2])*q[1]+(ZS[3]-q[2])*q[3];
          Ff[7]=-(R[0]+q[3])*q[0]-(R[1]+q[3])*q[1]-(R[2]+q[3])*q[2];


          for(i=0;i<8;i++)
          {
              g[i]=Ff[i]-F_old[i];
          }

          //========================= h(n+1) =========================
          for (i=0;i<8;i++)
          {
              for (j=0;j<8;j++)
              {
                  h_old[i][j]=h[i][j];
              }
          }
          //==================== Compute scalar sTg  =================

          sTg=0;
          for (i=0;i<8;i++)
          {
              sTg=sTg+s[i]*g[i];
          }

          //========== Compute scalar gTHg  ================
          for (i=0;i<8;i++)
          {
              gTH[i]=0;
              for (j=0;j<8;j++)
              {
                  gTH[i]=gTH[i]+g[j]*h_old[j][i];
              }
          }

          gTHg=0;
          for (i=0;i<8;i++)
          {
              gTHg=gTHg+gTH[i]*g[i];
          }
          //==== Compute rank one matrices ssT, gsT and sgT ===

          for (i=0;i<8;i++)
          {
              for (j=0;j<8;j++)
              {
                  SST[i][j]=s[i]*s[j]/sTg;
                  gST[i][j]=g[i]*s[j]/sTg;
                  SgT[i][j]=s[i]*g[j]/sTg;
              }
          }
          //================== h(n+1) ============= =================
          for (i=0;i<8;i++) //  %This part had an error
          {
              for (j=0;j<8;j++)
              {
                  hh[i][j]=0;
                  for (k=0;k<8;k++)
                  {

hh[i][j]=hh[i][j]+h_old[i][k]*gST[k][j]+SgT[i][k]*h_old[k][j];
                  }
                  hh[i][j]=hh[i][j]+(1+gTHg/sTg)*SST[i][j];
              }
          }
          //======== Make sure h(n,n) does not explode because of low sTg
if(sTg>Accuracy)
          {
              for (i=0;i<8;i++)
              {
                  for (j=0;j<8;j++)
                  {
                      h[i][j]=h_old[i][j]+hh[i][j];
                  }
              }
          }
}

      X_r_BFGS=q[0]*GG;
      Y_r_BFGS=q[1]*GG;
      Z_r_BFGS=q[2]*GG;

}

void Position_Fix_B(void)
{
      uint8_t j;
      uint8_t i;
      uint8_t k;
      uint8_t L;
      float xx;
      double q[4];
      double dq[4];
      double XS[4];//={12295772,-17793958,-8059098,-133015};
      double YS[4];//={-14225410,-5630114,-23261044,-26297132};
      double ZS[4];//={18708165,18908477,9944606,252052};
      double ATA[4][4];
      double InvA[4][4];
      double GG=1;
      double A[4][4];
      double Er;
      double IATA[4][4];
      double F[4];
      double Fd[4];
      double qn[4];
      double dqold[4];
      inv binv;
      float Lambda=10;
      double Cd;
      double Cf;
      float c =299792458;
//FIX
      for (k=0;k<4;k++)
      {
            XS[k]=X_PA[k]/GG;
            YS[k]=Y_PA[k]/GG;
            ZS[k]=Z_PA[k]/GG;
            R[k]=RR[k]/GG;
      }

      q[0]=-(10+517042)/GG;
      q[1]=-(10+4335720)/GG;
```

```
       q[2]=(-10+4633945)/GG;
       q[3]=100/GG;
       dq[0]=1;
       dq[1]=1;
       dq[2]=1;
       dq[3]=1;

       Cf=0;
       for (i=0;i<4;i++)
       {
           F[i]=(XS[i]-q[0])*(XS[i]-q[0])+(YS[i]-q[1])*(YS[i]-q[1])+(ZS[i]-
q[2])*(ZS[i]-q[2])-(R[i]+q[3])*(R[i]+q[3]);
           Cf=Cf+F[i]*F[i];
       }

       Er=10000;
       while(Er>0.5)
       {
           A[0][0]=-2*(XS[0]-q[0]);
           A[0][1]=-2*(YS[0]-q[1]);
           A[0][2]=-2*(ZS[0]-q[2]);
           A[0][3]=-2*(R[0]+q[3]);

           A[1][0]=-2*(XS[1]-q[0]);
           A[1][1]=-2*(YS[1]-q[1]);
           A[1][2]=-2*(ZS[1]-q[2]);
           A[1][3]=-2*(R[1]+q[3]);

           A[2][0]=-2*(XS[2]-q[0]);
           A[2][1]=-2*(YS[2]-q[1]);
           A[2][2]=-2*(ZS[2]-q[2]);
           A[2][3]=-2*(R[2]+q[3]);

           A[3][0]=-2*(XS[3]-q[0]);
           A[3][1]=-2*(YS[3]-q[1]);
           A[3][2]=-2*(ZS[3]-q[2]);
           A[3][3]=-2*(R[3]+q[3]);

           for (i=0;i<4;i++)
           {
               for (j=0;j<4;j++)
               {
                   if (j==i)
                   {
                       ATA[i][j]=Lambda;
                   }
                   else
                   {
                       ATA[i][j]=0;
                   }
                   for (k=0;k<4;k++)
                   {
                       ATA[i][j]=ATA[i][j]+A[k][i]*A[k][j]; //% Check A'*A
                   }
               }
           }
```

```
        binv=invmatrix(ATA);
        for (i=0;i<4;i++)
        {
            for (j=0;j<4;j++)
            {
                InvA[i][j]=binv.binv[i][j];
            }
        }

        for (i=0;i<4;i++)
        {
            for (j=0;j<4;j++)
            {
                IATA[i][j]=0;
                for (k=0;k<4;k++)
                {
                    IATA[i][j]=IATA[i][j]+InvA[i][k]*A[j][k];
                }
            }
        }


        for(i=0;i<4;i++)
        {
              dqold[i]=dq[i];
            dq[i]=0;
            for (j=0;j<4;j++)
            {
                dq[i]=dq[i]-IATA[i][j]*F[j];
            }
        }

        for (k=0;k<4;k++)
        {
            qn[k]=q[k];
            q[k]=qn[k]+dq[k];
        }
        Cd=0;
        for (k=0;k<4;k++)
        {
            Fd[k]=(XS[k]-q[0])*(XS[k]-q[0])+(YS[k]-q[1])*(YS[k]-
q[1])+(ZS[k]-q[2])*(ZS[k]-q[2])-(R[k]+q[3])*(R[k]+q[3]);
            Cd=Cd+Fd[k]*Fd[k];
        }
        if (Cd>Cf)
        {

            for (k=0;k<4;k++)
            {
                q[k]=qn[k];
            }
            Lambda=Lambda*10;
        }
        else
        {
            Cf=Cd;
            for (i=0;i<4;i++)
```

```
                {
                    F[i]=Fd[i];
                }
                Lambda=Lambda/2;
                Er=sqrt(Cf);
            }
                    X_r_LS=q[0]*GG;
                    Y_r_LS=q[1]*GG;
                    Z_r_LS=q[2]*GG;
                    test[6]=(float) Er;
        }
}
void Position_Fix_C(void)
{
        uint8_t j;
        uint8_t i;
        uint8_t k;
        uint8_t m;
        float F_old [4]={0,0,0,0};
        float A[4][4];
        float Ainv[4][4];
        float Ff[4];
        float Eta[4];
        float Etad[4];
        float q[4];

        float d [4];
        float gam;
        inv binv;
        float Accuracy;
        float XS[4];
        float YS[4];
        float ZS[4];
        float GG=1E8;
        float Gkf=100;
        for (k=0;k<4;k++)
        {
              XS[k]=X_PA[k]/GG;
              YS[k]=Y_PA[k]/GG;
              ZS[k]=Z_PA[k]/GG;
              R[k]=RR[k]/GG;
        }
        //====Functional Initialization=
        q[0]=-(10+517036)/GG;
        q[1]=-(10+4335706)/GG;
        q[2]=(-10+4633982)/GG;
        q[3]=10;
        for(k=0;k<4;k++)
        {
            Ff[k]=(XS[k]-q[0])*(XS[k]-q[0])+(YS[k]-q[1])*(YS[k]-q[1])+(ZS[k]-
q[2])*(ZS[k]-q[2])-(R[k]+q[3])*(R[k]+q[3]);
        }
        //==========Minimization==================================
        Accuracy=1e-15;
        uint st;
        uint iter=1000;
        for(st=0;st<iter;st++)
```

```
      {
            for (i=0;i<4;i++)
            {
                  A[i][0]=2*(XS[i]-q[0]);
                  A[i][1]=2*(YS[i]-q[1]);
                  A[i][2]=2*(ZS[i]-q[2]);
                  A[i][3]=-2*(R[i]-q[3]);
            }
            binv=invmatrix(A);
            for (i=0;i<4;i++)
            {
                  for (j=0;j<4;j++)
                  {
                        Ainv[i][j]=binv.binv[i][j];
                  }
            }
            //============================ d(n) ===========================

          for(i=0;i<4;i++)
          {
              d[i]=0;
              for(j=0;j<4;j++)
              {
                  d[i]=d[i]-Ainv[i][j]*Ff[j];
              }
          }
          //========================== gamma ===========================
          Gkf=0;
          for (i=0;i<4;i++)
          {
             Gkf= Gkf+Ff[i]*Ff[i];
          }
          gam=(1E-13)*sqrtf(Gkf);
          //==================== s(n), q(n) =================================
          for(k=0;k<4;k++)
          {
              q[k]=q[k]+gam*d[k];
          }

          //======================== f(n) and g(n) ====================
          for(k=0;k<4;k++)
          {
              F_old[k]=Ff[k];
          }
          for(k=0;k<4;k++)
          {
              Ff[k]=(XS[k]-q[0])*(XS[k]-q[0])+(YS[k]-q[1])*(YS[k]-
q[1])+(ZS[k]-q[2])*(ZS[k]-q[2])-(R[k]+q[3])*(R[k]+q[3]);
          }
}

      X_r_NR=q[0]*GG;
      Y_r_NR=q[1]*GG;
      Z_r_NR=q[2]*GG;

}
```

```
//==============================================================================
int main(void)
{
      Configure_CopernicusGPS();
      test[5]=0;
    while(1)
    {
      THUNDAR_GPS();
      Satellite_Fix_A();
      Position_Fix_A();
      Position_Fix_B();
      Position_Fix_C();
    }
}
```

# APPENDIX B. DYNAMIC MODEL SIMULATION CODE

```
clear
clc
IT=0; %counter i9terations
Dt=.785;
Dte=.785;
gam=0.1;
kb=0.154;
kt=0.13;
nr=20;
Iw=0.764;
m=230;
Nc=2014;
B=0.88;
Icr=127.7;
rw=0.2794;
Ra=0.32;
%Heading Angles and delta variables
THk=0;
thi=0;
Dx=0;
Dy=0;
DTHk=0;


%Path definition

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %PATH 1: Zigzag
% %%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Lmax=4.5;
%    f=1.4;
%
%  path(2,1)=0;        path(2,2)=1;        path(2,3)=2;
path(2,4)=3;        path(2,5)=4;        path(2,6)=5;        path(2,7)=6;
%  path(3,1)=0;        path(3,2)=1;        path(3,3)=2;
path(3,4)=3;        path(3,5)=4;        path(3,6)=5;        path(3,7)=6;
%
%  path(2,8)=7;        path(2,9)=8;        path(2,10)=9;
%  path(3,8)=6;        path(3,9)=6;        path(3,10)=6;
%
%  path(2,11)=10;        path(2,12)=11;        path(2,13)=12;
%  path(3,11)=6;        path(3,12)=6;        path(3,13)=6;
%
%
%  path(2,14)=13;        path(2,15)=14;        path(2,16)=15;
%  path(3,14)=7;        path(3,15)=8;        path(3,16)=9;
%
%  path(2,17)=16;        path(2,18)=17;        path(2,19)=18;
%  path(3,17)=10;        path(3,18)=11;        path(3,19)=12;
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PATH 2: Semicircle-
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lmax=3.58;
%   f=1.5;
%   NPP=11;
%   radius=20;
%   path(:,1)=0;
%   for k=2:2*NPP
%       path(2,k)=radius*(1-cos(k*0.5*pi/NPP));
%       path(3,k)=radius*sin(k*0.5*pi/NPP);
%   end
%
%  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PATH 3 :Sinusoid
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   Lmax=3;
   f=1.2
   NPP=45
   path(2,:)=[0:NPP];
   path(3,:)=10*sin([0:NPP]*4*pi/(2*NPP));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Traj=[];
X=0;
Y=0;
Pid=2;
xold=[4;4;4;0;0;0;0];
State=[];
Cont=[];
LPATH=length(path);
while (Pid<LPATH+1)
    Traj=[Traj;X,Y];
    DX=path(2,Pid)-X;
    DY=path(3,Pid)-Y;
    L=sqrt(DX^2+DY^2);

     if(L>Lmax)
        L=Lmax;
     end
    Umax=f*L;
    if((abs(DX)<1e-28)&&(DY>0))
        THk=pi/2;
        disp('Zero DX POS DY')
    elseif((abs(DY)<1e-28)&&(DX>0))
        THk=0;
        disp('Zero DY POS DX')
    elseif((abs(DX)<1e-28)&&(DY<0))
        THk=3*pi/2;
        disp('Zero DX NEG DY')
    elseif((abs(DY)<1e-28)&&(DX<0))
```

```
        THk=pi;
        disp('Zero DY NEG DX')
    elseif((DX>0)&&(DY<0))
       % THk=-atan(abs(DY)/(DX));
       THk=2*pi-atan(abs(DY)/DX);
        disp('DY')
    elseif((DX<0)&&(DY<0))
        THk=pi+atan((abs(DY))/(abs(DX)));
        disp('DX and DY')
    elseif((DX<0)&&(DY>0))
        THk=pi-atan(DY/(abs(DX)));
        disp('DX')
    elseif ((DX>0)&&(DY>0))
        THk=atan(DY/DX);
        disp('Normal')
    else
        disp('Undetermined')

    end
    DTH=THk-xold(5);
    thi=THk;
    xold(5)=thi;
    State=[State,xold];

%============================================================================
%%%% Define System Matrices
%================================== Matrix F(k)====================
    Fk(1,1)=1-(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw));    Fk(1,2)=0;
           Fk(1,3)=0;                        Fk(1,4)=0;
      Fk(1,5)=0;       Fk(1,6)=0;        Fk(1,7)=0;
    Fk(2,1)=0;                   Fk(2,2)=1-
(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw)); Fk(2,3)=0;
      Fk(2,4)=0;                   Fk(2,5)=0;       Fk(2,6)=0;
      Fk(2,7)=0;
    Fk(3,1)=0;                   Fk(3,2)=0;                  Fk(3,3)=1-
(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw)); Fk(3,4)=0;
      Fk(3,5)=0;       Fk(3,6)=0;        Fk(3,7)=0;
    Fk(4,1)=0;                   Fk(4,2)=0;                  Fk(4,3)=0;
                 Fk(4,4)=1+(B*B*kt*kb*nr*nr*Dt)/(4*Icr*Ra*rw*rw);
      Fk(4,5)=0;       Fk(4,6)=0;        Fk(4,7)=0;
    Fk(5,1)=0;                   Fk(5,2)=0;                  Fk(5,3)=0;
                 Fk(5,4)=(B*B*kt*kb*nr*nr*Dt)/(4*Icr*Ra*rw*rw)*Dt;
      Fk(5,5)=1;       Fk(5,6)=0;        Fk(5,7)=0;
    Fk(6,1)=0;                   Fk(6,2)=0;                  Fk(6,3)=-
(2*kt*kb*nr*nr*Dt)*Dt*cos(thi)/(Ra*(4*Iw+m*rw*rw));Fk(6,4)=0;
      Fk(6,5)=0;       Fk(6,6)=1;        Fk(6,7)=0;
    Fk(7,1)=0;                   Fk(7,2)=0;                  Fk(7,3)=-
(2*kt*kb*nr*nr*Dt)*Dt*sin(thi)/(Ra*(4*Iw+m*rw*rw));Fk(7,4)=0;
      Fk(7,5)=0;       Fk(7,6)=0;        Fk(7,7)=1;
    Hk(3,3)=-76.23;
    Hk(4,3)=-76.23;
    %======================== Matrix G(k)==========================
    Gk(1,1)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));                  Gk(1,2)=0;
    Gk(2,1)=0;
Gk(2,2)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));
    Gk(3,1)=(kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));
Gk(3,2)=(kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));
    Gk(4,1)=(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw);                  Gk(4,2)=-
(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw);
    Gk(5,1)=(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw)*Dt;               Gk(5,2)=-
(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw)*Dt;

    Gk(6,1)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*cos(thi);
      Gk(6,2)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*cos(thi);
    Gk(7,1)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*sin(thi);
      Gk(7,2)=(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*sin(thi);

    %================= Matrix H(k)==================== ==================
    Hk(1,1)=Nc*Dte/(2*pi*rw);     Hk(1,2)=0;                   Hk(1,3)=0;
                Hk(1,4)=0;                  Hk(1,5)=0;         Hk(1,6)=0;
      Hk(1,7)=0;
    Hk(2,1)=0;                    Hk(2,2)=Nc*Dte/(2*pi*rw);
Hk(2,3)=0;                  Hk(2,4)=0;                   Hk(2,5)=0;
      Hk(2,6)=0;       Hk(2,7)=0;
    Hk(3,1)=0;                    Hk(3,2)=0;
      Hk(3,3)=1/Dt;          Hk(3,4)=0;                  Hk(3,5)=0;
      Hk(3,6)=0;       Hk(3,7)=0;
    Hk(4,1)=0;                    Hk(4,2)=0;
      Hk(4,3)=1/Dt;          Hk(4,4)=0;                  Hk(4,5)=0;
      Hk(4,6)=0;       Hk(4,7)=0;
    Hk(5,1)=0;                    Hk(5,2)=0;                   Hk(5,3)=0;
                Hk(5,4)=1;                  Hk(5,5)=0;         Hk(5,6)=0;
      Hk(5,7)=0;
    Hk(6,1)=0;                    Hk(6,2)=0;                   Hk(6,3)=0;
                Hk(6,4)=0;                  Hk(6,5)=1/Dt;
      Hk(6,6)=0;       Hk(6,7)=0;
    Hk(7,1)=0;                    Hk(7,2)=0;                   Hk(7,3)=0;
                Hk(7,4)=0;                  Hk(7,5)=0;         Hk(7,6)=1;
      Hk(7,7)=0;
    Hk(8,1)=0;                    Hk(8,2)=0;                   Hk(8,3)=0;
                Hk(8,4)=0;                  Hk(8,5)=0;         Hk(8,6)=0;
      Hk(8,7)=1;
    %================== Matrix J(k)===================
    Jk(1,1)=0;                    Jk(1,2)=0;
    Jk(2,1)=0;                    Jk(2,2)=0;
    Jk(3,1)=0*3.46;               Jk(3,2)=0*3.46;
    Jk(4,1)=0*3.46;               Jk(4,2)=0*3.46;
    Jk(5,1)=0;                    Jk(5,2)=0;
    Jk(6,1)=0;                    Jk(6,2)=0;
    Jk(7,1)=0;                    Jk(7,2)=0;
    Jk(8,1)=0;                    Jk(8,2)=0;

    %=========== Matrix E1=======================
    E1(1,1)=0;                    E1(1,2)=0;                   E1(1,3)=0;
                E1(1,4)=0;                  E1(1,5)=1;         E1(1,6)=0;
      E1(1,7)=0;
```

```
E1(2,1)=0;                    E1(2,2)=0;                    E1(2,3)=0;
             E1(2,4)=0;                    E1(2,5)=0;        E1(2,6)=1;
  E1(2,7)=0;
E1(3,1)=0;                    E1(3,2)=0;                    E1(3,3)=0;
             E1(3,4)=0;                    E1(3,5)=0;        E1(3,6)=0;
  E1(3,7)=1;
%============================ Matrix E2================
E2(1,1)=-0;        E2(1,2)=-2*Dt/B;         E2(1,3)=2*Dt/B;
       E2(1,4)=-Dt/B;                E2(1,5)=DTH/(xold(5));
  E2(1,6)=0;       E2(1,7)=0;

E2(2,1)=Dt*cos(THk);     E2(2,2)=Dt*cos(THk);   E2(2,3)=-Dt*cos(THk);
  E2(2,4)=0;                    E2(2,5)=0;       E2(2,6)=DX/(xold(6));
  E2(2,7)=0;
E2(3,1)=Dt*sin(THk);     E2(3,2)=Dt*sin(THk);   E2(3,3)=-Dt*sin(THk);
  E2(3,4)=0;                    E2(3,5)=0;       E2(3,6)=0;
  E2(3,7)=DY/(xold(7));


for (i=1:3),
    for (j=1:7),
        E2(i,j)=E2(i,j);%0;
    end
end
%============================= Matrix W========================
for (i=1:7)
    for (j=1:7)
        if (i==j)
            W(i,j)=1;
        else
            W(i,j)=0;
        end;
    end;
end;
%========================= Covariance P===========================
for (i=1:7)
    for (j=1:7)
        if (i==j)
            P(i,j)=0.001;
        else
            P(i,j)=0;
        end;
    end;
end;
%========================= Covariance Q================================
for (i=1:8)
    for (j=1:8)
        if (i==j)
            Q(i,j)=0.001;
        else
            Q(i,j)=0;
        end;
    end;
end;
```

```
%======================== Uncertainty matrix A_\xi=====
Ax(7,7)=0;
for (i=1:7),
    for (j=1:4),
        if (i==j)
            Ax(i,j)=.5;
        else
            Ax(i,j)=0;
        end;
    end;
    for (j=4:7),
        Ax(i,j)=0;
    end;
end;
Ax(4,3)=0.05;
Ax(5,2)=0.025;
Ax(6,2)=0.025;
%===================== Uncertainty matrix A_Y=====================
Ay(8,7)=0;
Ay(2,2)=.01;
Ay(3,2)=.05;
Ay(4,4)=.068;%.679999977354;
Ay(5,4)=.068;%.679999977354;
Ay(7,6)=0;
%================ Uncertainty matrix B1=================
for (i=1:7)

    for (j=1:7)

        if (i==j)

            B1(i,j)=0.01;%.25;
        else

            B1(i,j)=0;
        end;
    end;
end;
%================== Uncertainty matrix B2============
for (i=1:7)

    for (j=1:2)

        if (i==j)

            B2(i,j)=0.5;%.5;
        elseif(i==3*j)

            B2(i,j)=0.5;%.5;
        else
            B2(i,j)=0.1;%.5;
        end;
    end;
end;
```

```
%==========================================================================
    %%%%
    %%%% START With The Robust Kalman Filter
    %%%%
    %============ Initial Ups ================
    for (i=1:7),
        for (j=1:7),
            if (i==j),
                UpsNew(i,j)=100;%0.5;  %%%%%%%Changed the variable name
            else

                UpsNew(i,j)=0;   %%%%%%%Changed the variable name
            end;
        end;
    end;
    Converg=10e8; %%%%%%%Convergence value
    while(Converg>0.00001) %%%%%%%Run in a loop

        for i=1:7
            for j=1:7
                Ups(i,j)=UpsNew(i,j);
            end
        end
        %=============== Calculate inverse Ups =============
        binv7=inv(Ups);
        for (i=1:7)

            for (j=1:7)

                InvUps(i,j)=binv7(i,j);
            end
        end
        %============= Calculate invXi ===============================
        for (i=1:7)

            for (j=1:7)

                invXi(i,j)=0;
                for (k=1:7)

                    invXi(i,j)=invXi(i,j)+B1(k,i)*B1(k,j);
                end
                %%%invXi(i,j)=invXi(i,j)-
gam*invXi(i,j);%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                invXi(i,j)=InvUps(i,j)-invXi(i,j)*gam*gam;

            end
        end
        %========================= Calculate Xi ==================
        binv7=inv(invXi);
        for (i=1:7)

            for (j=1:7)
```

```
                Xi(i,j)=binv7(i,j);
            end
        end
        %=========== Calculate Nk ================
        for (i=1:7)
            for (j=1:8)
                XiH(i,j)=0;
                for (k=1:7)
                    XiH(i,j)=XiH(i,j)+Xi(i,k)*Hk(j,k);
                end
            end
        end
        for (i=1:8)
            for(j=1:8)
                HXiH(i,j)=0;
                for (k=1:7)
                    HXiH(i,j)= HXiH(i,j)+Hk(i,k)*XiH(k,j);
                end
            end
        end
        for (i=1:8)
            for (j=1:8)
                AYAT(i,j)=0;
                for (k=1:7)
                    AYAT(i,j)=AYAT(i,j)+Ay(i,k)*Ay(j,k);
                end
                Nk(i,j)=Q(i,j)+AYAT(i,j)*gam*gam+HXiH(i,j);
            end
        end

        %======================== Calculate Mk =========================
        for (i=1:7)

            for (j=1:8)

                FXiH(i,j)=0;
                for (k=1:7)

                    FXiH(i,j)=FXiH(i,j)+Fk(i,k)*XiH(k,j);
                end
            end
        end
        for (i=1:7)
            for (j=1:8)
                AxAT(i,j)=0;
                for (k=1:7)
                    AxAT(i,j)=AxAT(i,j)+Ax(i,k)*Ay(j,k);
                end
                Mk(i,j)=FXiH(i,j)+AxAT(i,j)*gam*gam;
            end
        end

%==========================================================================
        %   Update covariances and return Ups

%==========================================================================
        %Solve Riccati Equation
        %
        %
        for (i=1:7)
            for (j=1:7)
                AxAxT(i,j)=0;
                for (k=1:7)
                    AxAxT(i,j)= AxAxT(i,j)+Ax(i,k)*Ax(j,k);
                end
            end
        end

%==========================================================================
        %%compute inverse of Nk
        binv8=inv(Nk);
        for (i=1:8)
            for (j=1:8)
                NkInv(i,j)=binv8(i,j);
            end
        end

%==========================================================================
        % %compute InvNkMkT
        for (i=1:8)
            for (j=1:7)
                InvNkMkT(i,j)=0;
                for (k=1:8)
                    InvNkMkT(i,j)=InvNkMkT(i,j)+NkInv(i,k)*Mk(j,k);
                end
            end
        end

%==========================================================================
        % %Now compute MkInvNkMkT
        for (i=1:7)
            for (j=1:7)
                MkInvNkMkT(i,j)=0;
                for (k=1:8)
                    MkInvNkMkT(i,j)=MkInvNkMkT(i,j)+Mk(i,k)*InvNkMkT(k,j);
                end
            end
        end

%==========================================================================
        %%Compute XiFT
        for (i=1:7)
            for (j=1:7)
                XiFT(i,j)=0;
                for (k=1:7)
                    XiFT(i,j)= XiFT(i,j)+Xi(i,k)*Fk(j,k);
                end
            end
        end

%==========================================================================
        %%Now compute FXiFT
        for (i=1:7),
            for (j=1:7),
                FXiFT(i,j)=0;
                for (k=1:7),
                    FXiFT(i,j)=FXiFT(i,j)+Fk(i,k)*XiFT(k,j);
                end
            end
        end
        %============================== Calculate New Ups ===============
        for (i=1:7)

            for (j=1:7)

                UpsNew(i,j)=FXiFT(i,j)- MkInvNkMkT(i,j)+
AxAxT(i,j)*gam*gam+P(i,j);
            end
        end

%==========================================================================
%%= Check if Ups has converged ===============================
        Converg=0;
        for i=1:7
            for j=1:7
                CVG(i,j)=UpsNew(i,j)-Ups(i,j);
                Converg=Converg+(UpsNew(i,j)-Ups(i,j))*(UpsNew(i,j)-
Ups(i,j));
            end
        end
        Converg=sqrt(Converg);
        IT=IT+1;
        %Ups
        UpsNew;

    end
    %pause
    %IT=0;

    %================== Calculate Gammak ======================
    binv8=inv(Nk);

    for (i=1:8)

        for (j=1:8)

            InvNk(i,j)=binv8(i,j);
        end
    end
```

```
%============================ Finish Gammak =========
for (i=1:7)

    for (j=1:8)

        GAMMAk(i,j)=0;
        for (k=1:8)

            GAMMAk(i,j)=GAMMAk(i,j)+Mk(i,k)*InvNk(k,j);
        end
    end
end
%======================= Calculate Phik ===================
%===================== Calculate UpsBT ===========
for (i=1:7)

    for (j=1:7)

        UpsBT(i,j)=0;
        for (k=1:7)

            UpsBT(i,j)=UpsBT(i,j)+Ups(i,k)*B1(j,k);
        end
    end
end
%============ Calculate 1-gamma*BUpsBT =========
for (i=1:7)

    for (j=1:7)
        BUpsBT(i,j)=0;
        for (k=1:7)
            BUpsBT(i,j)=BUpsBT(i,j)+B1(i,k)*UpsBT(k,j);
        end
        if (i==j)
            RInv(i,j)=1-BUpsBT(i,j)*gam*gam;
        else
            RInv(i,j)=-BUpsBT(i,j)*gam*gam;
        end
    end
end
%=================== Calculate Inverse ==========
binv7=inv(RInv);
for (i=1:7)
    for (j=1:7)
        InvR(i,j)=binv7(i,j);
    end
end
%============= Calculate inv(1-gam*B1UpsB1^T)*B1 ===========
for (i=1:7)
    for (j=1:7)
        Prod1(i,j)=0;
        for (k=1:7)
            Prod1(i,j)=Prod1(i,j)+InvR(i,k)*B1(k,j);
            end
        end
    end
    %========== Calculate Ups*B1^T*inv(1-gam*B1UpsB1^T)*B1 ===========
    for (i=1:7)
        for (j=1:7)
            Prod2(i,j)=0;
            for (k=1:7)
                Prod2(i,j)=Prod2(i,j)+UpsBT(k,i)*Prod1(k,j);
            end
        end
    end
    for (i=1:7)
        for (j=1:7)
            Prod3(i,j)=0;
            if (i==j)
                Prod3(i,j)=1+gam*gam*Prod2(i,j); %Multiply with Gama not
division.
            else
                Prod3(i,j)=gam*gam*Prod2(i,j);
            end
        end
    end
    for (i=1:7)
        for (j=1:7)
            GkH(i,j)=0;
            for (k=1:8)
                GkH(i,j)=GkH(i,j)+GAMMAk(i,k)*Hk(k,j);
            end
            Prod4(i,j)=Fk(i,j)-GkH(i,j);
        end
    end
    for (i=1:7)
        for (j=1:7)
            PHIk(i,j)=0;
            for (k=1:7)
                PHIk(i,j)=PHIk(i,j)+Prod4(i,k)*Prod3(k,j);
            end
        end
    end
    %===== The Robust Linear Quadratic Regulator ========
    %
    % Now Determine the Controller:
    %========== THxx =========================================
    %========================================
    %========================== Calculate AxW =================
    for (i=1:7)
        for (j=1:7)
            AxW(i,j)=0;
            for (k=1:7)
                AxW(i,j)=AxW(i,j)+Ax(i,k)*W(k,j);
            end
        end
    end
```

```
%============ Calculate AxWB1 ===================
for (i=1:7)
    for (j=1:7)
        AxWB1(i,j)=0;
        for (k=1:7)
            AxWB1(i,j)=AxWB1(i,j)+AxW(i,k)*B1(k,j);
        end
    end
end
%====================== Calculate IFA ==========
for (i=1:7)
    for (j=1:7)
        if (i==j)
            IFA(i,j)=1-Fk(i,j)-AxWB1(i,j);
        else
            IFA(i,j)=-Fk(i,j)-AxWB1(i,j);
        end
    end
end

%=================== Calculate E1IFA =================
for (i=1:3)
    for (j=1:7)
        E1IFA(i,j)=0;
        for (k=1:7)
            E1IFA(i,j)=E1IFA(i,j)+E1(i,k)*IFA(k,j);
        end
    end
end
%================ Calculate EEIFA ==================
for (i=1:3)
    for (j=1:7)
        EEIFA(i,j)=E2(i,j)+E1IFA(i,j);
    end
end
%============================= Calculate Thxx ===================
%Xweights=[0.001;0.002;0.005;0.01;0.01;0.01;0.02];
for (i=1:7)
    for (j=1:7)
        Thxx(i,j)=0;
        for (k=1:3)
            Thxx(i,j)=Thxx(i,j)+EEIFA(k,i)*EEIFA(k,j);
        end
        if (i==j)
            Thxx(i,j)=Thxx(i,j)+1;
        end
    end
end
%================= Calculate Thuu ====================
%====================
%========================= Calculate AxWB2 ==================
for (i=1:7)
    for (j=1:2)
        AxWB2(i,j)=0;
        for (k=1:7)
            AxWB2(i,j)=AxWB2(i,j)+AxW(i,k)*B2(k,j);
        end
    end
end

%============== Calculate GAW ==================
for (i=1:7)
    for (j=1:2)
        GAW(i,j)=Gk(i,j)+AxWB2(i,j);
    end
end
%=============== Calculate E1GAW ========================
for (i=1:3)
    for (j=1:2)
        E1GAW(i,j)=0;
        for (k=1:7)
            E1GAW(i,j)=E1GAW(i,j)+E1(i,k)*GAW(k,j);
        end
        E1GAW(i,j)=-E1GAW(i,j);
    end
end
%============= Calculate Thuu ====================
for (i=1:2)
    for (j=1:2)
        Thuu(i,j)=0;
        for (k=1:3)
            Thuu(i,j)=Thuu(i,j)+E1GAW(k,i)*E1GAW(k,j);
        end
        if i==j
            Thuu(i,j)=Thuu(i,j)+1;
        end

    end
end
%================ Calculate Thux ============================
for (i=1:7)
    for (j=1:2)
        Thux(i,j)=0;
        for (k=1:3)
            Thux(i,j)=Thux(i,j)+EEIFA(k,i)*E1GAW(k,j);
        end
    end
end
%
%Solve The Ricatti Equation
%================== Initial Sk =====================
for (i=1:7),
    for (j=1:7),
        if (i==j),
            SkNew(i,j)=10;%1E-6;
        else
            SkNew(i,j)=0;
        end;
```

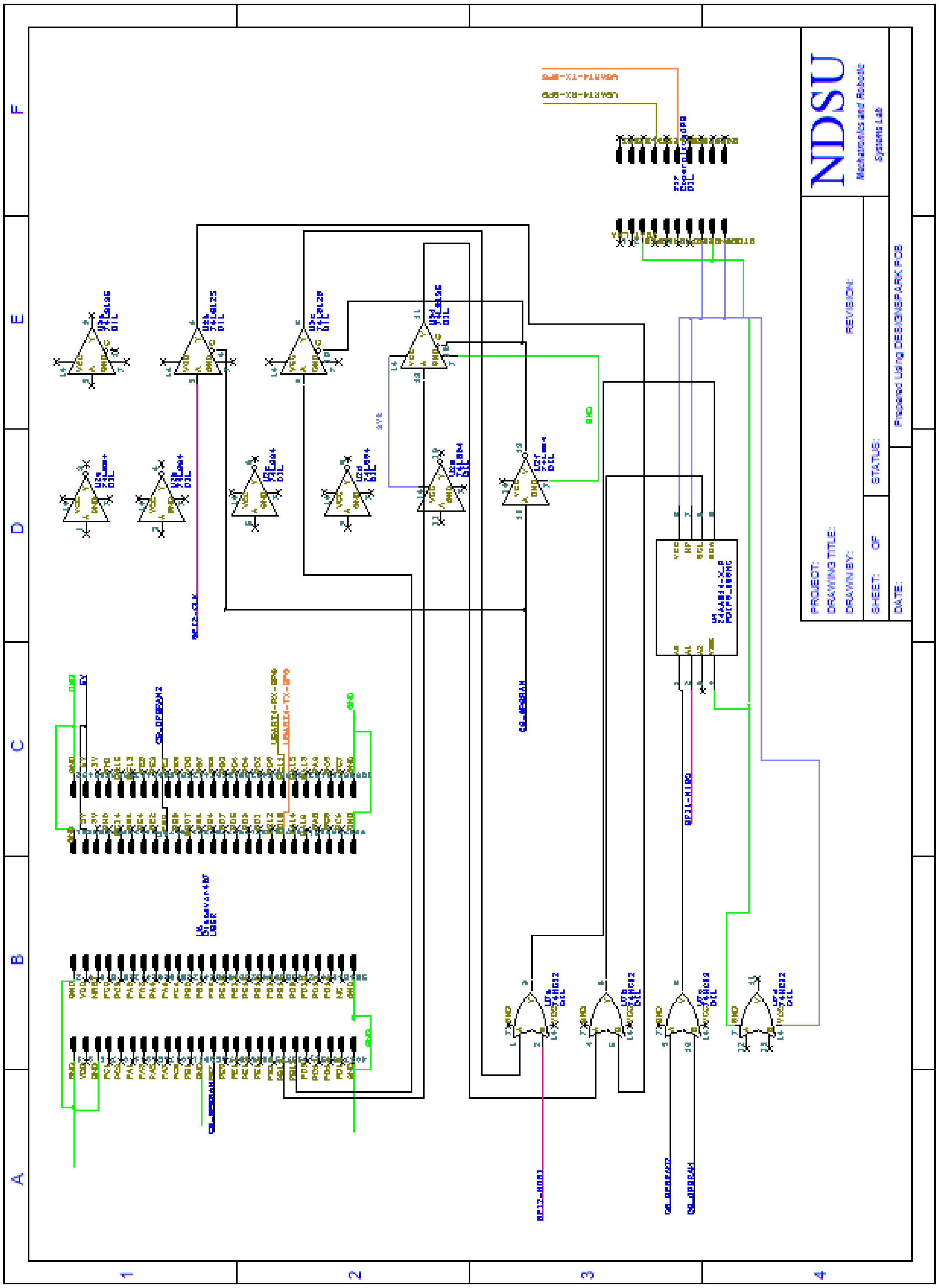
```
     end;
 end;
 Converg=100; %%%%%%Convregence value
 while(Converg>10) %%%%%%Run in a loop
     for i=1:7
         for j=1:7
             Sk(i,j)=SkNew(i,j);
         end
     end
 %================= Calculate FTS ==============
 for (i=1:7)
     for (j=1:7)
         FTS(i,j)=0;
         for (k=1:7)
             FTS(i,j)=FTS(i,j)+Fk(k,i)*Sk(k,j);
         end
     end
 end
%==================== Calculate FTSF ======================
for (i=1:7)
     for (j=1:7)
         FTSF(i,j)=0;
         for (k=1:7)
             FTSF(i,j)=FTSF(i,j)+FTS(i,k)*Fk(k,j);
         end
     end
 end

 %================= Calculate FTSG ============================
 for (i=1:7)

     for (j=1:2)

         FTSG(i,j)=0;
         for (k=1:7)

             FTSG(i,j)=FTSG(i,j)+FTS(i,k)*Gk(k,j);
         end
     end
 end
 %==================== Calculate GTS =======================
 for (i=1:2)
     for (j=1:7)
         GTS(i,j)=0;
         for (k=1:7)
             GTS(i,j)=GTS(i,j)+Gk(k,i)*Sk(k,j);
         end
     end
 end
 %============== Calculate GTSG ====================
 for (i=1:2)
     for (j=1:2)
         GTSG(i,j)=0;
         for (k=1:7)
```

```
            GTSG(i,j)=GTSG(i,j)+GTS(i,k)*Gk(k,j);
        end
    end
end
%================== Calculate FGTHxx ============
for (i=1:7)
    for (j=1:7)
        FGTHxx(i,j)=FTSF(i,j)+Thxx(i,j);  %+
    end
end
    %========================= Calculate GGTHuu ========
for (i=1:2)
    for (j=1:2)
        GGTHuu(i,j)=GTSG(i,j)+Thuu(i,j);
    end
end
%============= Calculate FGTHxu =========
for (i=1:7)
    for (j=1:2)
        FGTHxu(i,j)=FTSG(i,j)+Thux(i,j);
    end
end

%======== Calculate invGSG ================
binv2=inv(GGTHuu);
for (i=1:2)
    for (j=1:2)
        invGSG(i,j)=binv2(i,j);
    end
end
%================= Calculate FGTHGSG ==========================
for (i=1:7)
    for (j=1:2)
        FGTHGSG(i,j)=0;
        for(k=1:2)
            FGTHGSG(i,j)=FGTHGSG(i,j)+FGTHxu(i,k)*invGSG(k,j);
        end
    end
end
%============ Calculate TH_G_TH =======================
for (i=1:7)
    for (j=1:7)
        TH_G_TH(i,j)=0;
        for (k=1:2)
            TH_G_TH(i,j)=TH_G_TH(i,j)+FGTHGSG(i,k)*FGTHxu(j,k);
        end
    end
end
%========================= Calculate Snew ==============
for (i=1:7)
    for (j=1:7)
        SkNew(i,j)=FGTHxx(i,j)-TH_G_TH(i,j);
    end
end
```

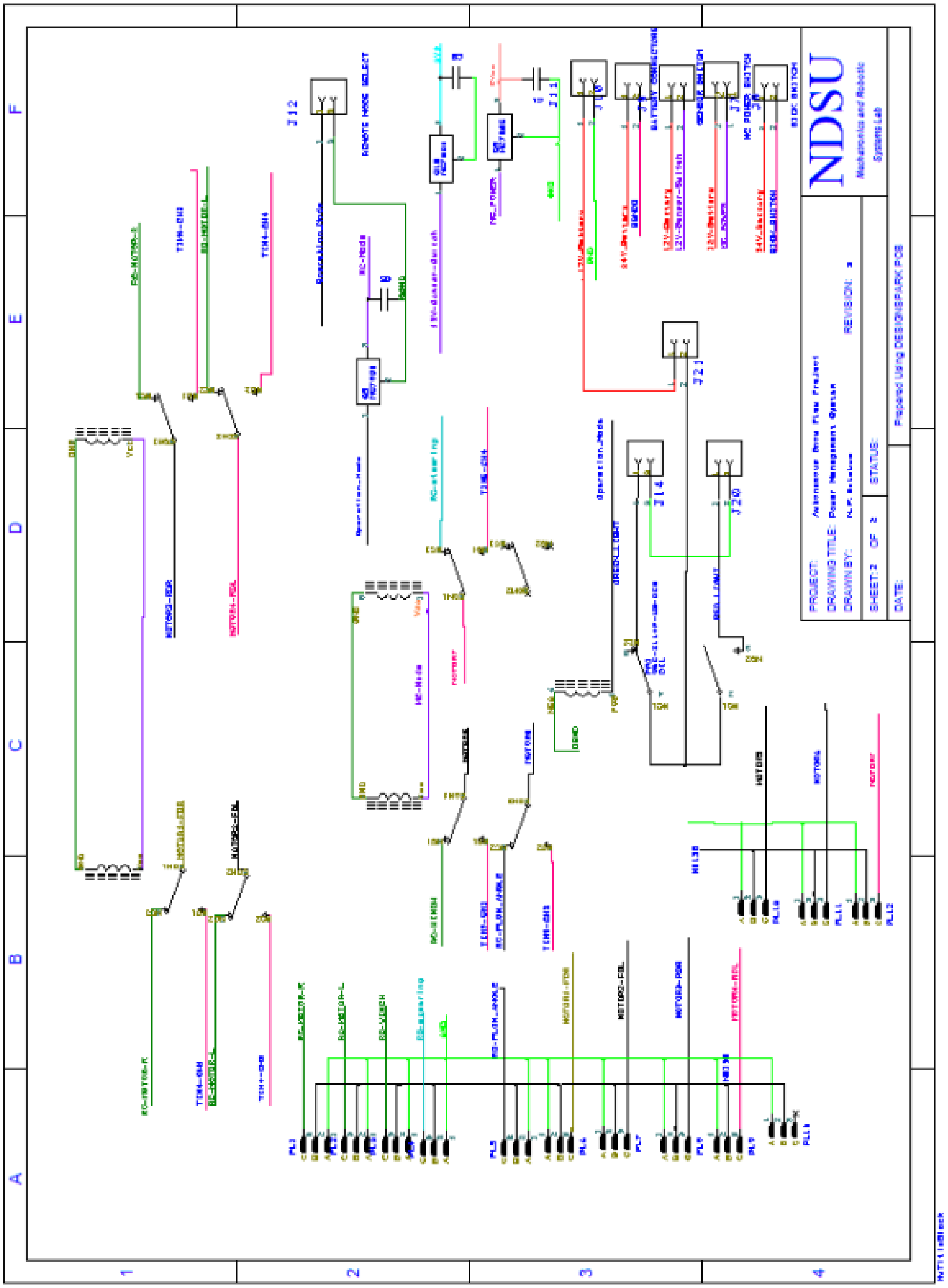
```
 %%%======= Check if Ups has converged =========================
    Converg=0;
    for i=1:7
        for j=1:7
            CVG(i,j)=SkNew(i,j)-Sk(i,j);
            Converg=Converg+(SkNew(i,j)-Sk(i,j))*(SkNew(i,j)-Sk(i,j));
        end
    end
    Converg=sqrt(Converg);
end

SkNew;

%=============== Calculate Lam ============
%==================================================
%============= Calculate New GTS ========================
for (i=1:2)

    for (j=1:7)

        GTS(i,j)=0;
        for (k=1:7)
            GTS(i,j)=GTS(i,j)+Gk(k,i)*Sk(k,j);
        end
    end
end
%================ Calculate New GTSG ===============================
for (i=1:2)

    for (j=1:2)

        GTSG(i,j)=0;
        for (k=1:7)
           GTSG(i,j)=GTSG(i,j)+GTS(i,k)*Gk(k,j);
        end
    end
end
%=========== Calculate New GGTHuu =======
for (i=1:2)
    for (j=1:2)
        GGTHuu(i,j)=GTSG(i,j)+Thuu(i,j);
    end
end
%============= Calculate New invGSG =============
binv2=inv(GGTHuu);
for (i=1:2)
    for (j=1:2)
        invGSG(i,j)=binv2(i,j);
    end
end
%========= Calculate New FTS ================
for (i=1:7)
    for (j=1:7)
        FTS(i,j)=0;
        for (k=1:7)
            FTS(i,j)=FTS(i,j)+Fk(k,i)*Sk(k,j);
        end
    end
end

%============= Calculate New FTSG ============
for (i=1:7)
    for (j=1:2)
        FTSG(i,j)=0;
        for (k=1:7)
            FTSG(i,j)=FTSG(i,j)+FTS(i,k)*Gk(k,j);
        end
    end
end
% ======================= Calculate New FGTHxu ======================
for (i=1:7)
    for (j=1:2)
        FGTHxu(i,j)=FTSG(i,j)-Thux(i,j);
    end
end
%=================== Calculate Lamk =========
for (i=1:2)
    for (j=1:7)
        LAMk(i,j)=0;
        for (k=1:2)
            LAMk(i,j)=LAMk(i,j)-GGTHuu(i,k)*FGTHxu(j,k);
        end
    end
end
%Y=measure data

xi=xold;
Yx=Hk*xi;
xi=PHIk*xi+GAMMAk*Yx;
U=-LAMk*xi;
if(U(1)>Umax)
    U(1)=Umax;
elseif (U(1)<-Umax)
    U(1)=-Umax;
end
if(U(2)>Umax)
    U(2)=Umax;
elseif (U(2)<-Umax)
    U(2)=-Umax;
end
Cont=[Cont,U];
xnew=(Fk*xi+Gk*U);
dx=xnew(6)-xi(6);
dy=xnew(7)-xi(7);
if (((dx>0) &&(DX<0))||((dx<0) &&(DX>0)))
    disp('X Error')
end
X=xnew(6);
```

```
    Y=xnew(7);
    Pid=Pid+1;
    xold=xnew;
end
    Traj=[Traj;X,Y];
    figure(1)
    plot(Traj(:,1),Traj(:,2),'-r+',path(2,:),path(3,:),'-.bX');
    legend('Simulated Trajectory','Intended Path')
    grid
    xlabel('Easting [m]')
    ylabel('Northing [m]')
    TTx=interp(Traj(:,1),100);
    TTy=interp(Traj(:,2),100);
    PPx=interp(path(2,:),100);
    PPx=PPx';
    PPy=interp(path(3,:),100);
    %for sinusoid only
    PPy=PPy';
    TTX=TTx;
    TTY=TTy;
    PPX=PPx;
    PPY=PPy;
    TTx=0;
    PPx=0;
    TTy=0;
    PPy=0;
    TTx=TTX(1:3510);
    TTy=TTY(1:3510);
    PPx=PPX(1:3510);
    PPy=PPY(1:3510);
    LP=0;
    for k=2:length(PPx)
        LP=LP+sqrt((PPx(k)-PPx(k-1))^2+(PPy(k)-PPy(k-1))^2);
    end;
    ER=sqrt((PPx-TTx).^2+(PPy-TTy).^2);
  mean(ER)
  figure(2)
   plot(TTx,TTy,'-r',PPx,PPy,'-.b');
    legend('Simulated Trajectory','Intended Path')
    grid
    xlabel('Easting [m]')
    ylabel('Northing [m]')
    %Cont=[Cont(:,1:38),Cont(:,38),Cont(:,38)];
    %plot(Traj(:,1),Traj(:,2),'-.r+',path(2,:),path(3,:),'-
bX',Traj(:,1),[State(5,1:NPP-1),State(5,NPP-1)],'-
kd',Traj(:,1),Cont(1,:),'-g+',Traj(:,1),Cont(2,:),'-c+');
```

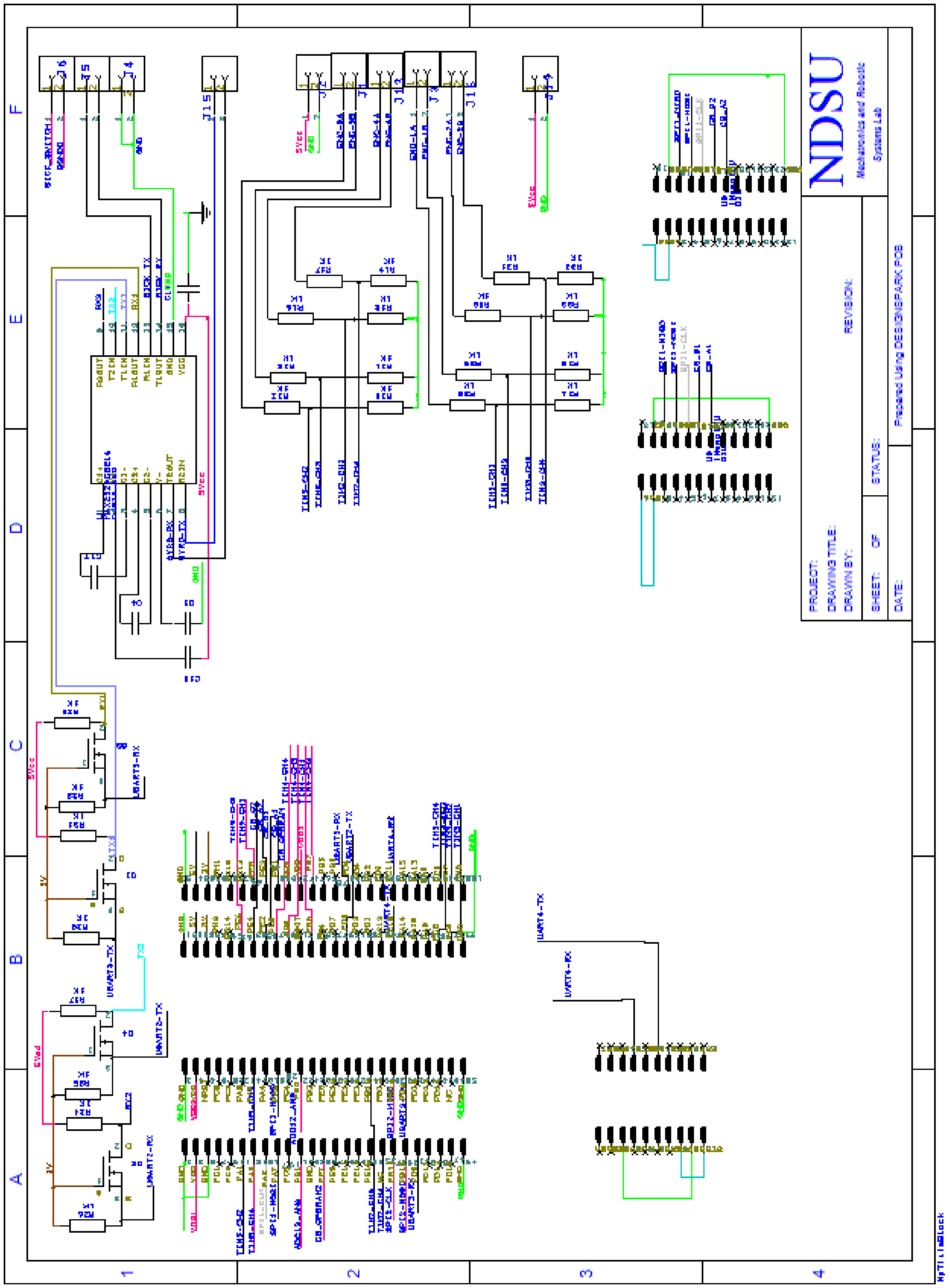
## APPENDIX C. CIRCUIT BOARD DESIGN



C1: GPS Layer



C2: Power Layer


C3: Sensor Layer

# APPENDIX D. CONTROL AND FILTERING CODE

```
/**
  ******************
  * @attention NDSU - Autonomous Snow Plow Control Firmware, V1.0
  * @author  MF Selekwa
  * @version V1.0.0
  * @date    2-January-2014
  * @brief   Main program body
**/
/* Includes ----------------------------------------------*/
#include "thundarconfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "timers.h"
#include "queue.h"
#include <math.h>

typedef struct inv8
{
    double binv[8][8];
} inv8;


// Way point format: HOME POSITION, BP1,BP2, BP3,BP4, HOME POSITION
struct Robot
{
uint8_t WP;   //Way point Index
uint16_t VR;
uint16_t VL;
};

float probe[50];
struct SensorData
{
float AX1;  // IMU data
float GZ1; //End of IMU data
float AX2;  // IMU data
float GZ2; //End of IMU data
uint32_t NIMU;
float X;    //GPS coordinates data
float Y;  //end of GPS coordinates data

float X0;    //Initial GPS X-coordinate
float Y0;  //Initial GPS Y-coordinate
uint LEnc;  //Cummulative Left Encoder data
uint REnc;  //Cummulative Right Encoder data
int LEInc;  //Incremental Left Encoder data
int REInc;  //Incremental Right Encoder data
};


double PHIk[7][7]={{0}};
double GAMMAk[7][8]={{0}};
double Ups[7][7]={{0}};
double UpsNew[7][7]={{0}};

double invmatrix88[8][8]={{0}};

double LAMk[2][7]={{0}};
double Sk[7][7]={{0}};
double SkNew[7][7]={{0}};

uint8_t Temp2[180];

uint16_t MaxNegSpeed = 94;
uint16_t ZeroSpeed  = 86;
uint16_t MaxPosSpeed = 78;
uint8_t WayPoint=0;

uint64_t u64IdleTicksCnt=0; // Counts when the OS has no task to execute.
uint64_t tickTime=0;        // Counts OS ticks (default = 1000Hz).

struct SensorData GEDISensor={0};
struct Robot Controls={0};

double path[3][11];
double Xr; double Yr;
uint32_t Pid;
void GetInitCoordinates(void);
void GPS(void *pvparameters);
void ENCODERS(void *pvparameters);
void IMU(void *pvparameters);
void DriveRobot(void *pvparameters);
void MissionController1(void *pvparameters);

/*=============================================================================
            Inverse Functions
*===========================================================================*/
inv8 invmatrix8(double a[8][8],uint8_t sz)
{
      uint8_t i;uint8_t j;uint8_t k;double s;uint8_t L;
      inv8 invv8;
      double binv[8][8];
      double t;
      for (i=0;i<sz;i++)
      {
            for (j=0;j<sz;j++)
            {
                  if (i==j)
                  {
                        binv[i][j]=1;
                  }
                  else
                  {
                        binv[i][j]=0;
                  }
            }
      }
```

```
	for (j=0;j<sz;j++)
	{
		for (i=j;i<sz;i++)
		{
			if (a[i][j]!=0)
			{
				for (k=0;k<sz;k++)
				{
					s=a[j][k];
					a[j][k]=a[i][k];
					a[i][k]=s;
					s=binv[j][k];
					binv[j][k]=binv[i][k];
					binv[i][k]=s;
				}
				t=1/a[j][j];
				for (k=0;k<sz;k++)
				{
					a[j][k]=t*a[j][k];
					binv[j][k]=t*binv[j][k];
				}
				for (L=0;L<sz;L++)
				{
					if(L!=j)
					{
						t=-a[L][j];
						for (k=0;k<sz;k++)
						{
							a[L][k]=a[L][k]+t*a[j][k];
							binv[L][k]=binv[L][k]+t*binv[j][k];
						}
					}
				}
			}
		}
	}
	for (i=0;i<sz;i++)
	{
		for (j=0;j<sz;j++)
		{
			invv8.binv[i][j]=binv[i][j];
		}
	}
	return(invv8);
}
/*=============================================================================
	Filter Functions
*============================================================================*/

void KalmanFilter(double Ax[7][7],double Ay[8][7],double B1[7][7], double Hk
[8][7], double Fk [7][7],double P[7][7],double Q[8][8],double gam)
{
	uint8_t i;uint8_t j;uint8_t k;
	uint32_t w;
	inv8 binv8;

	//Referenced intermediate terms
```

```
double Xi[7][7];
double Nk[8][8];
double Mk[7][8];
//Intermediate Terms
//Xi
double InvUps[7][7];
double invXi[7][7];

//Nk
double AYAT[8][8];

//Mk
double AxAT[7][8];
double XiH[7][8];
double FXiH[7][8];

//Gamma
double HXiH[8][8];
double InvNk[8][8];
double GkH[7][7];

//Ups
double XiFT[7][7];
double FXiFT[7][7];
double InvNkMkT[8][7];
double MkInvNkMkT[7][7];
double AxAxT[7][7];

//Phi
double UpsBT[7][7];
double BUpsBT[7][7];
double RInv[7][7];
double InvR[7][7];
double Prod1[7][7];
double Prod2[7][7];
double Prod3[7][7];
double Prod4[7][7];

//Convergence Variables
double Converg;
uint16_t IT;
//=====================================================================
      Calculate Phik and Gammak
//=====================================================================
//======================== Calculate inverse Ups =====================

Converg=10E8;
IT=0;
//while(Converg>1E5)
while(IT<75)
{
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            Ups[i][j]=UpsNew[i][j];
      }
```

```
}
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            invmatrix88[i][j]=Ups[i][j];
            if(i==j)
            {
                  probe[7+i]=Ups[i][j];
            }
      }
}
binv8=invmatrix8(invmatrix88,7);
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            InvUps[i][j]=binv8.binv[i][j];
      }
}
//================= Calculate invXi ===========================
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            invXi[i][j]=0;
            for (k=0;k<7;k++)
            {
                  invXi[i][j]=invXi[i][j]+B1[k][i]*B1[k][j];
            }
            invXi[i][j]=InvUps[i][j]-invXi[i][j]*(gam*gam);
      }
}
//================= Calculate Xi ========
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            invmatrix88[i][j]=invXi[i][j];
      }
}
binv8=invmatrix8(invmatrix88,7);
for (i=0;i<7;i++)
{
      for (j=0;j<7;j++)
      {
            Xi[i][j]=binv8.binv[i][j];
      }
}
//=============== Calculate Nk ================================
for (i=0;i<7;i++)
{
      for (j=0;j<8;j++)
      {
            XiH[i][j]=0;
            for (k=0;k<7;k++)
            {
```

```
                        XiH[i][j]=(XiH[i][j]+Xi[i][k]*Hk[j][k]);
                }
        }
}
for (i=0;i<8;i++)
{
        for(j=0;j<8;j++)
        {
                 HXiH[i][j]=0;
                for (k=0;k<7;k++)
                {
                         HXiH[i][j]=(HXiH[i][j]+Hk[i][k]*XiH[k][j]);
                }
        }
}
//================= Calculate InvNkHXiH ============
for (i=0;i<8;i++)
{
        for (j=0;j<8;j++)
        {
                AYAT[i][j]=0;
                for (k=0;k<7;k++)
                {
                        AYAT[i][j]=AYAT[i][j]+Ay[i][k]*Ay[j][k];
                }
                Nk[i][j]=(Q[i][j]+AYAT[i][j]*(gam*gam)+HXiH[i][j]);
        }
}

//================= Calculate Mk ====================
for (i=0;i<7;i++)
{
        for (j=0;j<8;j++)
        {
                FXiH[i][j]=0;
                for (k=0;k<7;k++)
                {
                        FXiH[i][j]=(FXiH[i][j]+Fk[i][k]*XiH[k][j]);
                }
        }
}
for (i=0;i<7;i++)
{
        for (j=0;j<8;j++)
        {
                AxAT[i][j]=0;
                for (k=0;k<7;k++)
                {
                        AxAT[i][j]=AxAT[i][j]+Ax[i][k]*Ay[j][k];
                }
                Mk[i][j]=(FXiH[i][j]+AxAT[i][j]*(gam*gam));
        }
}
//=================================================================
        Update covariances and return Ups
//=================================================================
    for (i=0;i<7;i++)
```

```
        {
            for (j=0;j<7;j++)
            {
                 AxAxT[i][j]=0;
                for (k=0;k<7;k++)
                {
                       AxAxT[i][j]= AxAxT[i][j]+Ax[i][k]*Ax[j][k];
                }
            }
        }
        //%compute inverse of Nk
          for (i=0;i<8;i++)
          {
                for (j=0;j<8;j++)
                {
                      invmatrix88[i][j]=Nk[i][j];
                }
          }
          binv8=invmatrix8(invmatrix88,8);
        for (i=0;i<8;i++)
        {
            for (j=0;j<8;j++)
            {
                InvNk[i][j]=binv8.binv[i][j];
            }
        }
       // %compute InvNkMkT
        for (i=0;i<8;i++)
        {
            for (j=0;j<7;j++)
            {
                InvNkMkT[i][j]=0;
                for (k=0;k<8;k++)
                {
                      InvNkMkT[i][j]=(InvNkMkT[i][j]+InvNk[i][k]*Mk[j][k]);
                }
            }
        }
       // %Now compute MkInvNkMkT
        for (i=0;i<7;i++)
        {
            for (j=0;j<7;j++)
            {
                MkInvNkMkT[i][j]=0;
                for (k=0;k<8;k++)
                {

MkInvNkMkT[i][j]=(MkInvNkMkT[i][j]+Mk[i][k]*InvNkMkT[k][j]);
                }
            }
        }
        //%compute XiFT
        for (i=0;i<7;i++)
        {
            for (j=0;j<7;j++)
            {
                 XiFT[i][j]=0;
                for (k=0;k<7;k++)
                {
                       XiFT[i][j]=(XiFT[i][j]+Xi[i][k]*Fk[j][k]);
                }
            }
        }

        //%Now compute FXiFT
        for (i=0;i<7;i++)
        {
            for (j=0;j<7;j++)
            {
                FXiFT[i][j]=0;
                for (k=0;k<7;k++)
                {
                      FXiFT[i][j]=(FXiFT[i][j]+Fk[i][k]*XiFT[k][j]);
                }
            }
        }
          //======== Calculate Ups =============
        for (i=0;i<7;i++)
        {
            for (j=0;j<7;j++)
            {
                UpsNew[i][j]=0.5*(FXiFT[i][j]-
MkInvNkMkT[i][j])+AxAxT[i][j]*(gam*gam)+P[i][j];
            }
        }

      //===== Check if Ups has converged ==========================
      Converg=0;
          for (i=0;i<7;i++)
          {
                for (j=0;j<7;j++)
                {
              Converg=Converg+(UpsNew[i][j]-Ups[i][j])*(UpsNew[i][j]-
Ups[i][j]);
                }
          }
       Converg=sqrt(Converg)/49;
       probe[49]=Converg;
     IT=IT+1;
     probe[48]= IT;
    }
    //=============== Calculate Gammak ================
    for (i=0;i<7;i++)
    {
        for (j=0;j<7;j++)
        {
                Ups[i][j]=UpsNew[i][j];//1E10;
        }
    }
    for (i=0;i<8;i++)
    {
          for (j=0;j<8;j++)
          {
                invmatrix88[i][j]=Nk[i][j];
        }
}
binv8=invmatrix8(invmatrix88,8);
for (i=0;i<8;i++)
{
        for (j=0;j<8;j++)
        {
                InvNk[i][j]=binv8.binv[i][j];
        }
}
//========== Finish Gammak ==========================
for (i=0;i<7;i++)
{
        for (j=0;j<8;j++)
        {
                GAMMAk[i][j]=0;
                for (k=0;k<8;k++)
                {
                        GAMMAk[i][j]=GAMMAk[i][j]-Mk[i][k]*InvNk[k][j];
                }
        }
}
//==================== Calculate Phik ==================
//==================== Calculate UpsBT =================
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                UpsBT[i][j]=0;
                for (k=0;k<7;k++)
                {
                        UpsBT[i][j]=UpsBT[i][j]+Ups[i][k]*B1[j][k];
                }
        }
}
//======Calculate BUpsBT ==============================
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                BUpsBT[i][j]=0;
                for (k=0;k<7;k++)
                {
                        BUpsBT[i][j]=BUpsBT[i][j]+B1[i][k]*UpsBT[k][j];
                }
                if (i==j)
                {
                        RInv[i][j]=1-BUpsBT[i][j]*(gam*gam);
                }
                else
                {
                        RInv[i][j]=-BUpsBT[i][j]*(gam*gam);
                }
        }
}
//=============== Calculate Inverse =============
for (i=0;i<7;i++)
```

```
{
        for (j=0;j<7;j++)
        {
                invmatrix88[i][j]=RInv[i][j];
        }
}
binv8=invmatrix8(invmatrix88,7);
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                InvR[i][j]=binv8.binv[i][j];
        }
}
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                Prod1[i][j]=0;
                for (k=0;k<7;k++)
                {
                        Prod1[i][j]=Prod1[i][j]+InvR[i][k]*B1[k][j];
                }
        }
}
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                Prod2[i][j]=0;
                for (k=0;k<7;k++)
                {

Prod2[i][j]=Prod2[i][j]+UpsBT[i][k]*Prod1[k][j]*(gam*gam);
                }
        }
}
for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
                Prod3[i][j]=0;
                if (i==j)
                {
                        Prod3[i][j]=1+Prod2[i][j];
                }
                else
                {
                        Prod3[i][j]=Prod2[i][j];
                }
        }
}

for (i=0;i<7;i++)
{
        for (j=0;j<7;j++)
        {
```

```
                  GkH[i][j]=0;
                  for (k=0;k<8;k++)
                  {
                        GkH[i][j]=GkH[i][j]-GAMMAk[i][k]*Hk[k][j];
                  }
                  Prod4[i][j]=Fk[i][j]-GkH[i][j];
            }
      }
      for (i=0;i<7;i++)
      {
            for (j=0;j<7;j++)
            {
                  PHIk[i][j]=0;
                  for (k=0;k<7;k++)
                  {
                        PHIk[i][j]=PHIk[i][j]+Prod4[i][k]*Prod3[k][j];
                  }
            }
      }
}

/*============================================================================
      Control Function
============================================================================
 */

void H2Control(double Ax[7][7],double B1[7][7],double B2[7][2],double W
[7][7],double Fk [7][7],double Gk [7][2],double E1[3][7],double E2[3][7])
{
      uint8_t i;uint8_t j;uint8_t k;
      inv8 binv8;
      //Control Matrices
      double Thuu[2][2]={{0}};
      double Thxx[7][7]={{0}};
      double Thux[7][2]={{0}};

      double AxW[7][7]={{0}};
      double AxWB1[7][7]={{0}};
      double IFA[7][7]={{0}};
      double E1IFA[3][7]={{0}};
      double EEIFA[3][7]={{0}};

      double AxWB2[7][2]={{0}};
      double GAW[7][2]={{0}};
      double E1GAW[3][2]={{0}};

      double FTS[7][7]={{0}};
      double FTSF[7][7]={{0}};
      double FGTHxx[7][7]={{0}};
      double FTSG[7][2]={{0}};
      double FGTHxu[7][2]={{0}};
      double GTS[2][7]={{0}};
      double GTSG[2][2]={{0}};
      double GGTHuu[2][2]={{0}};
      double invGSG[2][2]={{0}};
      double FGTHGSG[7][2]={{0}};
      double TH_G_TH[7][7]={{0}};
```

```
//Convergence Variables
double Converg;
double CVG[7][7];
uint16_t IT;

      //======================== THxx ========
      //============================================
      //============== Calculate AxW ===============
    for (i=0;i<7;i++)
    {
        for (j=0;j<7;j++)
        {
            AxW[i][j]=0;
            for (k=0;k<7;k++)
            {
                  AxW[i][j]=AxW[i][j]+Ax[i][k]*W[k][j];
            }
        }
    }
      //=========== Calculate AxWB1 =====
    for (i=0;i<7;i++)
    {
        for (j=0;j<7;j++)
        {
            AxWB1[i][j]=0;
            for (k=0;k<7;k++)
            {
                  AxWB1[i][j]=AxWB1[i][j]+AxW[i][k]*B1[k][j];
            }
        }
    }
      //============ Calculate IFA ==========
    for (i=0;i<7;i++)
    {
        for (j=0;j<7;j++)
        {
            if (i==j)
            {
                  IFA[i][j]=1-Fk[i][j]-AxWB1[i][j];
            }
            else
            {
                  IFA[i][j]=-Fk[i][j]-AxWB1[i][j];
            }
        }
    }
      //============= Calculate E1IFA =========
    for (i=0;i<3;i++)
    {
        for (j=0;j<7;j++)
        {
            E1IFA[i][j]=0;
            for (k=0;k<7;k++)
            {
                  E1IFA[i][j]=E1IFA[i][j]+E1[i][k]*IFA[k][j];
            }
        }
    }
      //========= Calculate EEIFA ====================================
    for (i=0;i<3;i++)
    {
        for (j=0;j<7;j++)
        {
            EEIFA[i][j]=E2[i][j]+E1IFA[i][j];
        }
    }
    //===== Calculate Thxx ===================================
    for (i=0;i<7;i++)
    {
        for (j=0;j<7;j++)
        {
            Thxx[i][j]=0;
            for (k=0;k<3;k++)
            {
                Thxx[i][j]=Thxx[i][j]+EEIFA[k][i]*EEIFA[k][j];
            }
            if (i==j)
            {
                  Thxx[i][j]=Thxx[i][j]+1;
            }
            Thxx[i][j]=Thxx[i][j]*0.1;
        }
    }
    //============ Calculate Thuu ========
    //========= Calculate AxWB2
    for (i=0;i<7;i++)
    {
        for (j=0;j<2;j++)
        {
            AxWB2[i][j]=0;
            for (k=0;k<7;k++)
            {
                  AxWB2[i][j]=AxWB2[i][j]+AxW[i][k]*B2[k][j];
            }
        }
    }

    //=============== Calculate GAW ================
    for (i=0;i<7;i++)
    {
        for (j=0;j<2;j++)
        {
            GAW[i][j]=Gk[i][j]+AxWB2[i][j];
        }
    }
    //============= Calculate E1GAW ==============
    for (i=0;i<3;i++)
    {
        for (j=0;j<2;j++)
        {
            E1GAW[i][j]=0;
            for (k=0;k<7;k++)
            {
                  E1GAW[i][j]=E1GAW[i][j]+E1[i][k]*GAW[k][j];
            }
            E1GAW[i][j]=-E1GAW[i][j];
        }
    }
    //=================== Calculate Thuu ===================
    for (i=0;i<2;i++)
    {
        for (j=0;j<2;j++)
        {
            Thuu[i][j]=0;
            for (k=0;k<3;k++)
            {
                  Thuu[i][j]=Thuu[i][j]+E1GAW[k][i]*E1GAW[k][j];
            }
            if (i==j)
            {
                  Thuu[i][j]=Thuu[i][j]+1;
            }
            Thuu[i][j]=Thuu[i][j]*0.1;
        }
    }
      //======================= Calculate Thux ================
      //================ Calculate Thux ====================
      for (i=0;i<7;i++)
      {
            for (j=0;j<2;j++)
            {
                  Thux[i][j]=0;
                  for (k=0;k<3;k++)
                  {
                        Thux[i][j]=Thux[i][j]+EEIFA[k][i]*E1GAW[k][j];
                  }
                   Thux[i][j]=Thux[i][j]*0.1;
            }
      }
      //================= Calculate Sk ===================
      //============ Calculate FTS ================================
      for (i=0;i<7;i++)
      {
            for (j=0;j<7;j++)
            {
            if (i==j)
            {
                SkNew[i][j]=10;
            }
            else
            {
                  SkNew[i][j]=0;
            }
            }
      }
      Converg=100;
      while(Converg>10)
      {
            for (i=0;i<7;i++)
            {
                  for (j=0;j<7;j++)
                  {
                      Sk[i][j]=SkNew[i][j];
                        if(i==j)
                        {
                              probe[14+i]=Sk[i][j];
                        }
              }
            }
      for (i=0;i<7;i++)
      {
            for (j=0;j<7;j++)
            {
                  FTS[i][j]=0;
                  for (k=0;k<7;k++)
                  {
                        FTS[i][j]=FTS[i][j]+Fk[k][i]*Sk[k][j];
                  }
            }
      }

      //=============== Calculate FTSF ===========================
      for (i=0;i<7;i++)
      {
            for (j=0;j<7;j++)
            {
                  FTSF[i][j]=0;
                  for (k=0;k<7;k++)
                  {
                        FTSF[i][j]=FTSF[i][j]+FTS[i][k]*Fk[k][j];
                  }
            }
      }

      //=========== Calculate FTSG ===================================
      for (i=0;i<7;i++)
      {
            for (j=0;j<2;j++)
            {
                  FTSG[i][j]=0;
                  for (k=0;k<7;k++)
                  {
                        FTSG[i][j]=FTSG[i][j]+FTS[i][k]*Gk[k][j];
                  }
            }
      }
      //================ Calculate GTS ===================
      for (i=0;i<2;i++)
      {
            for (j=0;j<7;j++)
            {
                  GTS[i][j]=0;
                  for (k=0;k<7;k++)
                  {
                        GTS[i][j]=GTS[i][j]+Gk[k][i]*Sk[k][j];
                  }
            }
```

```
  }
  //================ Calculate GTSG ==============
  for (i=0;i<2;i++)
  {
        for (j=0;j<2;j++)
        {
              GTSG[i][j]=0;
              for (k=0;k<7;k++)
              {
                    GTSG[i][j]=GTSG[i][j]+GTS[i][k]*Gk[k][j];
              }
        }
  }
  //================ Calculate FGTHxx ==================
for (i=0;i<7;i++)
{
    for (j=0;j<7;j++)
    {
        FGTHxx[i][j]=FTSF[i][j]+Thxx[i][j];
    }
}
//================== Calculate FGTHxu =============================
for (i=0;i<7;i++)
{
    for (j=0;j<2;j++)
    {
        FGTHxu[i][j]=FTSG[i][j]+Thux[i][j];
    }
}
//================ Calculate GGTHuu ==================
for (i=0;i<2;i++)
{
    for (j=0;j<2;j++)
    {
        GGTHuu[i][j]=GTSG[i][j]+Thuu[i][j];
    }
}
//=========== Calculate invGSG =======================
  for (i=0;i<2;i++)
  {
        for (j=0;j<2;j++)
        {
              invmatrix88[i][j]=GGTHuu[i][j];
        }
  }
binv8=invmatrix8(invmatrix88,2);
for (i=0;i<2;i++)
{
    for (j=0;j<2;j++)
    {
        invGSG[i][j]=binv8.binv[i][j];
    }
}
  //============= Calculate FGTHGSG =====================
  for (i=0;i<7;i++)
  {
        for (j=0;j<2;j++)
```

```
                  {
                        FGTHGSG[i][j]=0;
                        for (k=0;k<2;k++)
                        {

      FGTHGSG[i][j]=FGTHGSG[i][j]+FGTHxu[i][k]*invGSG[k][j];
                        }
                  }
            }
            //===================== Calculate TH_G_TH ====================
            for (i=0;i<7;i++)
            {
                  for (j=0;j<7;j++)
                  {
                        TH_G_TH[i][j]=0;
                        for (k=0;k<2;k++)
                        {

      TH_G_TH[i][j]=TH_G_TH[i][j]+FGTHGSG[i][k]*FGTHxu[j][k];
                        }
                  }
            }
          //============= Calculate Snew ==============================
          for (i=0;i<7;i++)
          {
              for (j=0;j<7;j++)
              {
                  SkNew[i][j]=FGTHxx[i][j]-TH_G_TH[i][j];
              }
          }
           //================ Check if Ups has converged ==================
        Converg=0;
            for (i=0;i<7;i++)
            {
                  for (j=0;j<7;j++)
                  {
                Converg=Converg+(SkNew[i][j]-Sk[i][j])*(SkNew[i][j]-
Sk[i][j]);
                  }
            }
        Converg=sqrt(Converg);
        probe[47]=Converg;
        IT=IT+1;
        probe[46]=IT;
      }
            //============== Calculate Lam =================
            //============== Calculate New GTS =============
            for (i=0;i<2;i++)
            {
                  for (j=0;j<7;j++)
                  {
                        GTS[i][j]=0;
                        for (k=0;k<7;k++)
                        {
                              GTS[i][j]=GTS[i][j]+Gk[k][i]*Sk[k][j];
                        }
                  }
```

```
  }
  //============= Calculate New GTSG ==========================
  for (i=0;i<2;i++)
  {
        for (j=0;j<2;j++)
        {
              GTSG[i][j]=0;
              for (k=0;k<7;k++)
              {
                    GTSG[i][j]=GTSG[i][j]+GTS[i][k]*Gk[k][j];
              }
        }
  }
//============ Calculate New GGTHuu ==========================
for (i=0;i<2;i++)
{
    for (j=0;j<2;j++)
    {
        GGTHuu[i][j]=GTSG[i][j]+Thuu[i][j];
    }
}
//================ Calculate New invGSG ====================
  for (i=0;i<2;i++)
  {
        for (j=0;j<2;j++)
        {
              invmatrix88[i][j]=GGTHuu[i][j];
        }
  }
binv8=invmatrix8(invmatrix88,2);
for (i=0;i<2;i++)
{
    for (j=0;j<2;j++)
    {
        invGSG[i][j]=binv8.binv[i][j];
    }
}
  //=============== Calculate New FTS ===========================
  for (i=0;i<7;i++)
  {
        for (j=0;j<7;j++)
        {
              FTS[i][j]=0;
              for (k=0;k<7;k++)
              {
                    FTS[i][j]=FTS[i][j]+Fk[k][i]*Sk[k][j];
              }
        }
  }

  //=============== Calculate New FTSG =========================
  for (i=0;i<7;i++)
  {
        for (j=0;j<2;j++)
        {
              FTSG[i][j]=0;
              for (k=0;k<7;k++)
```

```
                        {
                              FTSG[i][j]=FTSG[i][j]+FTS[i][k]*Gk[k][j];
                        }
                  }
            }
         // ============= Calculate New FGTHxu ======================
          for (i=0;i<7;i++)
          {
              for (j=0;j<2;j++)
              {
                  FGTHxu[i][j]=FTSG[i][j]+Thux[i][j];
              }
          }
          //=============== Calculate Lamk ==========
            for (i=0;i<2;i++)
            {
                  for (j=0;j<7;j++)
                  {
                        LAMk[i][j]=0;
                        for (k=0;k<2;k++)
                        {
                              LAMk[i][j]=LAMk[i][j]-
GGTHuu[i][k]*FGTHxu[j][k];
                        }
                  }
            }
}
/*=====================================================================
            Initial RTOS Code
===========================================================================*/
//static void prvSetupHardware( void );

/* The semaphore (in this case binary) that is used by the FreeRTOS tick hook
 * function and the event semaphore task.*/
xSemaphoreHandle xEventSemaphore = NULL;

/* The counters used by the various examples.  The usage is described in the
 * comments at the top of this file.*/
static volatile uint32_t ulCountOfReceivedSemaphores;

/*
 * When FreeRTOS crashes, you often end up in a hard fault.
 */

/*-------------------------------------------------------------*/
static void prvEventSemaphoreTask( void *pvParameters )
{
    while(1)
    {
        /* Block until the semaphore is 'given'. */
        xSemaphoreTake( xEventSemaphore, portMAX_DELAY );

        /* Count the number of times the semaphore is received. */
        ulCountOfReceivedSemaphores++;
        STM_EVAL_LEDToggle(LED5);
    }
}
```

```
/*-----------------------------------------------------------*/

void HardFault_Handler (void){
      STM_EVAL_LEDOn(LED5);
      STM_EVAL_LEDOn(LED6);
}

// This FreeRTOS callback function gets called once per tick (default =
1000Hz).
// ---------
void vApplicationTickHook( void )
{
      static signed portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
      static uint32_t ulCount = 0;
    ++tickTime;
    ulCount++;
    if(ulCount >= 250UL)
    {
        xSemaphoreGiveFromISR(xEventSemaphore, &xHigherPriorityTaskWoken );
        ulCount = 0UL;
    }
}

/*-----------------------------------------------------------*/

// This FreeRTOS call-back function gets when no other task is ready to
execute.
// On a completely unloaded system this is getting called at over 2.5MHz!
// ----------------------------------------------------------------------------
--
void vApplicationIdleHook( void ) {
    ++u64IdleTicksCnt;
}

// A required FreeRTOS function.
void vApplicationMallocFailedHook( void ) {
    configASSERT( 0 );  // Latch on any failure / error.
}

/*=============================================================================
                  Real Time Tasks
=============================================================================*/
void GetInitCoordinates(void)
{
      USART_TypeDef* chan;

      chan=UART4;
      union
        {
             uint32_t HexValue;
             float FloatValue;
         }TempData;
      uint8_t res;
      uint8_t i;
      uint8_t m;
      uint8_t n;
      ret_t Data;
```

```
	uint8_t GetGPSData[]={0x10,0x8E,0x17,0x10,0x03};
	res=0;



		Retry1:
		for(i=0;i<180;i++)
		{
			Data.Data[i]=0;
		}
		m=sizeof(GetGPSData)/sizeof(uint8_t);
			for(n=0;n<=m;n++)
			{
				myUSART_SendByte(chan,GetGPSData[n]);
			}
		Data=TrapByte(0x8F,0x17,chan);
		if(Data.Check==1 && res<3){goto Retry1;}
		res=0;

	TempData.HexValue=(Data.Data[6]<<24)+(Data.Data[7]<<16)+(Data.Data[8]<<
8)+(Data.Data[9]); //
		GEDISensor.Y0=TempData.FloatValue;
		probe[38]=GEDISensor.Y0;

	TempData.HexValue=(Data.Data[10]<<24)+(Data.Data[11]<<16)+(Data.Data[12
]<<8)+(Data.Data[13]);
		GEDISensor.X0=TempData.FloatValue;
		probe[39]=GEDISensor.X0;
		STM_EVAL_LEDToggle(LED3);

}


void GPS(void *pvparameters)
{
	USART_TypeDef* chan;

	chan=UART4;
	union
	  {
	       uint32_t HexValue;
	       float FloatValue;
	   }TempData;
	uint8_t res;
	uint8_t i;
	uint8_t m;
	uint8_t n;
	ret_t Data;
	uint8_t GetGPSData[]={0x10,0x8E,0x17,0x10,0x03};
	res=0;

	while(1)
	{

		Retry1:
		for(i=0;i<180;i++)
		{
```

```
                Data.Data[i]=0;
            }
            m=sizeof(GetGPSData)/sizeof(uint8_t);
                for(n=0;n<=m;n++)
                {
                    myUSART_SendByte(chan,GetGPSData[n]);
                }
            Data=TrapByte(0x8F,0x17,chan);
            if(Data.Check==1 && res<3){goto Retry1;}
            res=0;

        TempData.HexValue=(Data.Data[6]<<24)+(Data.Data[7]<<16)+(Data.Data[8]<<
8)+(Data.Data[9]); //
            GEDISensor.Y=TempData.FloatValue-GEDISensor.Y0;

        TempData.HexValue=(Data.Data[10]<<24)+(Data.Data[11]<<16)+(Data.Data[12
]<<8)+(Data.Data[13]);
            GEDISensor.X=TempData.FloatValue-GEDISensor.X0;
            STM_EVAL_LEDToggle(LED3);
            vTaskDelay(500);
        }
}

void ENCODERS(void *pvparameters)
{
        uint16_t temp=0;
        int   of;
        while(1)
        {
            temp=(uint16_t) TIM_GetCounter(TIM3);
            of=-temp+GEDISensor.LEnc;
            if((of)>60000)
            {
                GEDISensor.LEInc=65535-of+GEDISensor.LEInc;
            }
            else if((of)<-60000)
            {
                GEDISensor.LEInc=65535+of+GEDISensor.LEInc;
            }
            else
            {
                GEDISensor.LEInc=of+GEDISensor.LEInc;
            }
            GEDISensor.LEnc=temp;

            temp=(uint16_t) TIM_GetCounter(TIM5);
            of=temp-GEDISensor.REnc;
            if((of)>60000)
            {
                GEDISensor.REInc=65535-of+GEDISensor.REInc;
            }
            else if((of)<-60000)
            {
                GEDISensor.REInc=65535+of+GEDISensor.REInc;
            }
            else
            {
```

```
                        GEDISensor.REInc=of+GEDISensor.REInc;
                  }
                  GEDISensor.REnc=temp;
            STM_EVAL_LEDToggle(LED6);
            vTaskDelay(23);
            }
      }

void MissionController1(void *pvparameters)
{
      uint8_t i;
      uint8_t j;
      uint16_t Track;
      //State Matrices
      double Hk[8][7];
      double Fk[7][7];
      double Gk[7][2];
      double Jk[8][2];
      double xnext[7]={0};
      double xold[7]={0};

      //Uncertainty and Structural Matrices
      double Ax[7][7]={{0}};
      double Ay[8][7]={{0}};
      double B1[7][7]={{0}};
      double B2[7][2]={{0}};
      double W[7][7];

      //Measurement and Input Matrices
      double Yk[8];
      double U[2];

      //Noise Matrices
      double P[7][7]={{0}};
      double Q[8][8]={{0}};

      //Path Tracking Matrices
      double E1[3][7];
      double E2[3][7];

      ///Adjustable Scalars
      double Dt=.785;
      double Dte=.785;
      double gam=0.1;
      double kb=0.154;
      double kt=0.13;
      double nr=20;
      double Iw=0.764;
      double m=230;
      double Nc=4096;
      double B=0.88;
      double Icr=127.7;
      double rw=0.2794;
      double Ra=0.32;

      //Heading Angles and delta variables
      double THk=0;
      double thi=0;//M_PI_4;
      double Dx=0;
      double Dy=0;
      double DTHk=0;
      //================== Path Matrix========================
      path[0][0]=0;            path[0][1]=0;            path[0][2]=0;
      path[0][3]=0;            path[0][4]=0;            path[0][5]=0;
      path[0][6]=0;            path[0][7]=0;
      path[1][0]=0;            path[1][1]=1;            path[1][2]=2;
      path[1][3]=3;            path[1][4]=4;            path[1][5]=5;
      path[1][6]=6;            path[1][7]=7;
      path[2][0]=0;            path[2][1]=0;            path[2][2]=0;
      path[2][3]=0;            path[2][4]=0;            path[2][5]=0;
      path[2][6]=0;            path[2][7]=0;

      Xr=0;
      Yr=0;
      Pid=0;
      Track=0;
      xold[0]=0.1; xold[1]=0.1; xold[2]=0.1; xold[3]=0; xold[4]=0; xold[5]=0;
xold[6]=0;
      while(1)
      {
    Dx=path[1][Pid]-Xr;
    Dy=path[2][Pid]-Yr;
      probe[25]=Dx;
      probe[26]=Dy;
    DTHk=THk-xold[4];
    probe[27]=DTHk;
    if(fabs(Dx)<1E-1)
    {
      THk=M_PI_4;
    }
    else
    {
      THk=atan((Dy)/(Dx));
      }
    probe[28]=THk;
    DTHk=THk-xold[4];
    probe[29]=DTHk;
    thi=THk;
    xold[4]=thi;
      //================== Matrix F(k)====================================
      Fk[0][0]=(double)(1-(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw)));
Fk[0][1]=0;
      Fk[0][2]=0;
      Fk[1][0]=0;
      Fk[1][1]=(double)(1-(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw)));
      Fk[1][2]=0;
      Fk[2][0]=0;
      Fk[2][1]=0;
                  Fk[2][2]=(double)(1-
(2*kt*kb*nr*nr*Dt)/(Ra*(4*Iw+m*rw*rw)));
      Fk[3][0]=0;
      Fk[3][1]=0;
                  Fk[3][2]=0;
      Fk[4][0]=0;
      Fk[4][1]=0;
                  Fk[4][2]=0;
      Fk[5][0]=0;
      Fk[5][1]=0;
                  Fk[5][2]=(double)(-
(2*kt*kb*nr*nr*Dt)*Dt*cos(thi)/(Ra*(4*Iw+m*rw*rw)));
      Fk[6][0]=0;
      Fk[6][1]=0;
                  Fk[6][2]=(double)(-
(2*kt*kb*nr*nr*Dt)*Dt*sin(thi)/(Ra*(4*Iw+m*rw*rw)));

      Fk[0][3]=0;
            Fk[0][4]=0;       Fk[0][5]=0;       Fk[0][6]=0;
      Fk[1][3]=0;
            Fk[1][4]=0;       Fk[1][5]=0;       Fk[1][6]=0;
      Fk[2][3]=0;
            Fk[2][4]=0;       Fk[2][5]=0;       Fk[2][6]=0;
      Fk[3][3]=(double)(1+(B*B*kt*kb*nr*nr*Dt)/(4*Icr*Ra*rw*rw)); Fk[3][4]=0;
      Fk[3][5]=0;       Fk[3][6]=0;
      Fk[4][3]=(double)((B*B*kt*kb*nr*nr*Dt)/(4*Icr*Ra*rw*rw)*Dt);Fk[4][4]=1;
      Fk[4][5]=0;       Fk[4][6]=0;
      Fk[5][3]=0;
            Fk[5][4]=0;       Fk[5][5]=1;       Fk[5][6]=0;
      Fk[6][3]=0;
            Fk[6][4]=0;       Fk[6][5]=0;       Fk[6][6]=1;
      //========================= Matrix H(k)=============================
      Hk[0][0]=(double)(Nc*Dte/(M_2_PI*rw));    Hk[0][1]=0;       Hk[0][2]=0;
                  Hk[0][3]=0;                   Hk[0][4]=0;
      Hk[0][5]=0;       Hk[0][6]=0;
      Hk[1][0]=0;
      Hk[1][1]=(double)(Nc*Dte/(M_2_PI*rw));Hk[1][2]=0;     Hk[1][3]=0;
            Hk[1][4]=0;                   Hk[1][5]=0;       Hk[1][6]=0;
      Hk[2][0]=0;                         Hk[2][1]=0;
      Hk[2][2]=(double)(1/Dt);Hk[2][3]=0;                   Hk[2][4]=0;
            Hk[2][5]=0;       Hk[2][6]=0;
      Hk[3][0]=0;                         Hk[3][1]=0;
      Hk[3][2]=(double)(1/Dt);Hk[3][3]=0;                   Hk[3][4]=0;
            Hk[3][5]=0;       Hk[3][6]=0;
      Hk[4][0]=0;                         Hk[4][1]=0;
      Hk[4][2]=0;                   Hk[4][3]=1;                   Hk[4][4]=0;
                  Hk[4][5]=0;       Hk[4][6]=0;
      Hk[5][0]=0;                         Hk[5][1]=0;
      Hk[5][2]=0;                   Hk[5][3]=0;
      Hk[5][4]=(double)(1/Dt);Hk[5][5]=0;       Hk[5][6]=0;
      Hk[6][0]=0;                         Hk[6][1]=0;
      Hk[6][2]=0;                   Hk[6][3]=0;                   Hk[6][4]=0;
                  Hk[6][5]=1;       Hk[6][6]=0;
      Hk[7][0]=0;                         Hk[7][1]=0;
      Hk[7][2]=0;                   Hk[7][3]=0;                   Hk[7][4]=0;
                  Hk[7][5]=0;       Hk[7][6]=1;
      //====================== Matrix G(k)==============================
      Gk[0][0]=(double)(2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));
      Gk[0][1]=(double)(0);
      Gk[1][0]=(double)0;
                        Gk[1][1]=(double)((2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw)));
```

```
Gk[2][0]=(double)(kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw));
Gk[2][1]=(double)((kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw)));
Gk[3][0]=(double)(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw);
Gk[3][1]=(double)(-(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw));
Gk[4][0]=(double)((B*kt*nr*Dt)/(2*Icr*Ra*rw*rw)*Dt);
Gk[4][1]=(double)(-(B*kt*nr*Dt)/(2*Icr*Ra*rw*rw)*Dt);
Gk[5][0]=(double)((2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*cos(thi));
Gk[5][1]=(double)((2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*cos(thi));
Gk[6][0]=(double)((2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*sin(thi));
Gk[6][1]=(double)((2*kt*nr*Dt)/(Ra*(4*Iw+m*rw*rw))*Dt*sin(thi));
//======== Matrix E1================================
E1[0][0]=0;                    E1[0][1]=0;                    E1[0][2]=0;
            E1[0][3]=0;                    E1[0][4]=1;        E1[0][5]=0;
E1[0][6]=0;
E1[1][0]=0;                    E1[1][1]=0;                    E1[1][2]=0;
            E1[1][3]=0;                    E1[1][4]=0;        E1[1][5]=1;
E1[1][6]=0;
E1[2][0]=0;                    E1[2][1]=0;                    E1[2][2]=0;
            E1[2][3]=0;                    E1[2][4]=0;        E1[2][5]=0;
E1[2][6]=1;
//==================== Matrix E2==============================
E2[0][0]=-.43*Dt;      E2[0][1]=.43*Dt;        E2[0][2]=0;
E2[0][3]=-Dt;          E2[0][4]=DTHk/(xold[4]);      E2[0][5]=0;
      E2[0][6]=0;
E2[1][0]=Dt*cos(THk);  E2[1][1]=Dt*cos(THk);   E2[1][2]=-Dt*cos(THk);
E2[1][3]=0;            E2[1][4]=0;
E2[1][5]=Dx/(xold[5]); E2[1][6]=0;
E2[2][0]=Dt*sin(THk);  E2[2][1]=Dt*sin(THk);   E2[2][2]=-Dt*sin(THk);
E2[2][3]=0;            E2[2][4]=0;                            E2[2][5]=0;
            E2[2][6]=Dy/(xold[6]);
for (i=0;i<3;i++)
{
    for (j=0;j<7;j++)
    {
      E2[i][j]=E2[i][j];//0;
    }
 }
//============================ Matrix W========================
for (i=0;i<7;i++)
{
    for (j=0;j<7;j++)
    {
        if (i==j)
        {
            W[i][j]=1;
        }
        else
        {
            W[i][j]=0;
        }
    }
}
//================= Covariance P===========================
for (i=0;i<7;i++)
{
    for (j=0;j<7;j++)
    {
            if (i==j)
            {
                P[i][j]=1E-6;
            }
            else
            {
                P[i][j]=0;
            }
        }
    }
    //=========================== Covariance Q============================
    for (i=0;i<8;i++)
    {
        for (j=0;j<8;j++)
        {
            if (i==j)
            {
                Q[i][j]=1E-6;
            }
            else
            {
                Q[i][j]=0;
            }
        }
    }
    //=================== Uncertainty matrix A_\xi=======================
    for (i=0;i<7;i++)
    {
        for (j=0;j<4;j++)
        {
            if (i==j)
            {
                Ax[i][j]=.001;//.5;
            }
            else
            {
                Ax[i][j]=0;
            }
        }
        for (j=4;j<7;j++)
        {
            Ax[i][j]=0;
        }
    }
    Ax[4][3]=0.005;//
    Ax[4][2]=0.005;//
    Ax[6][2]=0.005;//
//  //================ Uncertainty matrix A_Y==========================
    Ay[0][0]=0;
    Ay[1][1]=0;
    Ay[2][2]=.0001;
    Ay[3][2]=.0001;
    Ay[4][3]=.0001;
    Ay[5][4]=.0001;
    Ay[6][5]=0;
    Ay[7][6]=0;
```

```
//    //=============== Uncertainty matrix B1========================
      for (i=0;i<7;i++)
      {
          for (j=0;j<7;j++)
          {
              if (i==j)
              {
                 B1[i][j]=1E-5;//.25;
              }
              else
              {
                 B1[i][j]=0;
              }
          }
      }
//    //================ Uncertainty matrix B2=======================
      for (i=0;i<7;i++)
      {
          for (j=0;j<2;j++)
          {
              if (i==j)
              {
                  B2[i][j]=1E-5;//.5;
              }
              else if(i==j*3)
              {
                  B2[i][j]=1E-5;//.5;
              }
              else
              {
                  B2[i][j]=0;
              }
          }
      }
      //============================= Initial Ups ===============
      //======================= Get Filter Matrices ======
            KalmanFilter(Ax,Ay,B1,Hk,Fk,P,Q,gam);
          //=============== Control Algorithm ========================
            H2Control(Ax,B1,B2,W,Fk,Gk,E1,E2);
            //============== Measure sensors for vector Yk ==============

                  Yk[0]=(double)(GEDISensor.LEInc);

                  Yk[1]=(double)(GEDISensor.REInc);

                  Yk[2]=(double)(GEDISensor.AX1/GEDISensor.NIMU)*Dt;

                  Yk[3]=(double)(GEDISensor.AX1/GEDISensor.NIMU)*Dt;

                  Yk[4]=(double)(GEDISensor.GZ1/GEDISensor.NIMU)*Dt;

                  Yk[5]=(double)(GEDISensor.GZ2/GEDISensor.NIMU);

                  Yk[6]=(double)(GEDISensor.X);

                  Yk[7]=(double)(GEDISensor.Y);
```

```
        double Umax;


      //=========== Get Filter Matrices =======
    for (i=0;i<7;i++)
    {
        xnext[i]= 0;
        for (j=0;j<7;j++)
        {
            xnext[i]=xnext[i]+PHIk[i][j]*xold[j];
        }
        for (j=0;j<8;j++)
        {
            xnext[i]=xnext[i]+GAMMAk[i][j]*Yk[j];
        }
    }
    //================= Update State Estimate =========================
    for (i=0;i<7;i++)
    {
        xold[i]=xnext[i];
        probe[i]=xnext[i];
    }

   //================== Calculate U from Lamdak =====================
    for (i=0;i<2;i++)
    {
      U[i]=0;
      Umax=U[i];
        for (j=0;j<7;j++)
        {
            U[i]=U[i]+LAMk[i][j]*xold[j];
        }
        if (fabs(U[i])>Umax) {

        Umax=U[i];
        }
        U[i]=-U[i];
        probe[35+i]=U[i];
    }

    if (Umax>8)
    {
      for (i=0;i<2;i++)
      {
        U[i]=U[i]*8/Umax;
      }
    }


    Xr=xold[5];
    Yr=xold[6];
    //================== Reset Measurements =====================
      GEDISensor.LEInc=0;
      GEDISensor.REInc=0;
      GEDISensor.AX1=0;
```

```
            GEDISensor.AX2=0;
            GEDISensor.GZ1=0;
            GEDISensor.GZ2=0;
            GEDISensor.NIMU=0;
          //============= Voltage To PWM ===============================
      if(Track<40)
      {
            Track++;
      }

      else
      {
            if(Pid>8)
               {
                     Controls.VL=ZeroSpeed;
                     Controls.VR=ZeroSpeed;
               }
               else {

                        Pid++;

                        Controls.VR=U[0]+ZeroSpeed;
                     if (Controls.VR>MaxNegSpeed)
                     {
                        Controls.VR=MaxNegSpeed;
                     }
                     else if(Controls.VR<MaxPosSpeed)
                     {
                        Controls.VR=MaxPosSpeed;
                     }
                        Controls.VL=U[1]+ZeroSpeed;
                     if (Controls.VL>MaxNegSpeed)
                     {
                        Controls.VL=MaxNegSpeed;
                     }
                     else if(Controls.VL<MaxPosSpeed)
                     {
                        Controls.VL=MaxPosSpeed;
                     }
               }
      }

          vTaskDelay(785);
          STM_EVAL_LEDToggle(LED5);
          STM_EVAL_LEDToggle(LED4);
      }
}

void IMU(void *pvparameters)
{
      while(1)
      {
            int16_t Temp;

            GPIO_ResetBits(GPIOE,GPIO_Pin_1);   //Set CS Low//3
            Temp=0;
            Temp=(mySPI_GetData(0x29));                     //Get High byte
            Temp=((Temp<<8)|(mySPI_GetData(0x28)));   //Get Low byte, combine
            GEDISensor.AX1=((Temp)*(1/393.5)+GEDISensor.AX1);//Convert to acc
            GPIO_SetBits(GPIOE,GPIO_Pin_1);           //Set CS High//3


            GPIO_ResetBits(GPIOE,GPIO_Pin_2);   //Set CS Low//4
            Temp=0;
            Temp=mySPI_GetData(0x2D);                       //Get High byte
            Temp=((Temp<<8)|(mySPI_GetData(0x2C)));   //Get Low byte, combine
            GEDISensor.GZ1=(((Temp+1)*(2/29.1))*(M_PI/180)+GEDISensor.GZ1);
            GPIO_SetBits(GPIOE,GPIO_Pin_2);           //Set CS high//4

            GPIO_ResetBits(GPIOE,GPIO_Pin_3);   //Set CS Low//3
            Temp=0;
            Temp=(mySPI_GetData(0x29));                     //Get High byte
            Temp=((Temp<<8)|(mySPI_GetData(0x28)));   //Get Low byte, combine

            GEDISensor.AX2=((Temp)*(1/393.5)+GEDISensor.AX2);
            GPIO_SetBits(GPIOE,GPIO_Pin_3);           //Set CS High//3


            GPIO_ResetBits(GPIOE,GPIO_Pin_4);   //Set CS Low//4
            Temp=0;
            Temp=mySPI_GetData(0x2D);                       //Get High byte
            Temp=((Temp<<8)|(mySPI_GetData(0x2C)));   //Get Low byte, combine
            GEDISensor.GZ2=(((Temp+1)*(2/29.1))*(M_PI/180)+GEDISensor.GZ2);
            GPIO_SetBits(GPIOE,GPIO_Pin_4);           //Set CS high//4


            STM_EVAL_LEDToggle(LED6);
            GEDISensor.NIMU++;
            vTaskDelay(7);
      }
}


void DriveRobot(void *pvparameters)
{
      while(1)
      {
      TIM4->CCR1=Controls.VL;
      TIM4->CCR2=Controls.VR;
      STM_EVAL_LEDToggle(LED6);
      vTaskDelay(53);
      }
}

/*============================================================================
                  Main Program
============================================================================*/

int main(void)
{
      uint8_t i;
      uint8_t j;
      /*=================================
       *                Initialize Code
```

```
	 * ==================================*/
	STM_EVAL_LEDInit(LED3);
	STM_EVAL_LEDInit(LED4);
	STM_EVAL_LEDInit(LED5);
	STM_EVAL_LEDInit(LED6);
	Configure_DCMotors();
	GPS_Configure();
	Encoder_Configure();
	IMU_Configure();
	GEDISensor.NIMU=1;
	GetInitCoordinates();
	for (i=0;i<7;i++)
	{
	    for (j=0;j<7;j++)
	    {
	        if (i==j)
	        {
	            UpsNew[i][j]=100;//1E10;
	        }
	        else
	        {
	            UpsNew[i][j]=0;
	        }
	    }
	}

	/*==================================
	 *                 Create Tasks
	 * ==================================*/
	xTaskCreate( GPS, ( signed char * ) "Read GPS", 100, NULL, 1, NULL );
	xTaskCreate( ENCODERS, ( signed char * ) "Read Encoders", 50, NULL, 2,
NULL );
	xTaskCreate( IMU, ( signed char * ) "Read IMU", 50, NULL, 2, NULL );
    xTaskCreate( DriveRobot, (signed char*) "Motors",50,NULL,1,NULL);
    xTaskCreate( MissionController1, (signed char*)
"Motion",5000,NULL,3,NULL);
	/*==================================
	 *                 Run Tasks
	 * ==================================*/

    vTaskStartScheduler(); // This should never return.


    while(1)
    {
    }

    return 1;
}
```