2019

EFFECT OF SENSOR ERRORS ON AUTONOMOUS STEERING CONTROL AND APPLICATION OF SENSOR FUSION FOR ROBUST NAVIGATION

Shuvodeep Bhattacharjya
Michigan Technological University, sbhatta2@mtu.edu

EFFECT OF SENSOR ERRORS ON AUTONOMOUS STEERING CONTROL AND APPLICATION OF SENSOR FUSION FOR ROBUST NAVIGATION

By

Shuvodeep Bhattacharjya

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mechanical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2019

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mechanical Engineering.

Department of Mechanical Engineering – Engineering Mechanics

Report Advisor: *Dr. Jeffrey D. Naber*

Committee Member: *Dr. Jeremy Worm*

Committee Member: *Dr. Darrell L. Robinette*

Department Chair: *Dr. William W. Predebon*

TABLE OF CONTENTS

LIST OF FIGURES v

LIST OF TABLES viii

ACKNOWLEDGEMENTS ix

ABSTRACT x

1 INTRODUCTION 1
- 1.1 Autonomous Vehicles 1
- 1.2 Typical System Architecture for Automated Driving 2
- 1.3 Sensors Used for Perception and Vehicle State Estimation 3
- 1.4 Research Organization and Objective 3

2 LITERATURE REVIEW 5
- 2.1 Autonomous Steering Controllers 5
- 2.2 Types of Errors in Sensors 6
- 2.3 Current Work in the areas of sensors and their limitations 8
- 2.4 Summary 9

3 NEED FOR CONTROLLER PERFORMANCE ANALYSIS 10
- 3.1 Experimental Setup 10
- 3.2 Selection of Controller Parameters 12
- 3.3 Controller Objective 12
- 3.4 Sensors Used 12
- 3.5 Test Results 13
- 3.6 Analysis 14

4 MODEL BASED CONTROLLER AND SENSOR ANALYSIS 15
- 4.1 Selection of Steering Controllers for Analysis 15
- 4.2 Modelling approach for sensors, actuators and vehicle kinematics 18

5 SIMULATION RESULTS AND ANALYSIS 39
- 5.1 Performance Analysis under Ideal Sensor Conditions 39
- 5.2 Performance Analysis Considering Sensor Errors 43

6 IMPROVING NAVIGATION / WAYPOINT TRACKING USING STATE ESTIMATION APPROACH....48

6.1 Kalman Filter Equations....49

6.2 Implementation of 1D – 2^{nd} Order Kalman Filter for Improved Position Feedback in Straight Line Path....50

6.2.1 Filter Results for Various Controllers Under 1D conditions....51

6.3 Implementation of 1^{st} Order Kalman Filter for Vehicle Heading Improvement 53

6.3.1 Filter Implementation Results for Vehicle Heading Estimation....53

7 CONCLUSION AND FUTURE SCOPE OF WORK....55

8 REFERENCES....56

9. APPENDIX....58

9.1 Python code used for initial vehicle test and analysis....58

9.2 Hardware Specifications....67

9.2.1 Controller Specification....67

9.2.2 Sensor Specifications....68

9.2.3 Test Vehicle Specification....70

LIST OF FIGURES

Figure 1-1: Typical System Architecture for Automated Driving [21] 2

Figure 1-2: Research Organization and Objective 4

Figure 2-1: Vehicle Coordinate System...7

Figure 3-1: Test Location for Getting Experimental Data 10

Figure 3-2: Flowchart for Waypoint Navigation 11

Figure 3-3: Test Result 1 - Comparison between Ideal Path and Actual Path..................13

Figure 3-4: Test Result 2 - Comparison between Ideal Path and Actual Path..................13

Figure 4-1: Effect of Look Ahead Distance [27] 16

Figure 4-2: Path Parameters for Stanley Controller [3][5] 17

Figure 4-3: Controller and Plant Model for Analysis 18

Figure 4-4: Results for Standard Deviation Analysis in X & Y direction GPS sensor modelling 19

Figure 4-5: Standard Deviation Analysis for Yaw or Current Heading 20

Figure 4-6: Standard Deviation Analysis of Accelerometer in X & Y Direction.............22

Figure 4-7: Straight Line Test @ 1m/s for Transfer Function Generation 23

Figure 4-8: Standard Deviation Analysis for Gyroscope Yaw-Rate 23

Figure 4-9: Circle Test Results 24

Figure 4-10: Gyroscope (Yaw-Rate) Output for Circle Test 25

Figure 4-11: Analysis of Test Data and Simulation Data for Transfer Function Generation for Gyroscope..25

Figure 4-12: Schematic - Steering System Duty Cycle Decoding Process 26

Figure 4-13: Schematic - Steering System Duty Cycle Verification Process....................27

Figure 4-14: Actual Response vs. Ideal Response Analysis for Transfer Function 28

Figure 4-15: Actuator Hysteresis Modelling 28

Figure 4-16: Drivetrain Dynamics 29

Figure 4-17: Vehicle Kinematics Model 29

Figure 4-18: Interfacing of Vehicle States with Sensor Blocks 31

Figure 4-19: Start Stop type Speed Control 31

Figure 4-20: Flowchart to Determine Target Heading for Waypoint Based Controllers ..33

Figure 4-21: Flowchart to Determine Path Heading for Stanley Controller 33

Figure 4-22: PI Controller Implementation 34

Figure 4-23: Look Ahead Distance for Pure Pursuit Controller 35

Figure 4-24: Pure-Pursuit Controller Implementation 35

Figure 4-25: Stanley Controller Implementation 36

Figure 4-26: Custom Path 37

Figure 4-27: Straight Path 37

Figure 4-28: Dynamic Lane Change 38

Figure 5-1: Path Tracking Performance of Controllers 39

Figure 5-2: Cross Track Error of Vehicle on Custom Path 39

Figure 5-3: Path Tracking Performance of Controllers on Straight Path 40

Figure 5-4: Cross Track Error of Vehicle on Straight Path 40

Figure 5-5: Path Tracking Performance of Controllers for Dynamic Lane Change 41

Figure 5-6: Cross Track Error of Vehicle for Dynamic Lane Change 41

Figure 5-7: Effect of Sensor Errors and Location Specific Noise on Navigation Performance 43

Figure 5-8: Effect of Sensor Errors on Cross Track Error for Custom Path 44

Figure 5-9: Effect of Stray Magnetic Fields on Magnetometer Output for Vehicle Heading 44

Figure 5-10: Effect of Sensor Errors Navigation Performance under 1D condition 45

Figure 5-112: Effect of Sensor Errors on Navigation Performance 46

Figure 5-123: Effect of sensor error on Cross Track Error...47

Figure 6-1: Controller Performance with 1D Kalman Filter, MR=10 51

LIST OF TABLES

Table 1-1: SAE Levels of Automated Driving [20]..1

Table 4-1: Sensor Errors for GPS Modelling 19

Table 4-2: Sensor Errors for Magnetometer Modelling 20

Table 4-3: Sensor Errors for Accelerometer Modelling 22

Table 4-4: Sensor Errors for Gyroscope Yaw-Rate Modelling 24

Table 4-5: Sensor Errors for Wheel Speed Sensor Modelling..26

Table 4-6: Commanded vs Actual Steering Angle 27

Table 4-7: Sign Convention for Cross Track Error 32

Table 5-1: Max. Cross Track Error (Meters) Results for Custom Path............................42

Table 5-2: Max. Cross Track Error (Meters) Results for Custom Path Considering Sensor Errors..45

Table 5-3: Distance (Meters) between vehicle stop point and actual waypoint for straight line test for 1D condition 46

Table 5-4: Max. Cross Track Error (Meters) Results for Dynamic Lane Change Considering Sensor Errors 47

Table 6-1: Difference in distance between vehicle stop point and waypoint for various controllers with Kalman Filter MR=10 51

Table 6-2: Distance (Meters) between vehicle stop point and waypoint for various controllers with Kalman Filter MR=10 with GPS Error included.........................52

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Dr. Jeffrey D. Naber, for giving me the opportunity to work on this research project, providing the work space and equipment, and his guidance during the project work.

I would like to thank the rest of my committee members: Dr. Jeremy Worm, Dr. Darrell Robinette and Christopher Morgan, for giving their valuable time to my report defense and for technical suggestions.

My sincere thanks to Paul Dice, Research Engineer, APSRC and his team of engineers for their guidance and technical support in this project.

My special thanks to Ahammad Basha, Phd. Student, APSRC who introduced me to this project and to the world of autonomous vehicles. I would also like to thank my teammates for their technical support and coordination.

My special thanks to Dilip Ati, Grad. Student, Computer Science, for providing me the literatures and documents in the areas of sensor fusion.

Last but not least, I would like to thank my family and friends for their support and encouragement throughout this research project.

ABSTRACT

Autonomous steering control is one the most important features in autonomous vehicle navigation. The nature and tuning of the controller decides how well the vehicle follows a defined trajectory. A poorly tuned controller can cause the vehicle to oversteer or understeer at turns leading to deviation from a defined path. However, controller performance also depends on the state–feedback system. If the states used for controller input are noisy or has bias / systematic error, the navigation performance of the vehicle is affected irrespective of the control law and controller tuning. In this report, autonomous steering controller analysis is done for different kinds of sensor errors and the application of sensor fusion using Kalman Filters is discussed. Model-in-the-loop (MIL) simulation provides an efficient way for developing and performing controller analysis and implementing various fusion algorithms. Matlab/Simulink was used for this Model Based Development. Firstly, through experimentation the path tracking performance of the controller was analyzed followed by data collection for sensor, actuator and vehicle modelling. Then, the plant, actuator and controllers were modelled followed by the comparison of the results for ideal and non-ideal sensors. After analyzing the effects of sensor error on controller and vehicle performance, a solution was proposed using 1D-Kalman Filter (KF) based sensor fusion technique. It is seen that the waypoint tracking under 1D condition is improved to centimeter level and the steering response is also smoothened due to less noisy vehicle heading estimation.

1 INTRODUCTION

1.1 Autonomous Vehicles

Autonomous vehicles are robots capable of operating on public roads by perceiving the environment using sensors i.e. GPS for real time positional information, perception devices to detect obstacles, signage, road geometry, inertial sensors for vehicle states, etc. and make decisions using complex algorithms to follow appropriate navigational paths.

Autonomous vehicles can both be a boon and a bane for the society. Advantages of automated driving include better safety which is due to reduction in traffic collisions and related costs. Automated cars under certain predictable conditions tend to increase traffic flow which results in enhanced mobility for people and can relieve travelers from driving and navigation chores, increase fuel efficiency of a vehicle and facilitate business models for transportation industry. The disadvantages include high initial cost due to complexity in design, reliability under unpredictable conditions, legal framework and government regulations, costs associated with infrastructure and loss of driving-related jobs in the transportation industry.

Autonomous vehicles can have varying degree of automated driving i.e. from no to semi-autonomous to completely autonomous. SAE classifies the autonomous vehicles as follows, in table 1-1 based on different levels of driving automation [20]:

Table 1-1: SAE Levels of Automated Driving [20]

<table>
<tr><th>SAE Level</th><th>Involvement of Human</th><th>Function of Feature for Automated Driving</th><th>Feature Example</th></tr>
<tr><td>0</td><td rowspan="3">Always be in control of the vehicle</td><td>No support or automation</td><td>--</td></tr>
<tr><td>1</td><td>Provide warning and prompt for corrective action</td><td>a) Blind spot warning
b) Lane Departure Warning
c) Cruise Control</td></tr>
<tr><td>2</td><td>Provide support in the form of steering / brake assist</td><td>a) Lane Departure Assist
b) Adaptive Cruise Control</td></tr>
<tr><td>3</td><td rowspan="2">Not driving when the feature is active, but requires human involvement when the feature requests</td><td rowspan="2">Automated driving under certain conditions like highways, geo-fenced location, parking lots, etc.</td><td>a) Traffic Jam Chauffeur
b) Automated Valet Parking</td></tr>
<tr><td>4</td><td>Location specific driverless taxi service</td></tr>
<tr><td>5</td><td>No human involvement under any driving scenario</td><td>The vehicle can drive under all conditions</td><td>Driverless or Steering less vehicle</td></tr>
</table>

1.2 Typical System Architecture for Automated Driving

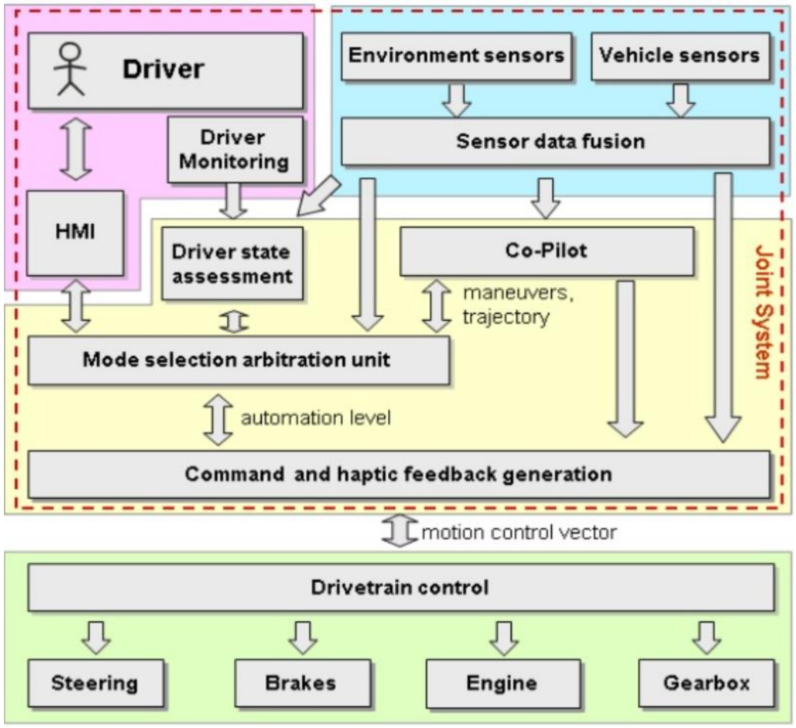

Figure 1-1: Typical System Architecture for Automated Driving [21]

From figure 1.2.1, the various stages of automated driving viz. from sensing to issuing commands for actuation is described as follows:

Stage 1: Perception and Driver Monitoring – In this stage, the environment is perceived for pedestrians, nearby vehicles, obstacles, road geometry and signage, and the states related to the motion and position of the vehicle is measured. A sophisticated fusion algorithm is used to combine all the sensory data to remove noise and errors in the measured data and give a better estimate of the vehicles states and environment. Simultaneously, the state of driver is also perceived via. sensors or through driver inputs from HMI.

Stage 2: Decision Making –Based on the inputs from the previous stage and stored road maps, decisions are made regarding the efficient and the safest path/route required for navigation, followed by the decisions for vehicle motion like velocity and steering angles. The algorithms used at this stage are very complex and of robust nature such that, failure of one sensor will not risk or affect the vehicle / driver.

Stage 3: Vehicle Motion / Drivetrain Control –Based on the velocity and steering angle commands the required actuation signals are generated.

1.3 Sensors Used for Perception and Vehicle State Estimation

a.) Environment Perception Sensors – Monocular / Stereo Camera, 2D/3D LIDAR, RADAR, Ultrasonic Sensors, Infrared Sensors.

b.) Drive State Monitoring – Camera, Infrared Sensors, Body Sensors like heart rate monitor [21] integrated on the seats

c.) Vehicle Position and Motion Sensors – Global Positioning Systems (GPS), Wheel Speed Sensors, Inertial Measurement Unit (IMU), Steering Angle Sensor

Processing data from all these sensors is one of the biggest challenges in the areas of autonomous driving. Processing is generally done in two stages – conversion of bit stream to engineering units followed by filtering of noise. The second stage requires the high amount of processing power as it involves the use of complex algorithms to remove noise / unwanted data.

1.4 Research Organization and Objective

This research is organized into 7 Chapters as depicted in Figure 1-2. The overall goal of this research is to highlight the effects of sensor errors on automatic steering control and improve the navigation performance by application of sensor fusion. This is done by conducting an experiment on a Remote Controlled (RC) vehicle, on which we installed the sensors, mentioned in section 3.4 having specification as per section 9.2.2 and bypassed the vehicle controller with our programmed controller, the specification of which is given in section 9.2.1. The results were analyzed in Chapter 3, followed by the modeling of the vehicle, actuators, sensors and the controllers in Chapter 4. The model was used to analyze steering controller performance under various path conditions for both ideal sensor feedback and noisy sensor feedback. The simulation results in Chapter 5 led to the implementation of sensor fusion via. Kalman Filter for 1-D waypoint tracking and vehicle heading estimation. The controller and vehicle model developed in chapter 4 is used in chapter 6 for tuning the filter for the specific application. The simulation results will show the improvement in waypoint tracking and vehicle heading estimation in the presence stray magnetic fields and disturbances.

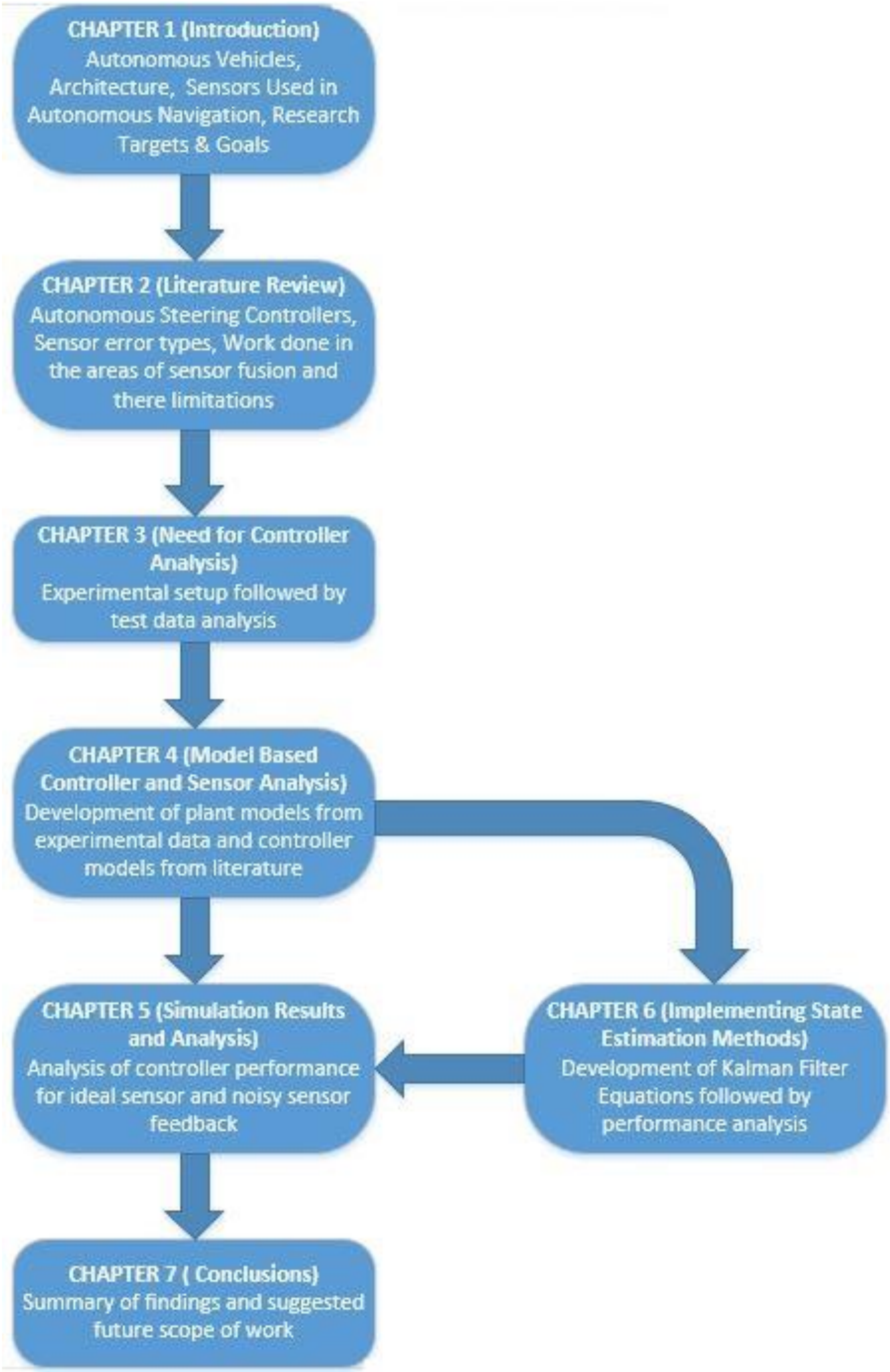

Figure 1-2: Research Organization and Objective

2 LITERATURE REVIEW

2.1 Autonomous Steering Controllers

In autonomous driving steering controls play a major part when it comes to path following and vehicle trajectory control. It consists of an algorithm which generates the required control outputs in the form of vehicle steering angle based on path generated and the vehicle dynamics. *Snider* in [3] discusses the various steering methods used for autonomous driving. The steering controllers are classified into various categories:

- Geometric
- Kinematic
- Optimal
- Preview / Predictive Type

A comparison between different types of controllers is made by *Snider* in *Figure 48* of [3]. It can be seen that the Pure Pursuit controller which is a proportional controller, is robust to disturbances, no path requirements, and is best for slow or discontinuous path driving. However, the path tracking ability is degraded once the vehicle speed increases or if the path has sharp corners. The Stanley controller which is a non-linear feedback controller developed by Stanford University [5], is slightly superior to the pure -pursuit controller when it comes to high speed driving or cornering. However, it is less robust to disturbances and has high steady state error when speed increases. The kinematic controller, even though it includes the kinematic model of the vehicle and does not cut corners, has very less robustness to disturbances, requires path curvature and its two derivatives, suffers increased steady state errors at high speed and tends to overshoot during rapidly changing corners. The low robustness and in-accuracy of the Kinematic controller can be attributed to the fact that it does not consider the path dynamics and other dynamic effects during high speeds. Also, there is an increased computational cost and increased difficulty in implementation. The Linear Quadratic Regulator (LQR) controller implements a dynamic bicycle model of the vehicle. However, solving the LQR requires high computational power since, it is an optimal control theory and is required to be solved for optimal gains for every iteration. The LQR controller performs the worst compared to the previous three controllers due to its linear nature as it excludes the non- linear path dynamics. *Snider* tried to improve the controller by adding a feed-forward term which improves high speed driving, it has the least steady state errors and does not cut corners. However, this controller has the worst robustness to disturbances and has significant overshooting problems during rapidly changing curvatures. The preview type controller is similar to a linear model predictive controller which is also a type of an optimal controller with an advantage of prediction horizon or look ahead distance similar to the Pure-Pursuit controller [3][5] allowing to account for the path dynamics. This allows for better robustness compared to the LQR and Kinematic controller, least steady state error and better control during high speed driving. However, this controller has moderate overshooting issues and cutting corners for rapidly changing vehicle speed or road curvature.

It can be inferred from [3] that a controller with higher number of state feedback variables is not necessarily more robust to noise or disturbances but it definitely makes it more complicated to implement and increases the computation requirements. Also, the results for cross track error in [3] show that every controller performs differently for different values of gains, vehicle speed and for the given track geometry. Another important point which can be inferred from [3] is that, geometric controllers are better at rejecting disturbances. One appreciable method, as described in [6] is the use of hybrid controller between Pure Pursuit and Stanley controller. In this an adaptive weighting factor is used for both the controllers where more weight is given to the look ahead nature of pure pursuit during sharp changes in trajectory and as the path smoothens the weight is shifted to Stanley controller. Other types of advance rule-based path tracking controllers like fuzzy controllers are discussed in [14].

2.2 Types of Errors in Sensors

In general, there are two primary kinds of errors associated with sensors:

- Systematic Errors / Bias
  - Can be positive or negative
  - For some sensors, it can be removed by calibration
- Noise or Random Errors
  - Can be reduced by the use of suitable signal filters
  - Can be improved by taking the average of multiple readings of the same parameter for the same system state, depending on sensor design and dynamics

Depending on the application and the manufacturing process, one form of error can be dominant over the other.

Navigational sensors like *Global Positioning Systems (GPS)* generally, have significant systematic errors. Section 9.2.2 of the appendix discusses the systematic error of the GPS under various operating conditions. Various studies have been done to identify the causes of systematic errors in GPS. Some of them are highlighted in [8] and [18]. One major reason as mentioned by *Md. R. Islam* and *J.M. Kim* in [8] is, distortion of the GPS signal by the US Department of Defense leading to selective availability to users. Another important source of error is propagation delay in the GPS signal. As mentioned in [18], humidity, hydrometeors, hygroscopic aerosol and particulates like sand, dust, aerosols, etc. in the atmosphere introduce microwave propagation delays due to refraction, dispersion and scattering of signal waves. This means that weather conditions like sandstorm, rain, hail and snowfall can also induce errors in GPS signals. Other error sources include satellite geometry i.e. number of satellite connections and their positions, multipath effect, clock inaccuracies, rounding errors, and receiver noise.

Another sensor which is commonly used in autonomous vehicles is the *Inertial Measurement Unit (IMU)*. It consists of the following:

- Magnetometer or Digital Compass – Used to measure earth's magnetic field there by giving the orientation of the vehicle w.r.t the earth's magnetic north
- Accelerometer – Used to determine the acceleration values along the x,y,z axis
- Gyroscope – Used to measure the rate of change of angle about the x,y,z axis and derive roll(ϕ), pitch(Θ) and yaw values($\dot{\varphi}$) as shown in figure 2-1

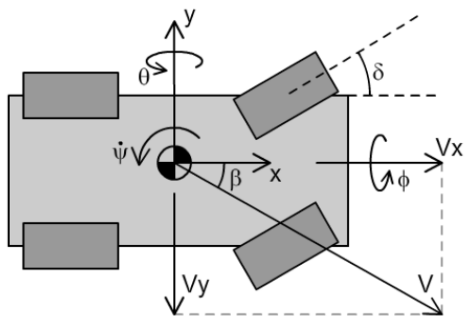

Figure 2-1: Vehicle Coordinate System

Errors in magnetometer is of both systematic and of random nature [23]. The systematic sources of errors include hard irons errors, null shift errors, soft irons errors, and scale factor errors. While the time varying errors come from nearby electronics, such as current carrying wires, on-off transition of nearby device or stray magnetic fields.

The accelerometers and gyroscopes are Micro Electro Mechanical Sensors (MEMS) [24] and these form the backbone of inertial measurements. As mentioned in [25], these sensors are fabricated on a silicon wafer using integrated circuit process sequences for electronic components and compatible micro-machining processes for micro-mechanical machining that selectively etch away parts of the silicon wafer or add new structural layers to form the mechanical and electromechanical devices. Since, machining is involved in its manufacturing, stresses are induced in the components which create bias or systematic errors in MEMS devices. Application of external forces or in-correct installations can also affect the systematic error. Random errors or noise in MEMS devices is generally due vibrations, errors from nearby electronics or by electro-magnetic interference (EMI). Sometimes in MEMS devices bias stability is an issue and they tend to drift over time. This means integration of acceleration to get velocity will induce a linear error and a quadratic error for distance. The same principle is valid, when deriving roll, pitch and yaw values from gyroscope.

From the above, it can be seen that navigational performance of GPS is largely affected by systematic errors whereas IMU's mostly have noise and drift over time. The systematic errors in GPS can be corrected by the use of Differential GPS or Real Time Kinematic (RTK) system which is a base station providing error correction signals to GPS. Although these methods require investment, they provide accuracy in the range of centimeters as mentioned in [16]. However, loss of signal or disconnection from the base-station is possible. The systematic errors in IMU can be removed by running internal calibration routines given by the manufacturer or by manually calibrating it by getting the mean of data sampled over a large time interval and subtracting it from every data. The noises can be removed by using appropriate filters. Some manufacturers provide a built in Kalman Filter or Low Pass Filter which outputs processed data. However, these add to the cost of the device.

2.3 Current Work in the areas of sensors and their limitations

Significant amount of work has been done in the areas of sensor fusion for estimating and reducing the errors in vehicle states. The entire premise of combining multiple sensory data is to overcome the limitation of individual sensors. As discussed in [26], the data obtained by combining two or more sensors has lower variance in output than each of the individual sensors. Another motivation behind sensor fusion is to derive or estimate another state variables which cannot be measured by an individual sensor. As discussed in Chapter 1, a large array of sensors are used for autonomous navigation. However, the cost involved is also high, especially with perception sensors like 3D Lidar. As discussed by *Vivacqua, Vassallo, and Martins* in [1], a low-cost sensor fusion method is proposed where GPS data is combined with prior map data and with camera data by analyzing short range lane markings, is used for localization of the vehicle. Although, this method avoids the use of costly perception sensors, the use of camera leads to the requirement of higher processing power. A similar method involving lane detection is implemented in [11] where a camera detects the lane marking and the data combined with GPS data and data from road information file is used for localization. In [4], Kalman filters are used to estimate the Error in GPS data by combining data from camera which was used to detect curved lanes and stop lines at intersections so as to improve waypoint following. In this again, GPS + RTK was used to develop reference trajectory. However, it is mentioned that this method fails in discontinuous locations of downtown areas where GPS error models are not suitable. One low cost method discussed by *Islam and Kim* in [8] is the use averaging and estimation techniques to improve GPS accuracy. However, this method improves GPS accuracy only up to 4 meters at best, which is not suitable for autonomous driving. Another method involving sensor fusion between GPS and IMU using Kalman Filter is discussed in [9] where the role of IMU is to dead reckon the GPS signals. A novel concept of contextual filtering is discussed, where to improve filter performance the bad GPS data entering the filter is rejected. A similar approach using Kalman Filters is used in [10] where GPS and IMU data is combined to improve navigational performance. However, in this 2 GPS are used and the data generated for fusion is through DGPS method or via. Carrier Phase Method, both of which can affect the filter performance when there is a loss of GPS connection. Another work discussed in [13], involves multi-sensor fusion having GPS, IMU, Ultrasonic Sensor, Camera and Laser Scanner. In this, combining multiple sensors eliminates the use of DGPS and RTK systems as it considers data from both local frame and global frame of reference. Compared to the Kalman Filter based estimation, one major drawback of this method is robustness, as the algorithm is executed serially and failure of one sensor can negatively affect the controller performance as there is no means of state-estimation. Some papers have also discussed about learning based methods. One of them is discussed in [2] which uses high precision RTK system to correct the GPS signals for improving its accuracy along with high precision IMU to collect waypoints based on which a cubic B-spline curve is generated to create a road map. This was used to provide a preview point to the Stanley controller for improved path tracking of the generated map. Another method in [12] involves the use of a learning based non-linear model predictive control which is designed for navigation in GPS denied environment and minimize path tracking errors. It uses a pre-defined vehicle model and a learned disturbance model. An on-board stereo camera was used for learning the terrain. Since, it uses a stereo camera, the image processing requirements are very high. In [14] a fuzzy controller is implemented for path tracking but it uses the fusion of Camera, DGPS, IMU and RFID. However, the paper does not discuss the fusion process or the error types associated with sensors. A study discussed in [19] by *Deilamsalehy and Havens* discusses the fusion of IMU, Camera and Lidar using an Extended Kalman Filter used to estimate the position of a vehicle in a GPS denied environments.

2.4 Summary

All the sensor related works discussed in the previous section, have some form of limitation when it comes to real-time implementation. The use of Camera or other perception devices with GPS improves the localization of the vehicle. However, it also requires high computational power. Also, in environments like snow covered roads and off-road regions where there are no road features like lane, stop-line, side-walks, etc. the perception based fusion methods can fail. The Kalman filter based methods involving the fusion of GPS / IMU are good for navigation but have drawbacks when it comes to tuning for a specific application and array of sensors. Some methods also use pre-defined maps or a road information file which again creates a requirement for high storage memory and real-time processing power. The methods used for sensor fusion have also not been tested with different types of steering controllers in real time, as discussed in section 2.1, for autonomous navigation.

3 NEED FOR CONTROLLER PERFORMANCE ANALYSIS

As discussed in previous sections, it is necessary to analyze controller performance by considering real sensor data. Sensors give the feedback of vehicle states. The output of a controller having a very high gain or an aggressive control action, can be affected by sensor errors leading to poor path tracking or navigational performance of the vehicle. Sensor noise can affect the steering ability or stability of the vehicle whereas systematic errors or bias would never allow the vehicle to have zero cross track or lateral error. Also, in [3], [6] & [14] the effects of steering actuator hysteresis and other dynamics are also not considered.

This chapter investigates the need for controller performance analysis for sensor systematic errors and noise. It is also worth investigating the effects actuator hysteresis on controller performance.

3.1 Experimental Setup

The type of vehicle and the set of hardwares used for navigation are mentioned in section 9.2 of the Appendix. The test location was APSRC, Michigan Tech. in Calumet, MI, as shown below in figure 3-1.

Figure 3-1: Test Location for Getting Experimental Data

A constant vehicle speed of 1m/s was used for the experiment. Due to the simple and versatile nature of PI control algorithm, it was used for steering control and waypoint navigation. Derivative part of the controller was not used since it would make the controller prone to high frequency noises. The code was developed in Python language and can be found in section 9.1 of the Appendix.

The flowchart in figure 3-2 explains the python code for the implementation of PI control. The following terms were considered during the development of the controller.

- *Distance to Target* - Shortest straight-line distance between vehicle current position and target point.
- *Current Heading* - Orientation of vehicle w.r.t North
- *Target Heading* - Orientation or angle of target point w.r.t to north and vehicle position
- *Heading error* - Target Heading – Current Heading

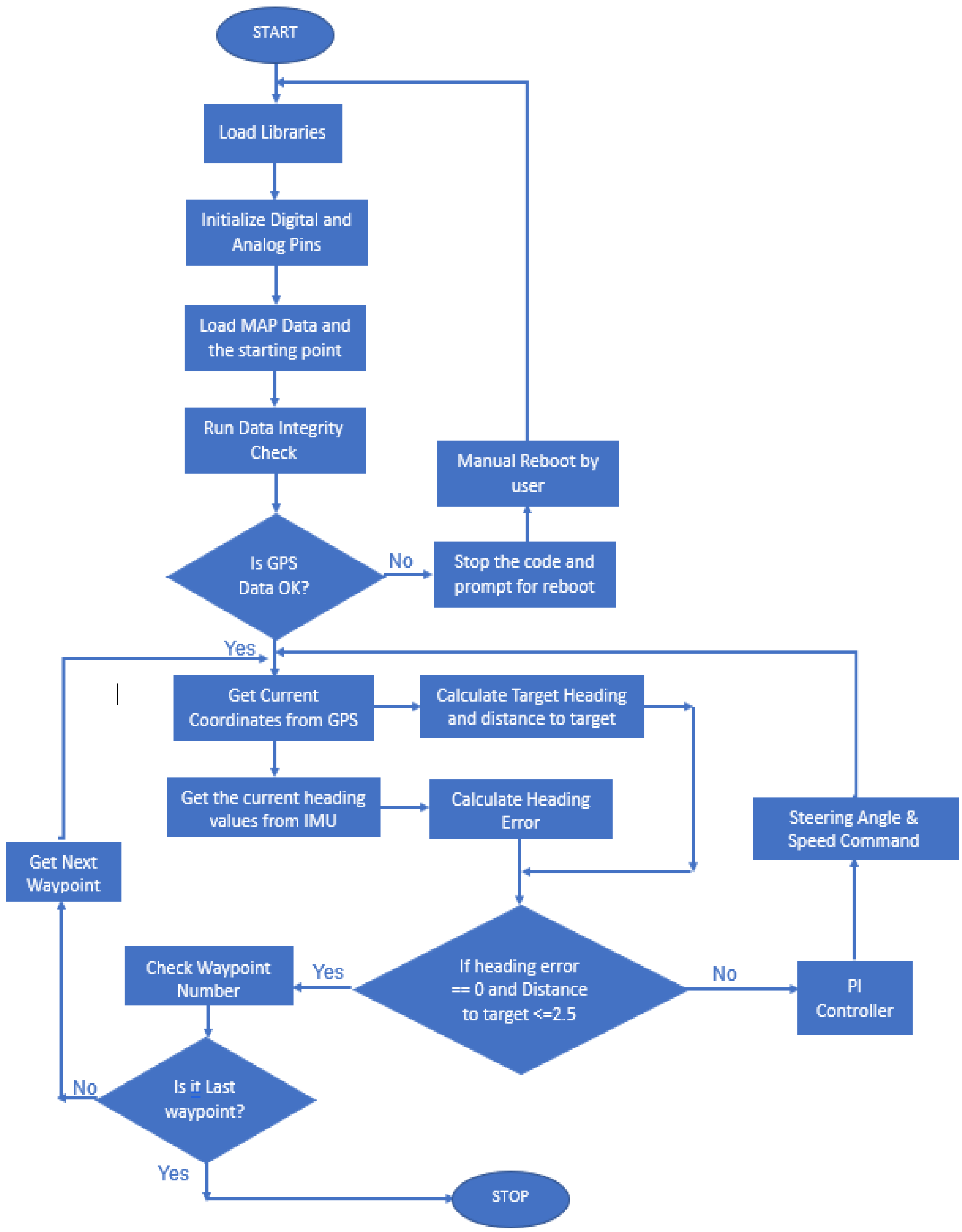

Figure 3-2: Flowchart for Waypoint Navigation

3.2 Selection of Controller Parameters

- PI Controller Gains

  P – Gain = 60/180 = 0.33 ~ **0.4** (Steering Angle / Degree Heading Error (HE)) , where 60° is the maximum possible angle sweep by the wheels and 180° is the maximum possible heading error, assuming the vehicle can take a U – turn.

  I – Gain was set to **0.001** to avoid unstable vehicle performance near waypoints or when the sign of heading error would change.

- Waypoint Tolerance

  It is the distance at which the vehicle stops before the waypoint. This was set to **2 meters** considering the systematic errors in GPS and magnetometer. This gives the controller a tolerance value for stopping around the waypoint.

3.3 Controller Objective

- Minimize the *distance to target*
- Minimize the orientation or *heading error*

3.4 Sensors Used

- Global Positioning System (GPS)
  Specifications are given in appendix section 9.2.2
  Used to give the position feedback in terms of latitude and longitude which is converted to Cartesian coordinate system using the WGS84 model [30].

- Inertial Measurement Unit (IMU)
  Specifications are given in appendix section 9.2.3
  The magnetometer or the digital compass part of the IMU was used to determine the vehicle heading or yaw w.r.t magnetic north.

3.5 Test Results

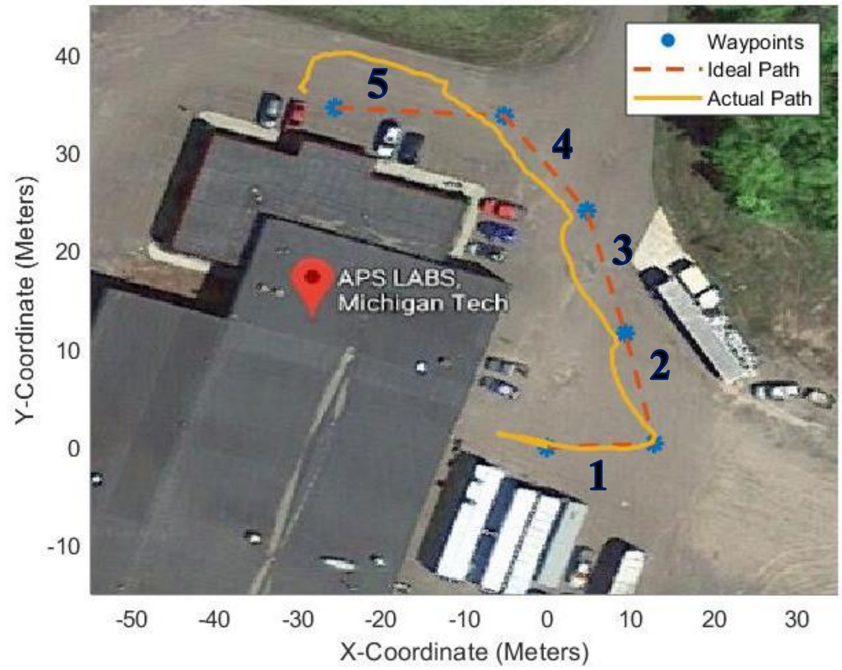

Figure 3-3: Test Result 1 - Comparison between Ideal Path and Actual Path

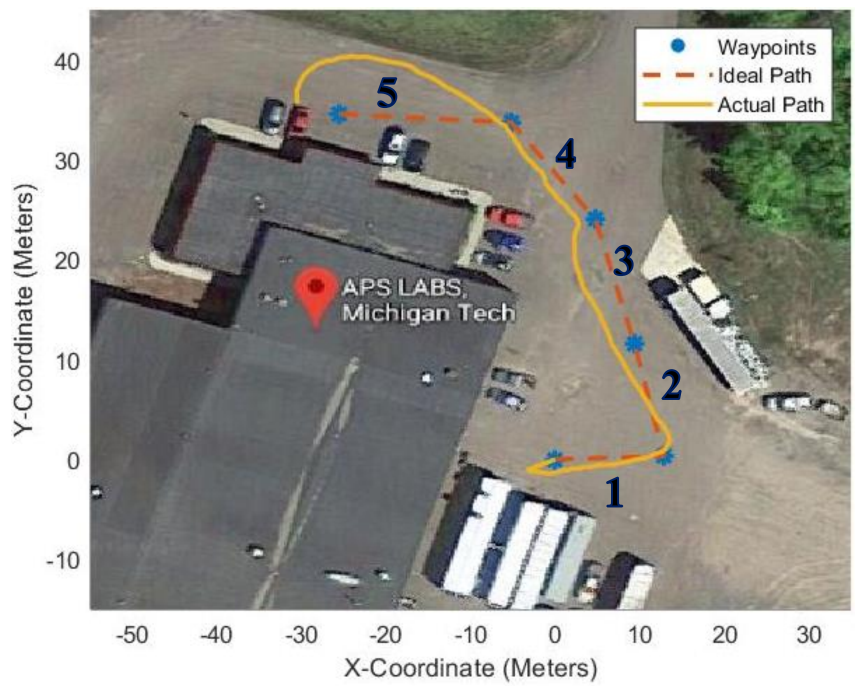

Figure 3-4: Test Result 2 - Comparison between Ideal Path and Actual Path

The path is divided into 5 segments, having a start point followed by 5 waypoints marked in blue as shown in figures 3-3 & 3-4.

3.6 Analysis

From figures 3.2.1 & 3.2.2, it can be clearly seen that the path tracking / waypoint following performance of the vehicle is severely affected by the disturbances in the sensors and data acquisition system. There is an overshoot of approximately 5 meters in segment number 5 of the path. For all the others segments the controller struggles to match with ideal trajectory and seems to have an offset. The bad performance of the steering controller can be attributed to the following factors:

- Controller gains not tuned considering the dynamics of the steering actuator of the vehicle
- Difference in update rates of the GPS @ 5 Hz and Magnetometer @ 10 Hz
- Best possible GPS positional accuracy of around 3 mtrs. as given in appendix section 9.2.2
- Presence of noise and stray magnetic fields affecting Magnetometer performance

All these factors show that there is a need for controller performance analysis for a given vehicle and sensor combination.

4 MODEL BASED CONTROLLER AND SENSOR ANALYSIS

4.1 Selection of Steering Controllers for Analysis

Based on the results in [3], [6] and the previous chapters, it can clearly be observed that from implementation perspective geometric controllers perform better compared to other controllers because of their simplicity and ability to be tuned for every track and velocity conditions. It might also be worth analyzing and tuning PI controller for waypoint navigation as they are simple and versatile when it comes SISO systems. The following controllers were selected for analysis:

- **PI Controller**
  - A closed-loop linear feedback controller used to control the process variable by minimizing the error between the set point and the measured process value.
  - Mathematically, PI control action can be defined as follows:

    $u(t) = \left(K_p * e(t)\right) + (K_i * \int e(t)dt)$ ……………… (1)

    where *u(t)* is the controller output, *e(t)* is error i.e. difference set value and process value, K_p is the proportional gain, K_i is the integral gain and *dt* is the time step.
  - Increasing the proportional gain K_p, increases the output value and vice-versa. Too high proportional gain can make the system unstable or can cause a large overshoot and too low value results in a small output response to a large input error leading to a less sensitive controller. A highly responsive controller is desirable for quick response to changes or disturbances in state. However, it may also be noted that an aggressive controller also responds to the noises in the measured variable. Proportional control action seizes to address the problem of steady state error, since a non – zero error is always needed to generate an output.
  - Integral control allows us to reduce the steady state error since, it is the sum of the instantaneous error over time which accumulates and provides the required control action to reduce the steady state error. A PI controller tends to be less responsive when the sign of the error signal changes due to the previously accumulated error by I control. This is known as Integrator Windup and takes time to unwind. Also, a very high value of K_i can make the system less responsive at start but highly unstable at the end due to accumulated error.

  - Implementation
    1. The controller output will be steering angle used to control the direction or current heading of the vehicle.
    2. The error term will be the difference between the target heading and the current heading.

3. The start and stop of the vehicle will be a rule based controller due to low velocity application.

- **Pure Pursuit Controller**
  - It is a waypoint based proportional controller which assumes a kinematic bicycle model of a vehicle having Ackerman Steering geometry.
  - The control law as mentioned in [3], is given by:

$$\delta = \tan^{-1}\left(\frac{2*L*\sin\alpha(t)}{L_d}\right) \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

where δ is the commanded steering angle, L is the wheelbase of the vehicle, α is the heading error between the vehicle's current heading and the target point heading measured from the vehicle, L_d is the look ahead distance.

  - It can clearly be seen that steering angle is proportional to the heading error w.r.t to the vehicle. Also, the effect of look ahead distance can be illustrated in figure 4-1.

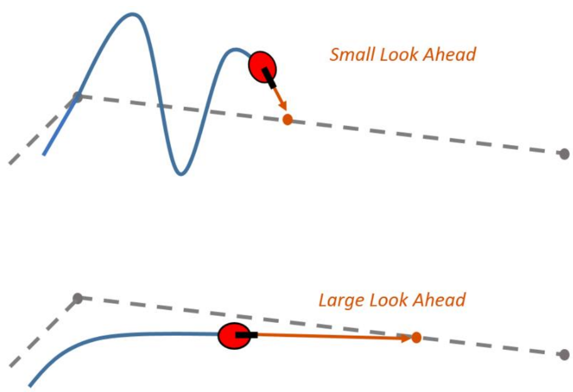

Figure 4-1: Effect of Look Ahead Distance [27]

  - Due to the presence of the Tan inverse function and L_d in the denominator, a small value leads to aggressive steering control which is suitable for making 90 Degrees turns. A large value of L_d leads to smooth control suitable for straight roads or smooth turns but will not be effective in tight corners or sharp turns.
  - The advantage of look ahead distance L_d is that it gives the controller a preview point which is similar to prediction horizon of a Model Predictive Control, thereby allowing the controller to determine the steering angle based on the path dynamics.
  - The obvious disadvantage is that for a given value of L_d, the control action will not be optimal for different road conditions, varying vehicle velocities and different distance to target values.

  - <u>Implementation</u>
    - ➢ The optimal value of L_d will be determined as a function of velocity and distance to target.

- ➢ The start and stop of the vehicle will be a rule-based controller due to low velocity application

- **Stanley Steering Controller**
  - It is a path based non-linear feedback controller. It is developed by Stanford University and used in the DARPA Challenge. This model also assume a bicycle model of the vehicle. As described in [3] and [5], the control law is given by the equation 3 and figure 4-2:

$$\delta = \theta_e + \tan^{-1}\left(\frac{k * e_{fa}(t)}{v_x(t)}\right) \dots\dots\dots\dots\dots\dots (3)$$

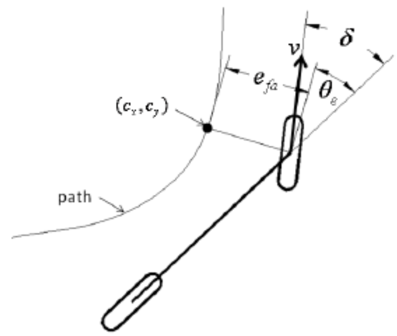

Figure 4-2: Path Parameters for Stanley Controller [3][5]

  where θ_e is the heading error between yaw or vehicle heading and path heading, $e_{fa}(t)$ is the time varying cross track error or lateral path error w.r.t vehicle, $v_x(t)$ time varying longitudinal velocity of the vehicle and k is the controller gain which has the units of sec^{-1} , hence it can assumed to be similar to the time constant of the controller. A high value of k means lower time constant, quick response of the controller and a low value of k means higher time constant, sluggish response of the controller.
  - From the above equation, it can be seen that the Stanley controller is superior to the previous two controllers due to the inclusion of cross track error term. As the vehicle deviates from the path, the cross track error increases creating a steering angle output for the vehicle so as to merge to the path.
  - However, compared to Pure Pursuit Controller, it has more number of inputs, hence, this controller will be more prone to disturbances and noise. Also, the effects of systematic error in GPS will be more evident, since vehicle current position is required for the calculation of the cross track error takes into account the vehicle current position.

  - <u>Implementation</u>
    - ➢ A reference path is generated from the given waypoints and is used to determine the path heading and cross track error.
    - ➢ As mentioned in [3], different gain values are required for different vehicle velocities, hence, the gain will be proportional to Time to Target.

4.2 Modelling approach for sensors, actuators and vehicle kinematics

After the selection of controllers, it was necessary to model the sensors and the actuator dynamics for analysis and tuning of controllers. The sensors can be modelled by the specifications given in the Appendix or by taking real test data for the individual sensors. The second method is chosen since, sensor output depends on the testing and the installation condition of the sensors. Figure 4-3 shows the top level of the model-based approach.

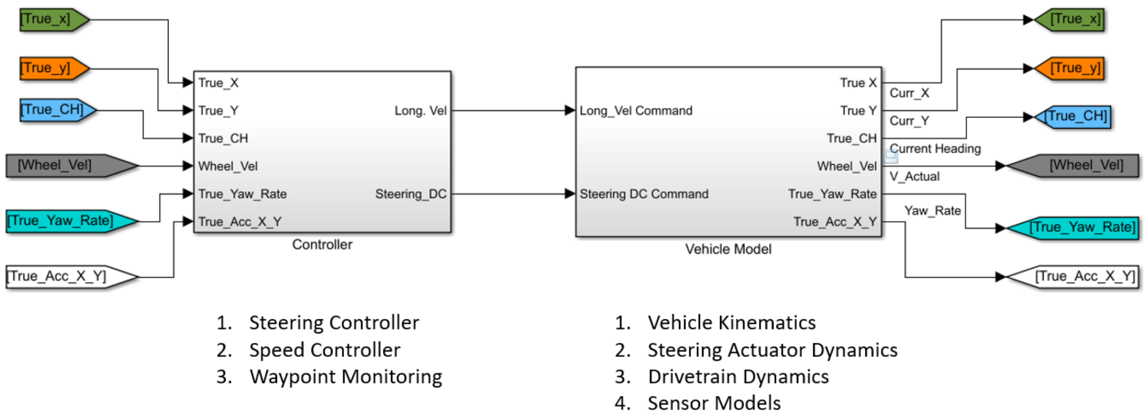

Figure 4-3: Controller and Plant Model for Analysis

4.2.1 Modelling approach for Global Positioning System (GPS)

For obtaining the true values of coordinates X&Y from the GPS, the following time-based model can be used for analysis:

$$J_{ZOH}(t) = ZOH\ (J(t) + b + n)\ \ldots\ldots\ldots\ldots\ldots\ldots\ (4)$$

J(t) represents the ideal and continues time varying values of X&Y coordinates, *b* denotes the bias/systematic error, *n* denotes the noise which is modelled as Gaussian, $J_{ZOH}(t)$ is the discretized value obtained after implementing the zero-order hold function [29] for a sample period of 0.2 seconds / 5 Hertz.

4.2.1.1 GPS Error Analysis

The following test data was taken over a span of 20 minutes at a given position so as to determine the random errors in the GPS. For systematic error, it was assumed that the GPS is operating under WAAS mode and the systematic error in position is 3 meters. This leads to an error of *2.12 m in each x and y coordinates*, since $\sqrt{(0-2.12)^2 + (0-2.12)^2}$ = 3 meters. For GPS, no dynamics were considered as there is no moving element inside the sensor.

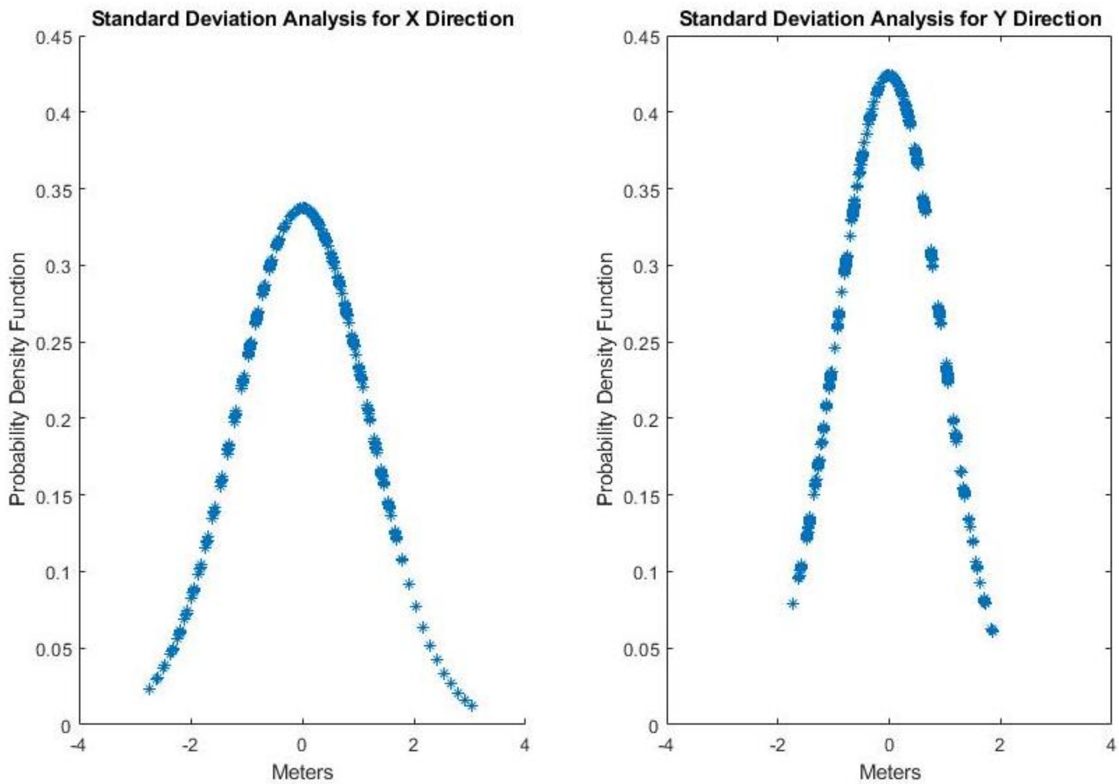

Figure 4-4: Results for Standard Deviation Analysis in X & Y direction GPS sensor modelling

Table 4-1: Sensor Errors for GPS Modelling

Direction	**1σ - Standard Deviation (Meters)**	**Variance in Position (Meters)**	**Systematic Error in Position (Meters)**
x	1.08	1.17	2.12
y	0.94	0.88	2.12

Table 4-1 summarizes the standard deviation values obtained from figure 4-4. A random number generator takes the variance as input for modelling the GPS noise as Gaussian.

4.2.2 Modelling approach for Magnetometer or Digital Compass

The following model is used to determine the true current or vehicle heading values:

$$H_{ZOH}(t) = ZOH(Sat\left(\left(\left(V(t) * Q_{factor}\right) * Q_{conv}\right) + b + n\right)) \quad \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

V(t) is the time varying voltage output from the sensor, Q_{factor} takes into account the quantization factor for the 16-bit ADC, Q_{conv} is conversion factor to convert voltage to degrees, $H_{ZOH}(t)$ is the discretized value obtained after implementing the zero-order hold function [29] for a sample period of 0.1 seconds / 10 Hertz, b and n represent the bias and Gaussian noise. *Sat()* function is used to keep the limit output to the range of 0 to 360 degrees. It is defined as follows:

$$Sat(H) = \begin{cases} H & if\ 0 \le t \le 180, \\ (360 + H) & if - 180 \le t < 0 \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (6)$$

For magnetometer, no dynamics were considered as there is no moving element inside the sensor.

4.2.2.1 Error analysis for Magnetometer or Digital Compass

The North direction reference for measurement was taken with the help of an Analog Magnetic Compass. For systematic error, the Magnetometer was aligned towards the north & the south direction and the average of the errors were taken. For noise, similar to GPS the test data was taken over a span of 20 minutes at the given position and orientation so as to determine the random errors. It was ensured that no stray magnetic field was present.

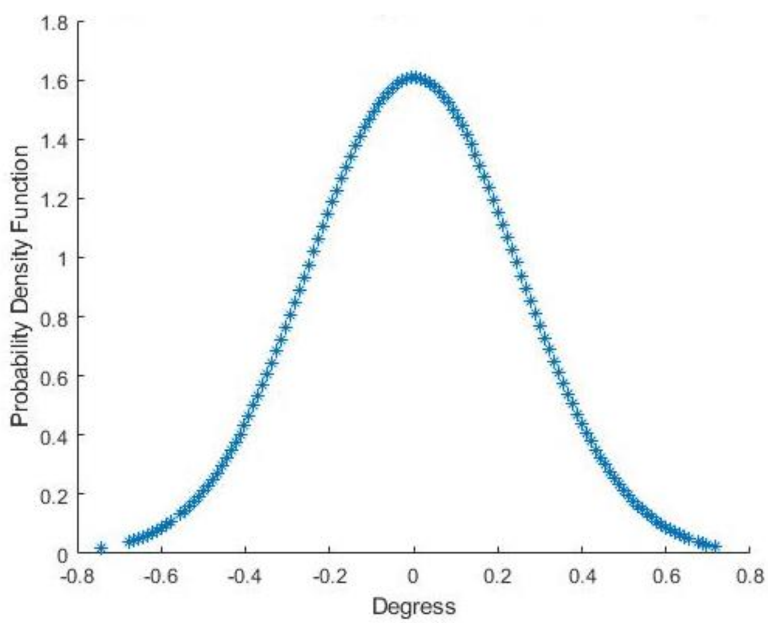

Figure 4-5: Standard Deviation Analysis for Yaw or Current Heading

Table 4-2: Sensor Errors for Magnetometer Modelling

1σ-Standard in Deviation Orientation (Degrees)	**Variance in Orientation (Degrees)**	**Systematic Error (Degrees)**
0.24	0.06	5

Table 4-2 summarizes the standard deviation values obtained from figure 4-5. A random number generator takes the variance as input for modelling the Magnetometer noise as Gaussian.

4.2.3 Modelling approach for IMU (Inertial Measurement Unit)

For modelling the various components of IMU viz. Accelerometer and Gyroscope, the following model was used to determine the true values of acceleration (for accelerometer) and true values of yaw-rate (for gyroscope):

$$J_{ZOH}(t) = ZOH(Sat\left(((L^{-1}\left(H(s) * \left(L\left(V(t) * Q_{factor}\right)\right)\right)) * Q_{conv}) + b + n\right)) \ldots (7)$$

V(t) is the time varying voltage output from the sensor, Q_{factor} takes into account the quantization factor for the 13-bit ADC, *L()* is the Laplace transform to convert t – domain to s- domain, *H(s)* is the transfer function taking into account the MEMS device dynamics, $L^{-1}()$ is the inverse Laplace to convert s-domain to t-domain, Q_{conv} is conversion factor to convert voltage to engineering units, $J_{ZOH}(t)$ is the discretized value obtained after implementing the zero-order hold function [29] for a sample period of 0.1 seconds / 10 Hertz, *b* and *n* represent the bias and Gaussian noise.

Sat() function is used to limit output of accelerometer and gyroscope as per the specifications in Appendix 9.2.2. It is defined as follows:

For Accelerometer,

$$Sat(Acc) = \begin{cases} -78.48\frac{m}{s^2} & if\ Acc < -78.48, \\ Acc & if -78.48 \le\ Acc \le 78.48, \\ 78.48\frac{m}{s^2} & if\ Acc > 78.48 \end{cases} \quad \ldots\ldots\ldots (8)$$

For Gyroscope (Yaw-Rate),

$$Sat(YR) = \begin{cases} -2000\ dps & if\ YR < -2000, \\ YR & if -2000 \le\ YR \le 2000, \\ 2000\ dps & if\ YR > 2000 \end{cases} \quad \ldots\ldots\ldots\ldots (9)$$

4.2.3.1 Error analysis for Accelerometer

Experiment 1: Standard Deviation Analysis

For the accelerometer, the test was done on a flat surface for accelerations in the x & y directions. The flatness of the surface was ensured by a spirit level. The data was recorded for a span of 20 minutes without changing the orientation. Since, it is a MEMS device, there will also be a transfer function associated along with systematic error and noise.

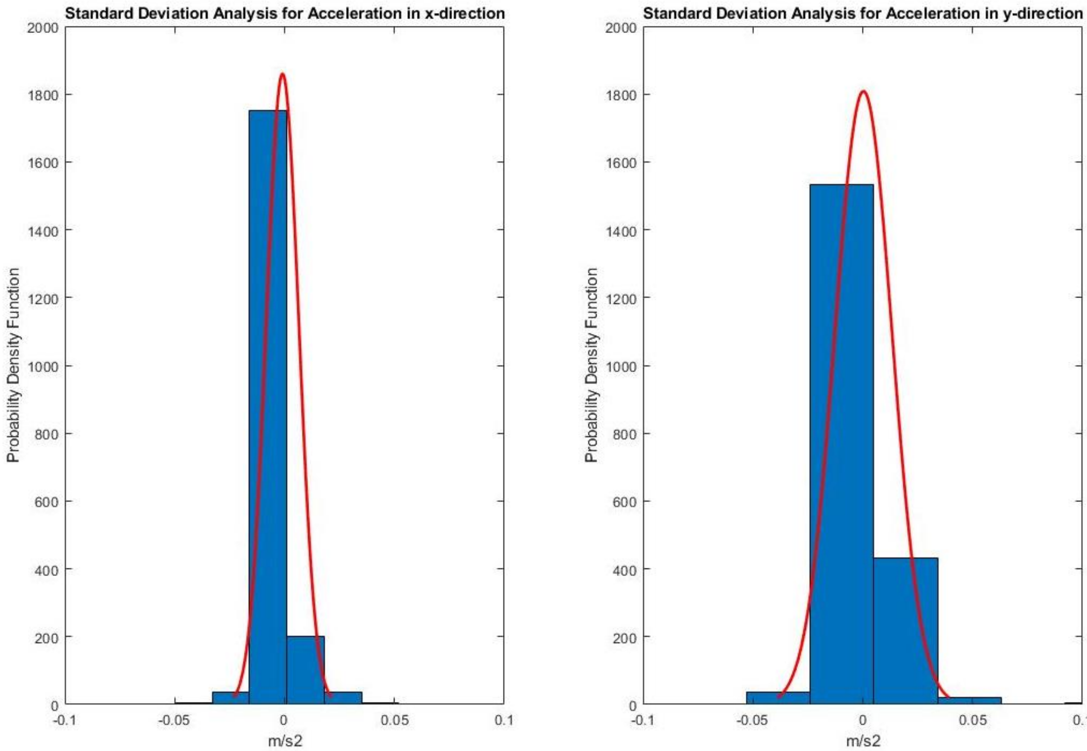

Figure 4-6: Standard Deviation Analysis of Accelerometer in X & Y Direction

Table 4-3: Sensor Errors for Accelerometer Modelling

Direction	**1σ-Standard Deviation (m/s^2)**	**Variance (Degrees) (m/s^2)**	**Systematic Error (m/s^2)**
X	0.074	0.0055	0.0099
Y	0.013	0.0002	0.0099

Table 4-3 summarizes the standard deviation values obtained from figure 4-6. It should also be noted that compared to GPS the systematic errors are negligible. The variance calculated was used for modelling the sensor noise as Gaussian.

<u>Experiment 2: Transfer Function Derivation</u>

In order to model the transfer function of accelerometer, the vehicle was commanded to move on a straight path at a constant speed of 1m/s. The acceleration plot from the test was used as basis for deriving the transfer function. The ideal accelerometer characteristics were approximated by the fact that, initially when the vehicle launches it will have maximum acceleration and while braking maximum deceleration. An input of ideal data was given, and the simulation output data was compared with real test data. Figure 4-7 shows a comparison between ideal, actual and simulated values of acceleration.

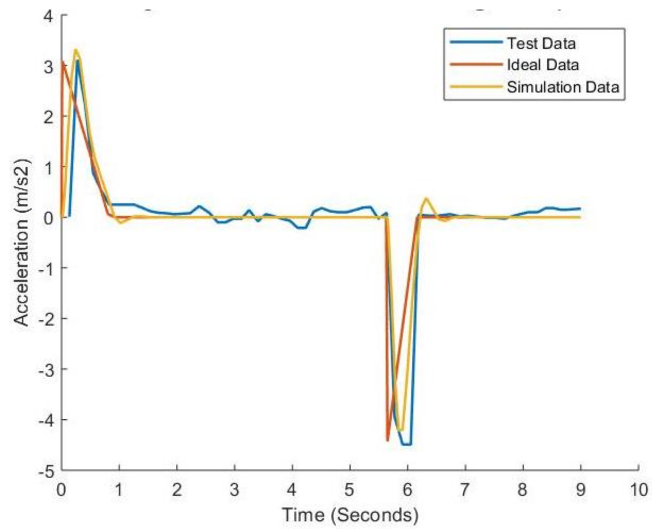

Figure 4-7: Straight Line Test @ 1m/s for Transfer Function Generation

The Transfer function H(s) for accelerometer was found out to be:

$$H(s) = \frac{1.08}{0.007s^2+0.075s+1} \quad \ldots\ldots\ldots\ldots\ldots\ldots (6)$$

It should be noted the above transfer function is that of the accelerometer on vehicle and other high-resolution methods are needed to separately derive the transfer function of the vehicle.

4.2.3.2 Error analysis for Gyroscope Yaw-Rate

The Gyroscope is similar to accelerometer since, it is also a MEMS device. Hence, it will also have a transfer function along with systematic error and noise.

<u>Experiment 1: Standard Deviation Analysis</u>

Similar to the process of accelerometer, the IMU sensor was place on a flat surface and the data was recorded for a span of 20 minutes without changing the orientation.

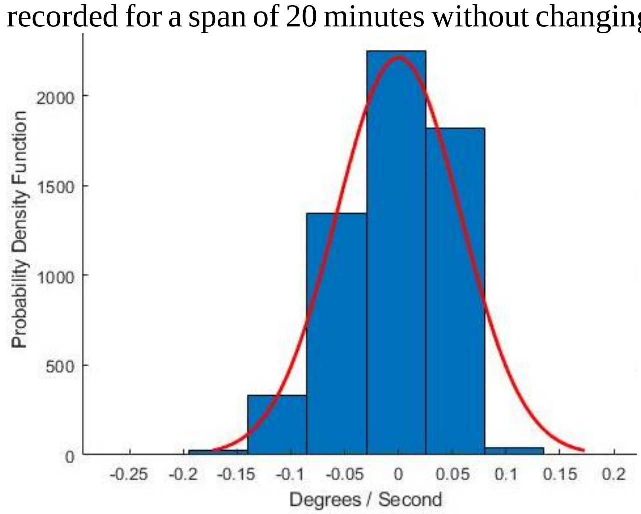

Figure 4-8: Standard Deviation Analysis for Gyroscope Yaw-Rate

Table 4-4: Sensor Errors for Gyroscope Yaw-Rate Modelling

1σ- Standard in Deviation Orientation (Degrees / Sec)	Variance in Orientation (Degrees / Sec)	Systematic Error (Degrees / Sec)
0.058	0.003	-0.064

Table 4-4 summarizes the standard deviation values obtained from figure 4-8, from which variance was obtained for modelling sensor noise as Gaussian.

Experiment 2: Transfer Function Derivation

In order to model the transfer function, the vehicle was tested on a circular path of diameter 4 meters at a constant steering angle and at a constant velocity of 1m/s.

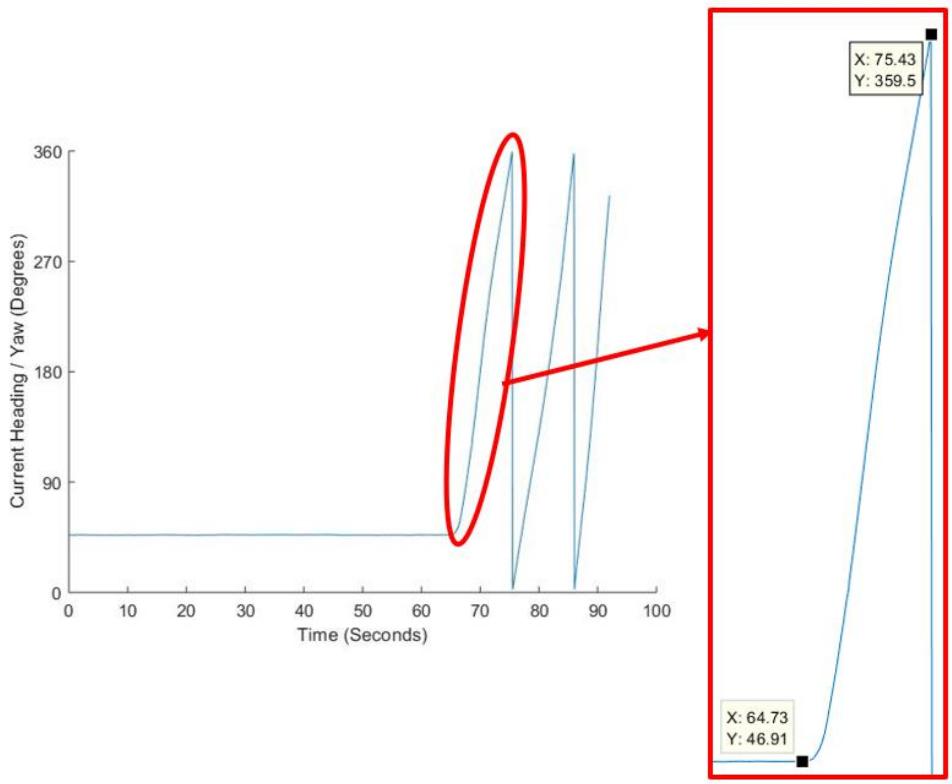

Figure 4-9: Circle Test Results

From figure 4-9, it can be seen that when the vehicle travels in a circle i.e. North East, South and West, the yaw or current heading values go up to 360 Degrees and then again comes down to zero. For above highlighted portion, the slope is constant and it can be assumed that the yaw rate is constant, which can be calculated as follows.

$$Theoritical\ Yawrate = \frac{359.5-46.9}{75.43-64.73} = 29.22\ deg/s \ \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

Figure 4-10 gives the measured value of Yaw – Rate from the gyroscope:

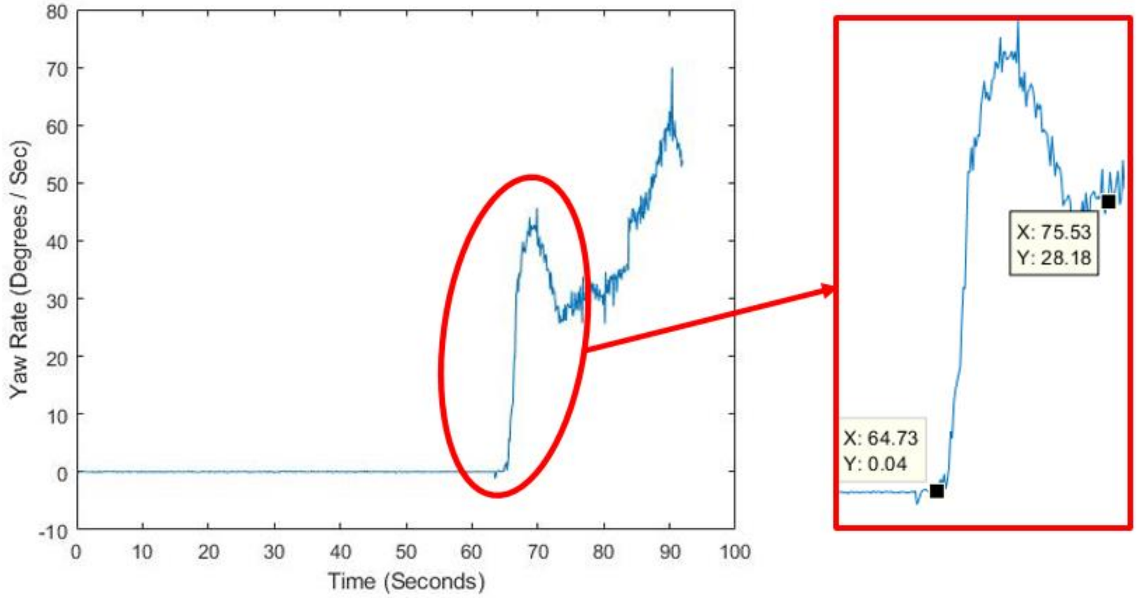

Figure 4-10: Gyroscope (Yaw-Rate) Output for Circle Test

The theoretical Yaw – Rate obtained from equation (7) was used as ideal yaw rate and was modelled as a step function and the simulation output data was compared with real test data. Figure 4-11 shows a comparison between ideal, actual and test values of yaw-rate.

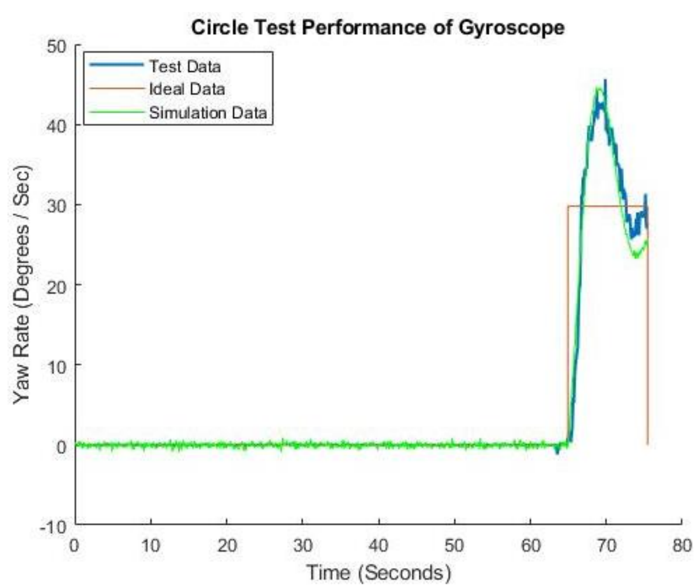

Figure 4-11: Analysis of Test Data and Simulation Data for Transfer Function Generation for Gyroscope

The Transfer function H(s) for the gyroscope was found out to be:

$$H(s) = \frac{0.8s+1.005}{2.3s^2+0.8s+1} \quad \text{.................. (8)}$$

4.2.4 Modelling approach for Wheel Speed Sensor

The wheel speed sensor is modelled similar to GPS with a sample period of 0.1 seconds / 10 Hertz. The wheel speed sensor is a variable reluctance type sensor and can be simply modelled as having noise and zero systematic error. The noise in the sensor can be due to the presence of residual magnetic field. Table 4-5 summarizes the sensor errors. Due to technical reasons, the standard deviation analysis of wheel speed sensor could not be performed.

Table 4-5: Sensor Errors for Wheel Speed Sensor Modelling

1σ - Standard Deviation in Speed (m/s)	Variance in Speed (m/s)	Systematic Error (m/s)
0.02	0.0004	0

4.2.5 Steering System Actuator

The steering actuator is a servo motor with a reduction gear ratio of 3:1, position of which is controlled by PWM signals from the controller.

Prior to developing the actuator model, the operating range of the actuator duty cycle was found using the methods described in figure 4-12 and 4-13, having 2 stages – decoding and verification. In order to determine the maximum range of steering angle and duty cycle range, the vehicle was suspended in air.

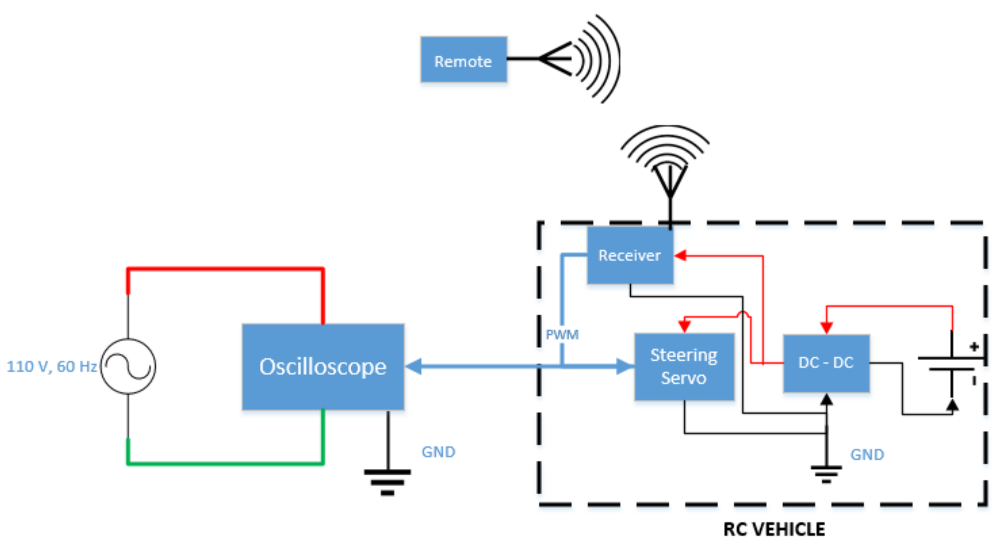

Figure 4-12: Schematic - Steering System Duty Cycle Decoding Process

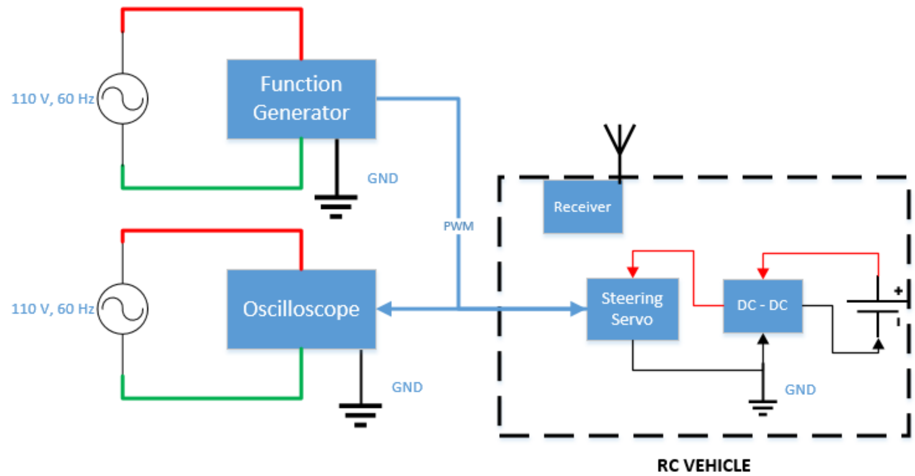

Figure 4-13: Schematic - Steering System Duty Cycle Verification Process

A linear map between steering angle (SA) and duty cycle (DC) was created, which is given by:

$$DC = (0.1667 * (SA)) + 26.067 \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(9)}$$

where a steering angle of -30 degrees corresponds to 21% DC and steering angle of 30 degrees corresponds to 31% DC. The steering system can be modelled as follows:

- Having <u>hysteresis</u> based on a certain road condition, i.e. for a commanded steering angle the actuator does not move exactly by that angle. The hysteresis was measured on asphalt road and table 4-6 was derived:

Table 4-6: Commanded vs Actual Steering Angle

Direction	Commanded Steering Angle	Commanded Duty Cycle (DC)	Actual Steering Angle (SA)
Left to Right	30.0	31.0	24.0
	0.0	26.2	-2.0
	-30.0	21.0	-30.0
Right to Left	-30.0	21.0	-27.0
	0.0	26.2	1.5
	30.0	31.0	28.0

From Table 4-6, the linear relation between duty-cycle and actual steering angle is given as follows:

$$SA_{L\text{-}R} = (5.4502*DC) - 144.73 \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(10)}$$

$$SA_{R\text{-}L} = (5.4518*DC) - 141.28 \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(11)}$$

- Having a <u>delay</u> instead of instantaneous response. This can be modelled as a first order transfer function. From appendix, Section 9.2.3, a 60° sweep of wheels, takes around 0.27 seconds. Hence, the ideal response could be modelled as a step function from 0 to 60° at a given instant, as shown in figure 4-14.

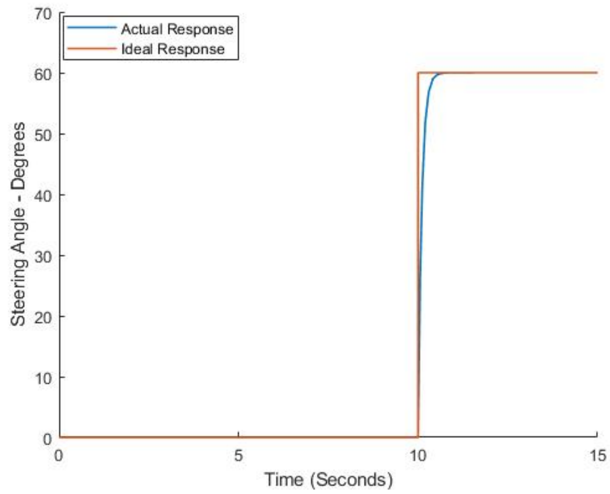

Figure 4-14: Actual Response vs. Ideal Response Analysis for Transfer Function

It can be seen the actual response reaches 58° SA at around 10.27 secs. The transfer function for the steering system is given by:

$$H(s) = \frac{1}{0.08s+1} \quad \ldots\ldots\ldots\ldots\ldots\ldots (12)$$

From equations 10 and 11, the actuator hysteresis implementation is described in the following flowchart:

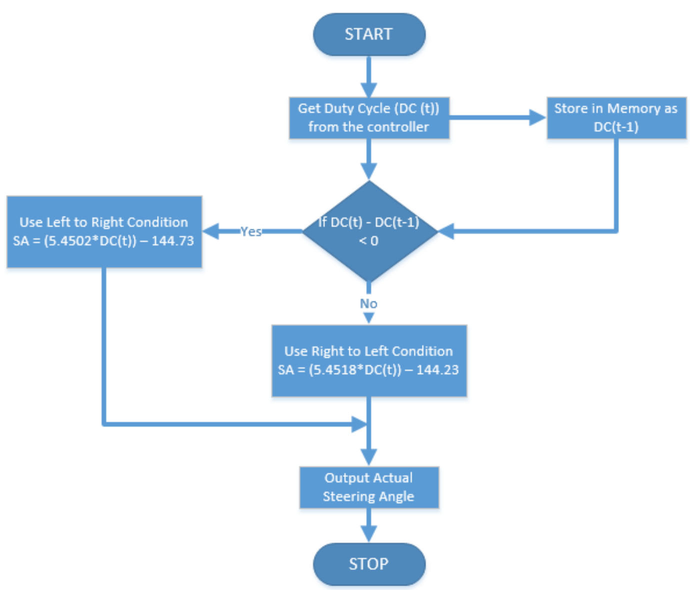

Figure 4-15: Actuator Hysteresis Modelling

4.2.6 Modelling the Drivetrain

Since, the entire analysis is being done on a RC vehicle and at low speeds, the dynamics can be modelled as a single order system with very fast dynamics shown in figure 4-16.

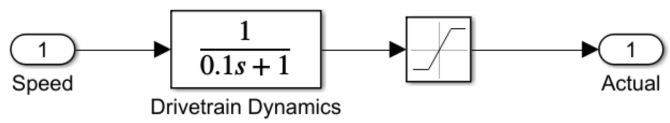

Figure 4-16: Drivetrain Dynamics

4.2.7 Modelling the Vehicle Kinematics

As described in [28], a vehicle moving with low speed and having Ackermann steering geometry can be approximated as two – wheeled model / bicycle model with zero slip angle. Figure 4-17 shows the top-level view of the kinematics model.

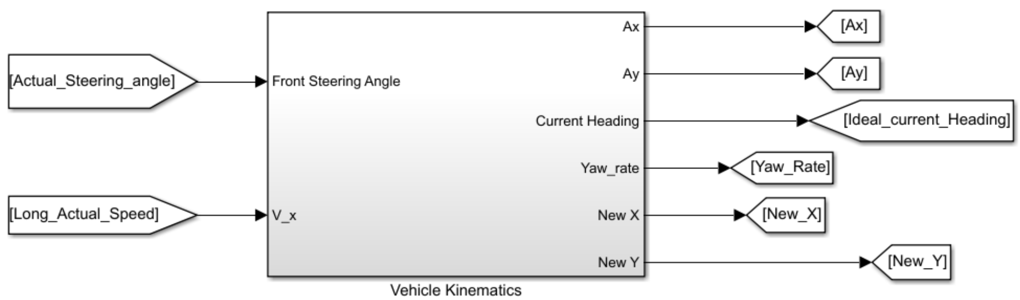

Figure 4-17: Vehicle Kinematics Model

For the given values of steering angle and longitudinal speed, inverse kinematics equations for front steered vehicle can be used to determine the angular speed of front and rear wheels by the following equations:

$w_f = \frac{V}{R*cos(\delta)}$, and $w_r = \frac{V}{R}$, where w_f and w_r are angular velocities of front and rear wheels, V is the longitudinal velocity, R is the wheel radius and δ is the steering angle of the vehicle.

After obtaining, front and rear angular velocities, the forward kinematics equations can be used to determine the velocities in longitudinal and lateral direction of the vehicle along with angular rotation about the vehicle’s perpendicular axis. This is shown in figure 4-18.

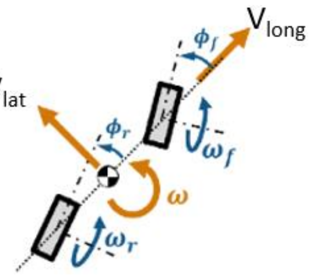

Figure 4.2.8.2: Representation of Linear and Angular Velocities [28]

$$V_{long} = \frac{R(w_f \cos(\emptyset_f) + w_r)}{2} \text{.................. (13)}$$

$$V_{lat} = \frac{R\, w_f \sin(\emptyset_f)}{2} \text{.................. (14)}$$

$$w = \frac{R\, w_f\, sin(\emptyset_f)}{L} \text{.................. (15)}$$

All these parameters can be represented graphically on the X & Y plane by the below figure, where θ is the Yaw or Current Heading w.r.t the X- Axis

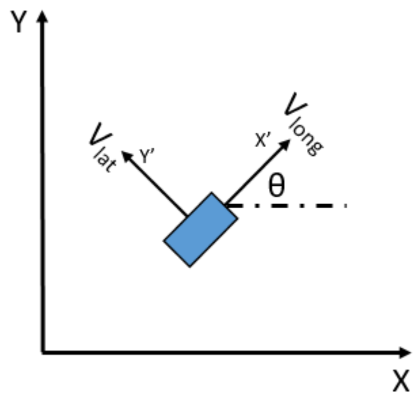

Figure 4.2.8.3: Graphical Representation of Vehicle in 2D Cartesian Coordinates

The above velocities are in the vehicle frame of reference (X' & Y') and in order to determine the position of the vehicle in the reference coordinate system (X & Y), the components of the velocities have to be resolved in both X & Y direction.

$$V_x = V_{long}\, cos(\theta) - V_{lat}\, sin(\theta) \text{.................. (16)}$$

$$V_y = V_{long}\, sin(\theta) + V_{lat}\, cos(\theta) \text{.................. (17)}$$

Integrating these equations gives us the X & Y coordinate of the vehicle at each time step. Differentiating these velocities gives us the acceleration of the vehicle w.r.t reference coordinates. Similarly,$Yaw_{final} = Yaw_{inital} + \int_0^{dt} w\, dt$, followed by conversion from radians to degrees.

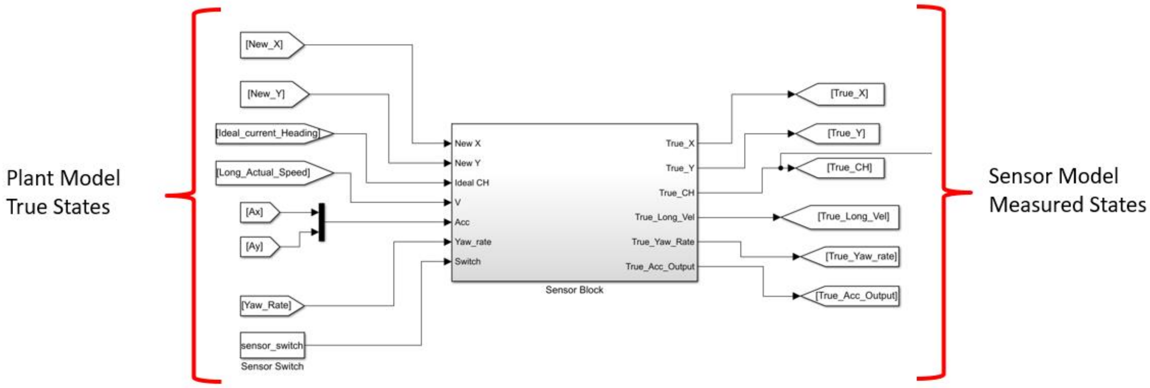

Figure 4-18: Interfacing of Vehicle States with Sensor Blocks

Figure 4-18 shows the ideal plant (vehicle) model states are being passed through the sensor block which adds the errors mentioned in section 4.2.1, 4.2.2 and 4.2.3, thereby representing the measured states or the sensor model data.

4.2.8 Controller Modelling

Controller modelling can be classified in the following parts:

- Speed Controller
- Navigation Monitoring
- Waypoint Monitoring
- Steering Controller

4.2.8.1 Speed Controller

Figure 4-19, shows a flowchart of the start-stop type speed control

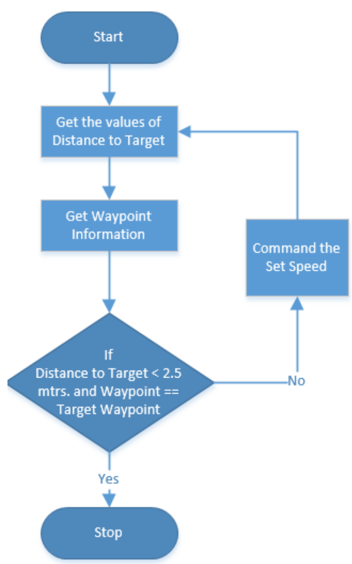

Figure 4-19: Start Stop type Speed Control

4.2.8.2 Navigation Monitoring

This subsystem takes the current position and orientation of the vehicle and generates the target heading, path heading, and distance to target values and cross track-error.

The distance to target is calculated by the formula,

$$d = \sqrt{(x_t - x_c)^2 + (y_t - y_c)^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots (18)$$

where x_t y_t are the target points and x_c y_c are the current vehicle coordinates.

In order to calculate the cross-track error (XTE), it is assumed that the path between waypoints is a straight line. For a curved path, like real road conditions, it will consist of multiple points and it can be linearized for every two consecutive points to obtain a straight line. The XTE is calculated by establishing a relation between a point and a straight line and then finding the shortest perpendicular distance by the following relation:

$$XTE = \frac{ax_c + by_c + c}{\sqrt{a^2 + b^2}} \quad \ldots\ldots\ldots\ldots\ldots\ldots (19)$$

Where a, b, c are the coefficients of the equation of the straight line given by ax + by +c =0 between two path points x_1 y_1 and x_2 y_2. The following logic table is used to determine the sign of the XTE:

Table 4-7: Sign Convention for Cross Track Error

Nature of Slope	Position of Point	Sign of XTE
+	Above Line	-
+	Below Line	+
-	Above Line / Right Side	+
-	Below Line / Left Side	-

For calculating the target heading for waypoint-based controllers, the current heading, current vehicle position and target points are used. The trigonometric block *atan2* has a range of (-Π to Π) radians. The flowchart in the figure 4-20 is used to determine the target heading for waypoint-based controllers:

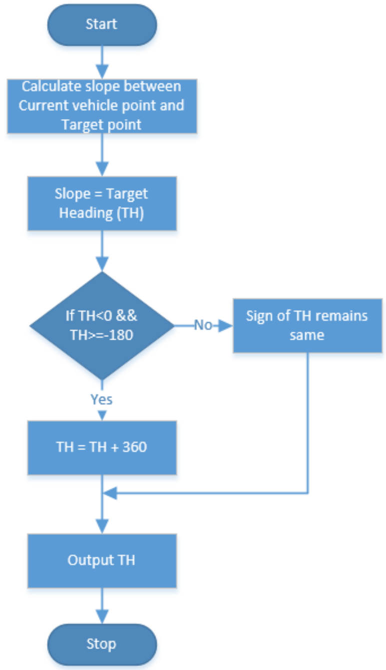

Figure 4-20: Flowchart to Determine Target Heading for Waypoint Based Controllers

For calculating the path heading for Stanley controller, the slope between current target point and previous target point is calculated as shown in figure 4-21.

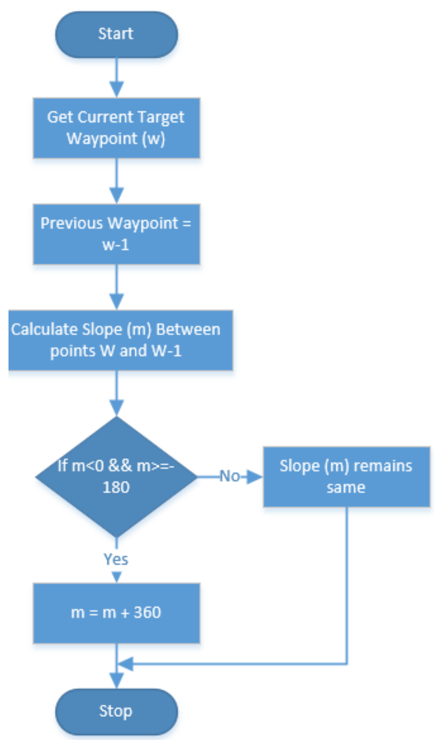

Figure 4-21: Flowchart to Determine Path Heading for Stanley Controller

Waypoint Monitoring

For waypoint monitoring, *distance to target* and *current waypoint number* variable were taken as inputs. The following flowchart shows logic for waypoint monitoring:

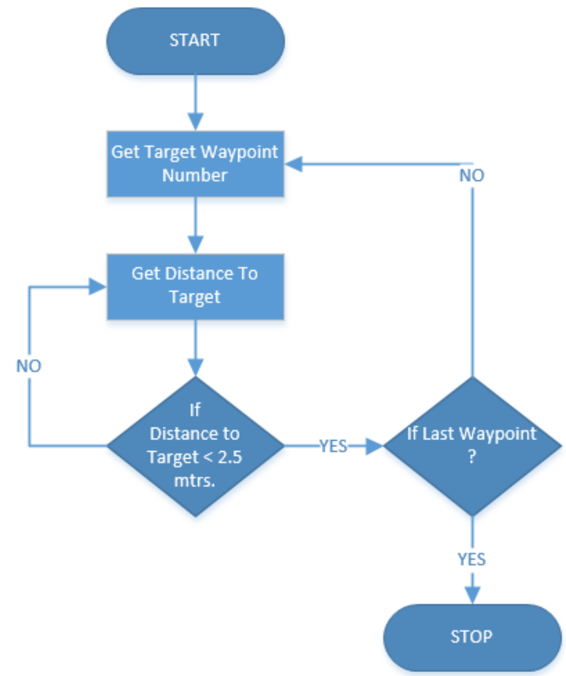

Figure 4.2.9.3.1: Flowchart Logic for Waypoint Monitoring

4.2.8.3 Steering Controller

- <u>PI Controller</u>
  As discussed in section 4.1, the controller is modelled as follows were $K_p = 1$ *and* $K_i = 0.3$ are determined by Ziegler–Nichols method [17].

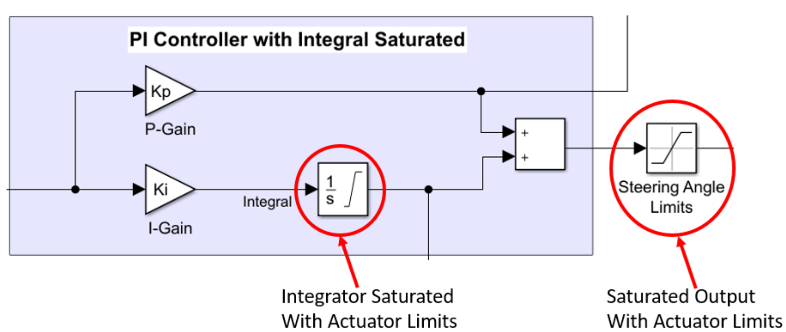

Figure 4-22: PI Controller Implementation

As shown in figure 4-22, the integrator and the final output of the controller is saturated or limited by the physical limits of the steering actuator.

- <u>Pure Pursuit Controller</u>

As discussed in Section 4.1, for different velocities and road geometry, the look ahead distance or the preview distance changes. This directly affects the response of the controller. Based on the required simulation conditions the look-ahead distance was defined as a function of velocity and distance to target given by the following relation:

$$L_d = \frac{1}{\frac{w_1}{D2T} + \frac{w_2 * \Delta t}{v}} \quad \text{.................} \; (20)$$

Where *Δt* is the time-step, w_1 & w_2 are the weights associated *distance to target* & *velocity (v)* term. It is also worth noticing that for a given constant velocity, as the distance to target (D2T) of the vehicle decreases the *Tan Inverse* yields a very high steering angle. Also, giving a very high weightage to the *velocity (v)* term decreases the steering performance at different turns and waypoint due to constant value of look ahead distance. Based on trial and error, the weights $w_1 = 0.4$ &$w_2 = 0.6$. Based on the scope of simulation, the look ahead distance was given a saturation limit of 0.7 to 1.2 was used for optimal performance.

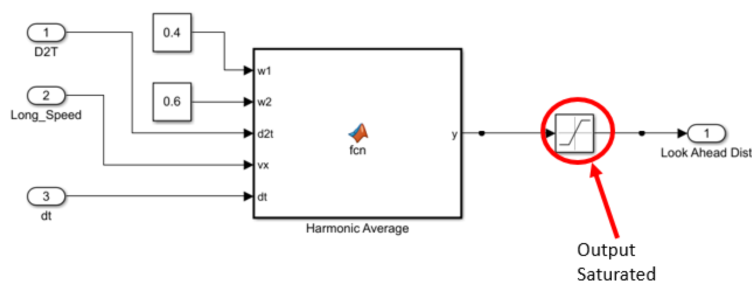

Figure 4-23: Look Ahead Distance for Pure Pursuit Controller

Figure 4-23 shows the implementation of equation 20 for the calculation of the look ahead distance.
Figure 4-24 shows the integration of look ahead distance calculator with the steering controller.

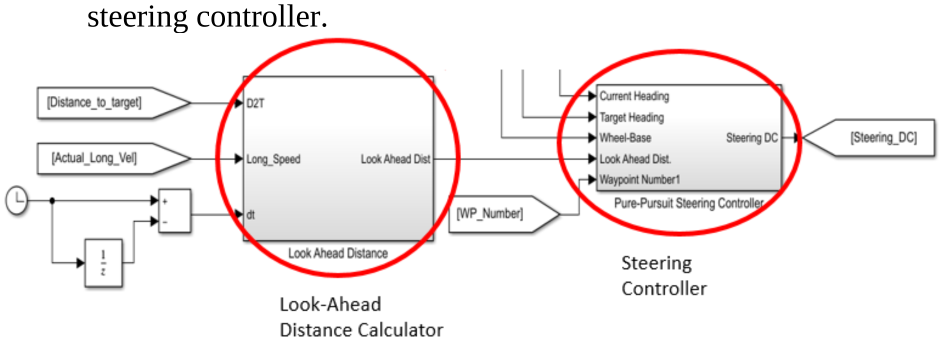

Figure 4-24: Pure-Pursuit Controller Implementation

- <u>Stanley Steering Controller</u>

  Similar to the Pure-Pursuit controller, the gain values are different for different vehicle velocities and for different path geometries. The units of gain is Sec^{-1} hence, it might be more effective to implement the gain as time to target where, Gain $k = 1 / T$, where T = *Time to Target* given by $D2T / v$. Here, D2T is the distance to target of the vehicle from the waypoint and v is the vehicle speed. Again, based on the scope of simulation, a saturation limit of 2 to 4 was used for optimal performance.

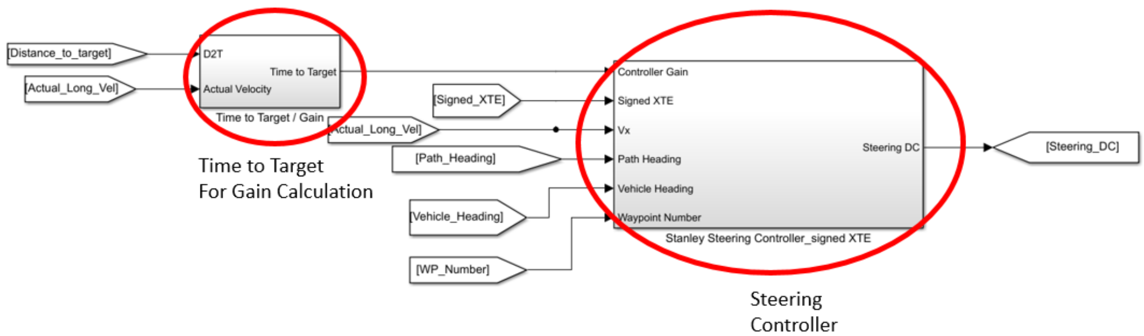

Figure 4-25: Stanley Controller Implementation

4.2.9 Test Cases for Controllers

All the 3 controllers were tested under 3 different path conditions, namely:

- Custom Path (Figure 4-26)
- Straight Path (Figure 4-27)
- Dynamic Lane Change (Figure 4-28)

The following sensors were used for the simulation:

- GPS – X&Y coordinates
- Magnetometer or Digital Compass – Vehicle Heading

The performance of each controller is based on the following metrics:

- Vehicle Trajectory
- Cross – Track Error (XTE) or Lateral Distance for 2D condition
- Distance between vehicle stop point and actual waypoint under 1D condition
- Controller Response for Location Specific Noise

Path 1:

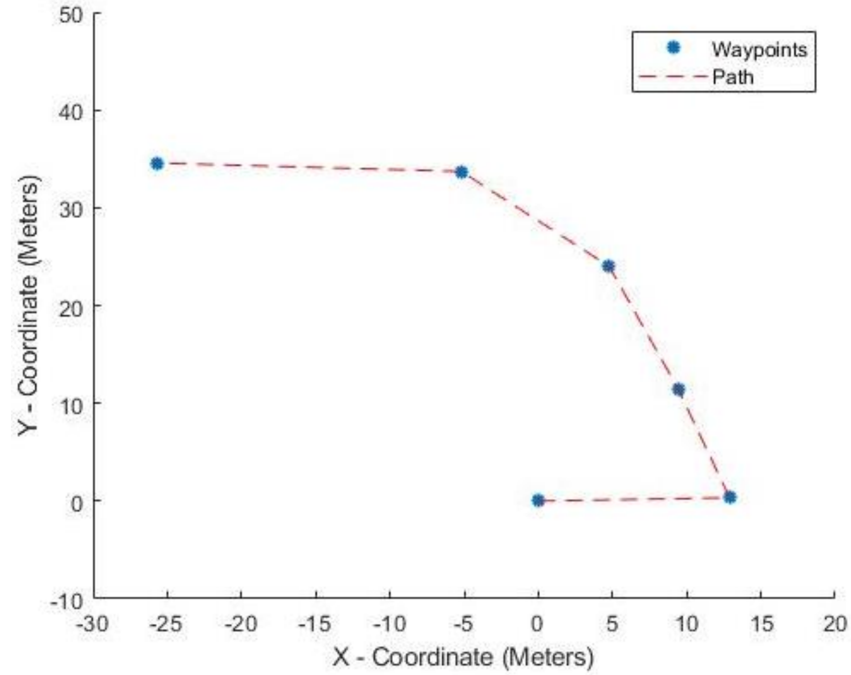

Figure 4-26: Custom Path

Path 2:

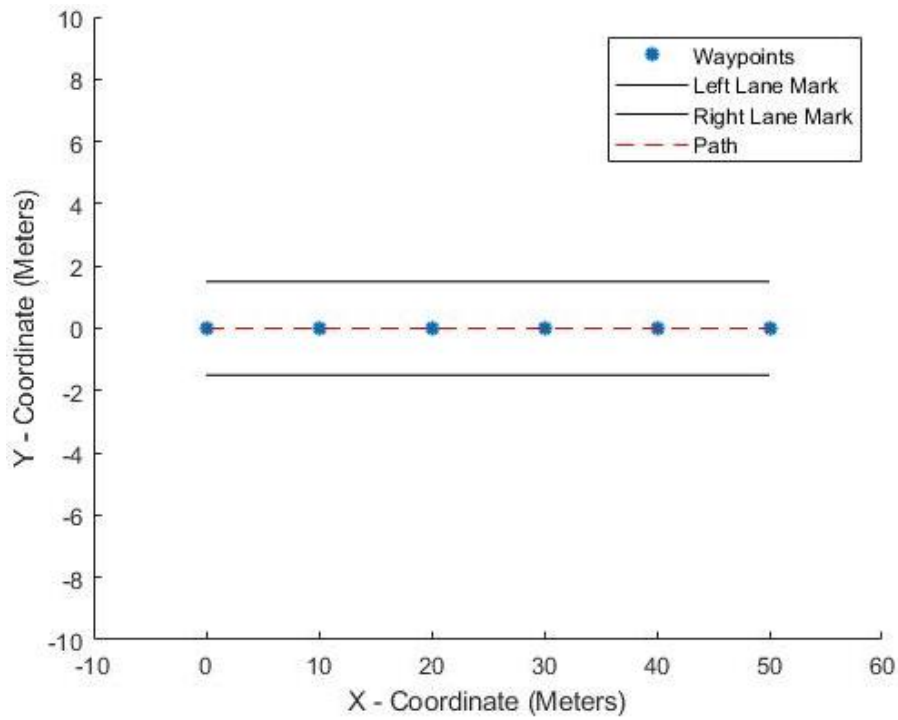

Figure 4-27: Straight Path

Path 3:

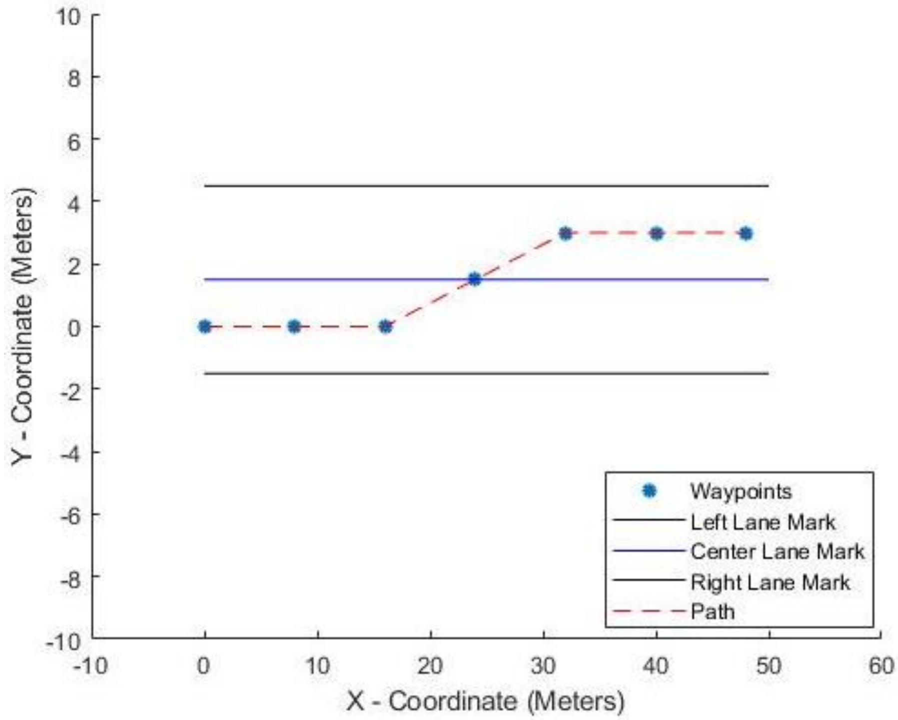

Figure 4-28: Dynamic Lane Change

5 SIMULATION RESULTS AND ANALYSIS

5.1 Performance Analysis under Ideal Sensor Conditions

5.1.1 Custom Path

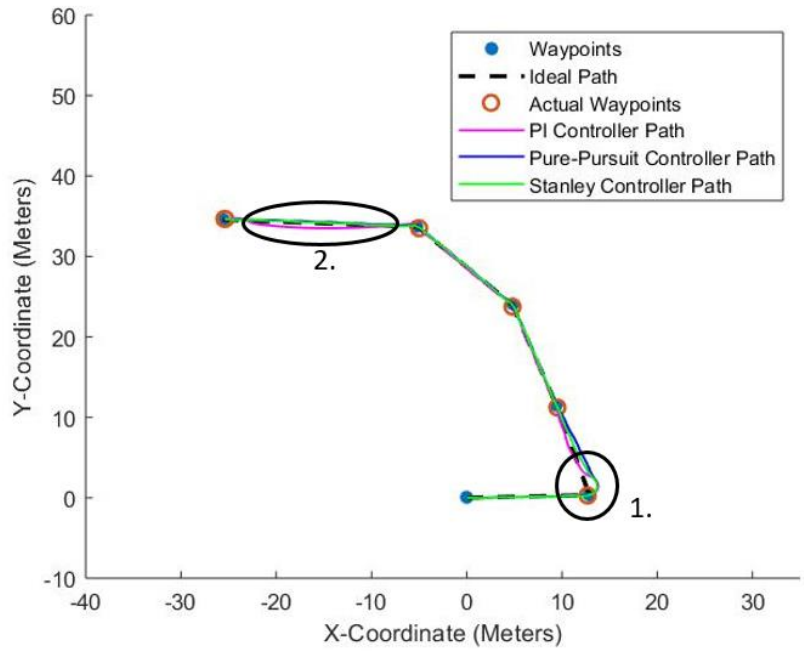

Figure 5-1: Path Tracking Performance of Controllers

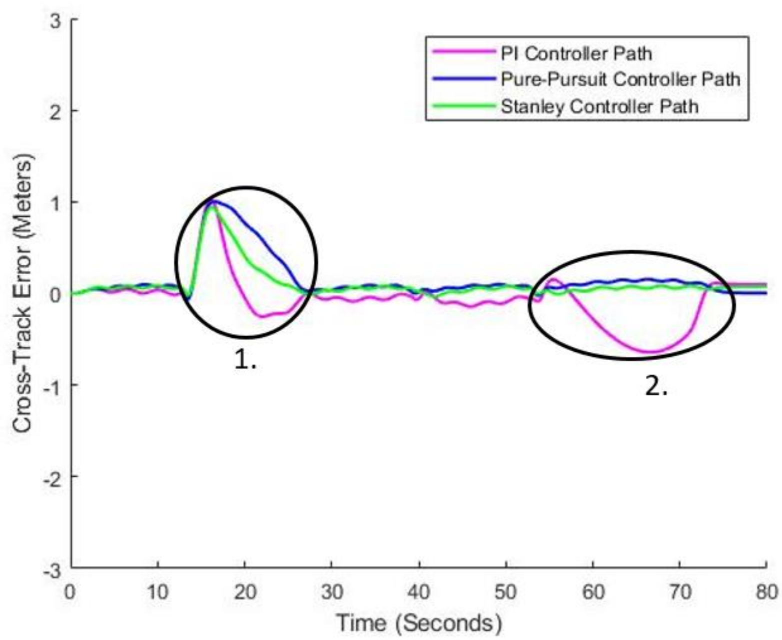

Figure 5-2: Cross Track Error of Vehicle on Custom Path

5.1.2 Straight Line

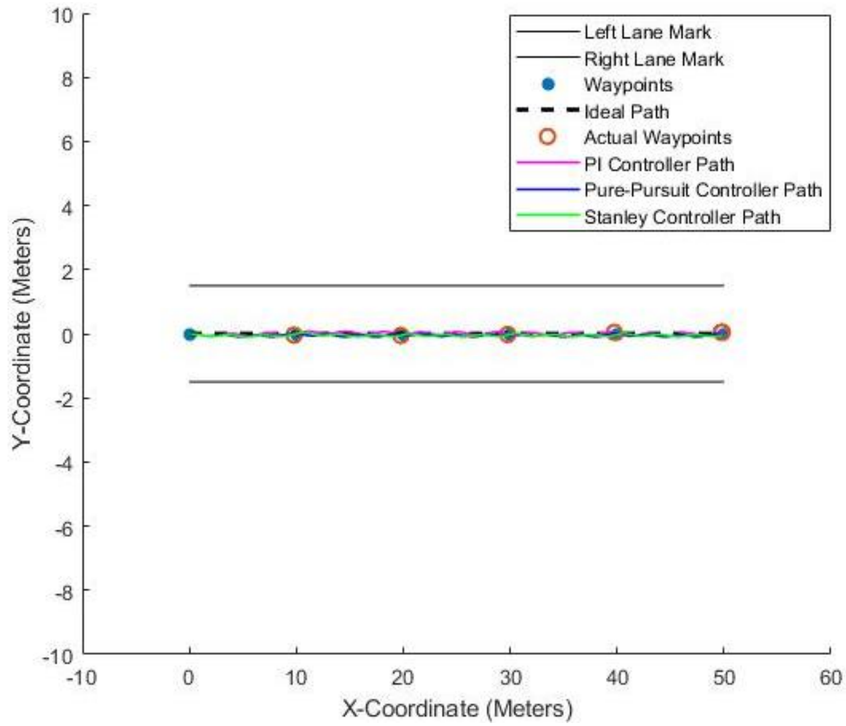

Figure 5-3: Path Tracking Performance of Controllers on Straight Path

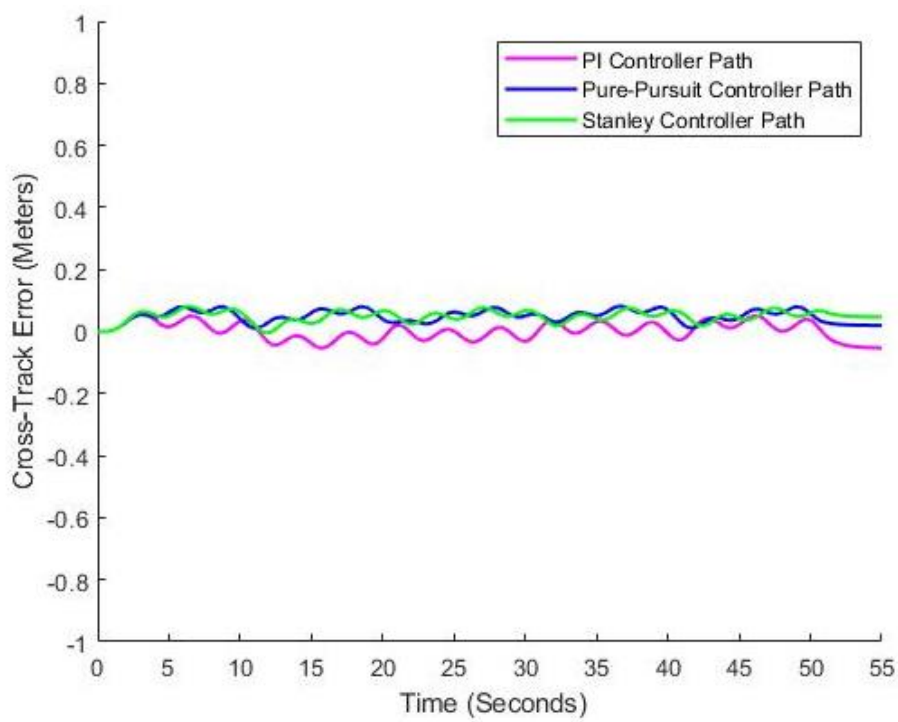

Figure 5-4: Cross Track Error of Vehicle on Straight Path

5.1.3 Dynamic Lane Change

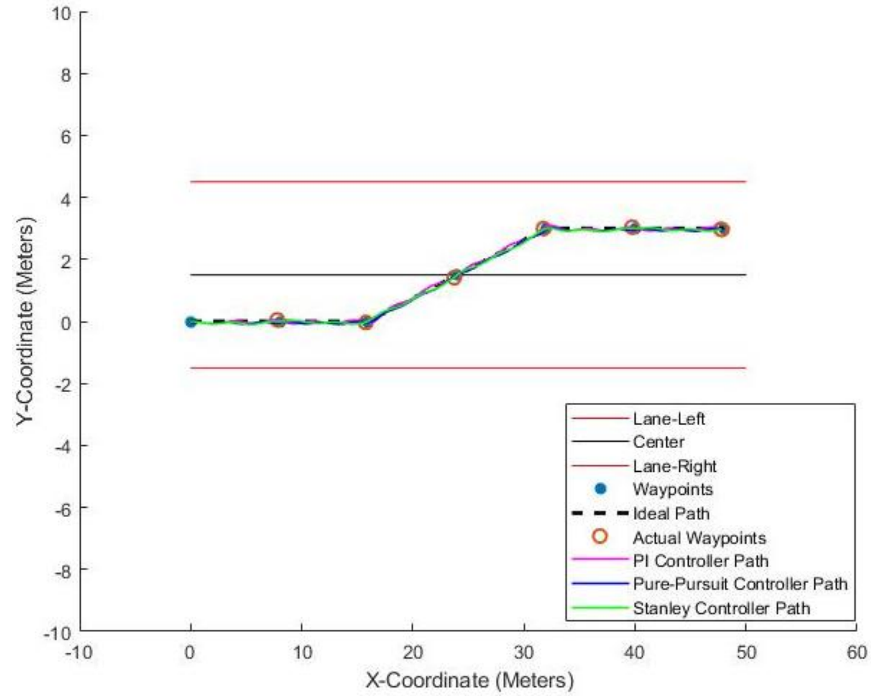

Figure 5-5: Path Tracking Performance of Controllers for Dynamic Lane Change

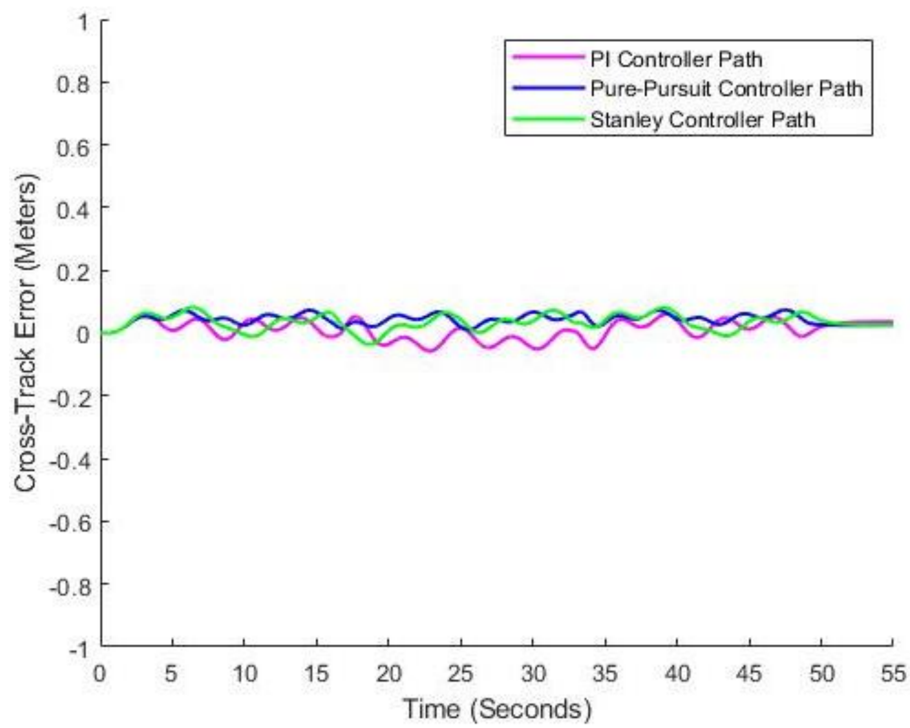

Figure 5-6: Cross Track Error of Vehicle for Dynamic Lane Change

From figures 5-1 to 5-6, the controller performance in terms of cross track error for various paths and controllers can be tabulated in Table 5-1.

Table 5-1: Max. Cross Track Error (Meters) Results for Custom Path

Path	PI Controller	Pure - Pursuit	Stanley
Custom 1	1.00	1.00	1.00
Custom 2	-0.64	0.10	0.00
Straight	-0.05	0.08	0.08
Dynamic Lane Change	-0.05	0.08	0.08

From table 5-1, it is seen that when path dynamics are significant, the PI controller performs the worst, as shown in section 1 and 2 for custom path in figure 5-1 and 5-2. Pure Pursuit and Stanley Controller have similar performance. It can also be observed that all the 3 controllers perform similarly for Straight Path and Dynamic Lane Change.

From figures 5-1 to 5-6, the path tracking performance of the controllers can be visualized. For the custom path region 1, all the controllers have the maximum deviation of 1 meter, but the PI controller converges very abruptly followed by an overshoot of 0.5 meter in the opposite direction. The other two controllers i.e. Pure Pursuit and Stanley converge smoothly converge with negligible overshoot. The Pure-Pursuit Controller convergence is due to the presence of Look Ahead Distance term. The Stanley controller converges faster than Pure-Pursuit Controller due to the presence of cross track error term as feedback. The performance analysis under ideal sensor condition validates the work done in [3] and [5].

5.2 Performance Analysis Considering Sensor Errors

In this, the controller analysis is done by including systematic error, noise and other dynamics in the sensor. Also, as discussed in section 4.2.10, a location specific random noise has been included in the custom path to consider the effects of stray magnetic fields.

5.2.1 Custom Path

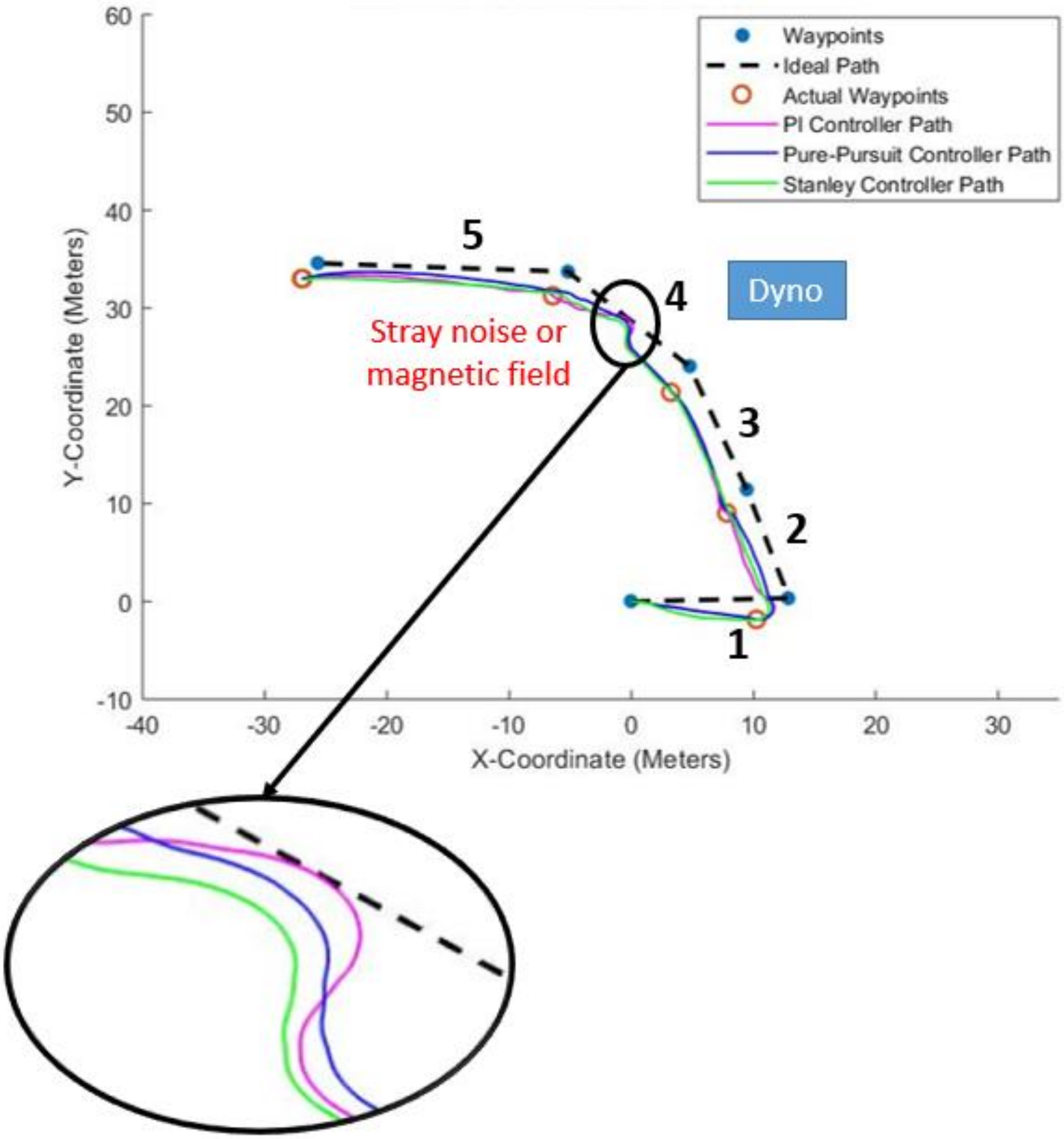

Figure 5-7: Effect of Sensor Errors and Location Specific Noise on Navigation Performance

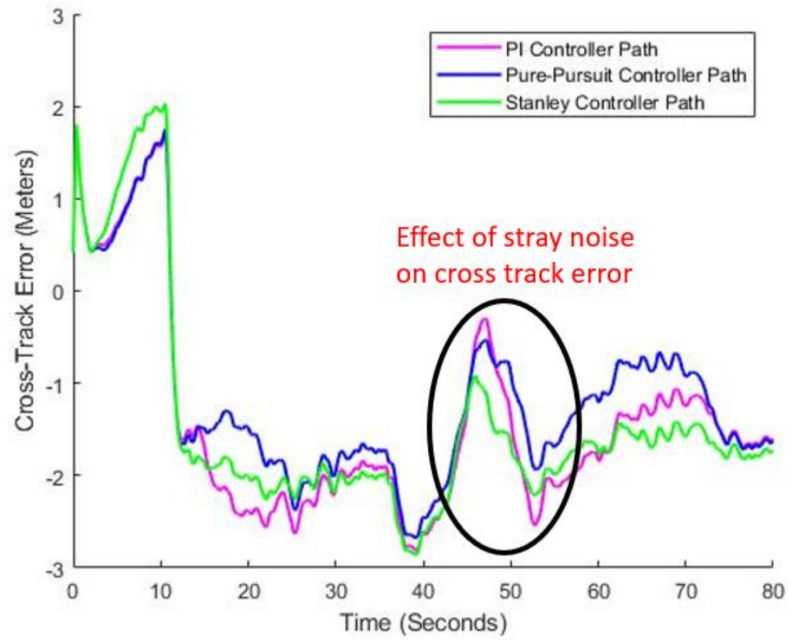

Figure 5-8: Effect of Sensor Errors on Cross Track Error for Custom Path

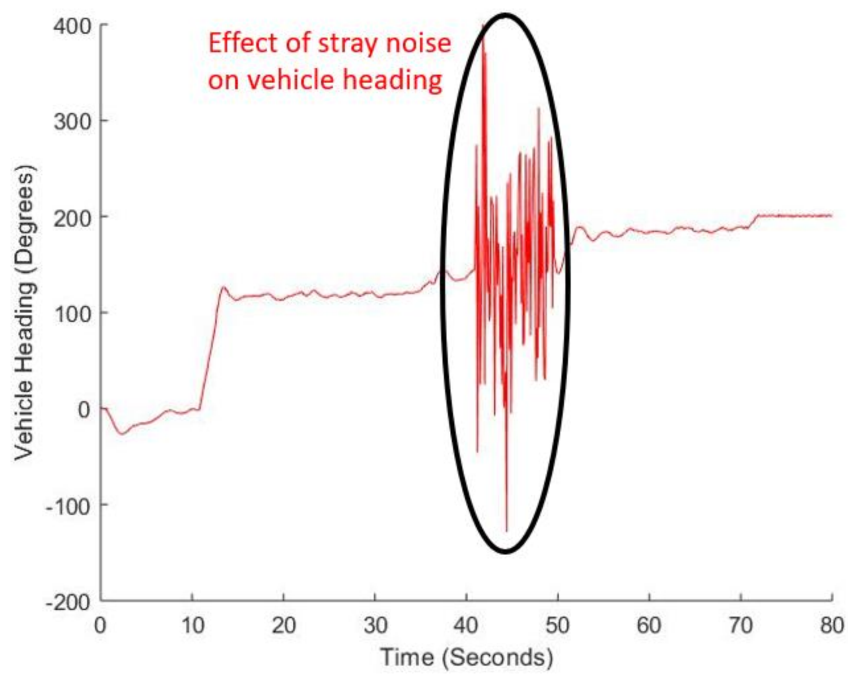

Figure 5-9: Effect of Stray Magnetic Fields on Magnetometer Output for Vehicle Heading

As seen in figure 5-7 and figure 5-8, the Stanley controller is affected the least by stray noise. From figure 5-9, the effect of stray fields can be seen in the vehicle heading values read by the magnetometer. From figures 5-7 to 5-8, the controller performance in terms of cross track error can be tabulated in Table 5-2.

Table 5-2: Max. Cross Track Error (Meters) Results for Custom Path Considering Sensor Errors

	PI Controller	**ΔXTE for PI Controller**	**Pure – Pursuit (PP)**	**ΔXTE for PP Controller**	**Stanley**	**ΔXTE for Stanley Controller**
1	1.7	-	1.7	-	2	-
2	-2.6	0.9	-2.2	0.5	-2.3	0.3
3	-2.8	0.2	-2.6	0.4	-2.8	0.5
4	**-0.3**	**2.5**	**-0.5**	**2.1**	**-0.9**	**1.9**
5	-1.6	1.3	-1.6	1.1	-1.7	0.8

Data in Table 5-2 shows that systematic error in GPS majorly affects the navigational performance of all the controllers. For a positive systematic error in X&Y directions, the vehicle moves away from the path.

The highlighted columns in table 5-2 compare the change in max. cross track error between the current segment and the prior segment. From Figure 5-7 and Table 5-2, segment 4, it can be seen that the PI controller is affected the most by the noise i.e. 2.5 meters of deviation. The circled section in figure 5-7 and 5-8 shows the effect of stray magnetic field which causes the vehicle to take an abrupt turn.

5.2.2 Straight Line Path

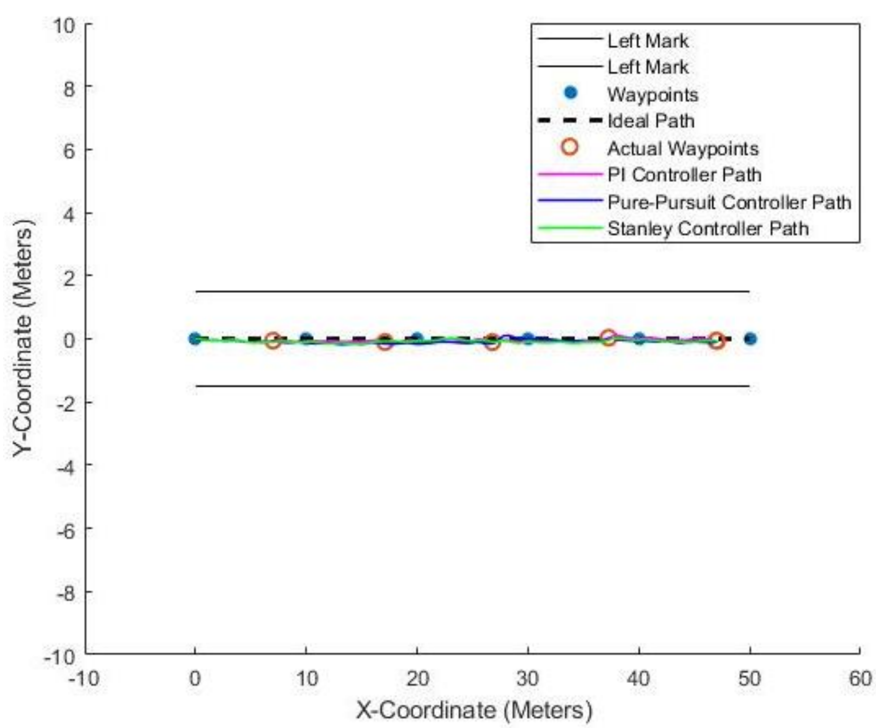

Figure 5-10: Effect of Sensor Errors Navigation Performance under 1D condition

Table 5-3: Distance (Meters) between vehicle stop point and actual waypoint for straight line test for 1D condition

Waypoint No.	PI	Pure Pursuit	Stanley
1	3.00	3.00	3.00
2	2.90	2.77	2.77
3	3.20	2.93	2.94
4	2.72	2.93	2.91
5	3.03	2.90	2.88

Figure 5-10 and Table 5-3 clearly show that due systematic error in GPS x-direction, the vehicle stops approximately 3 meters before the actual waypoint.

5.2.3 Dynamic Lane Change

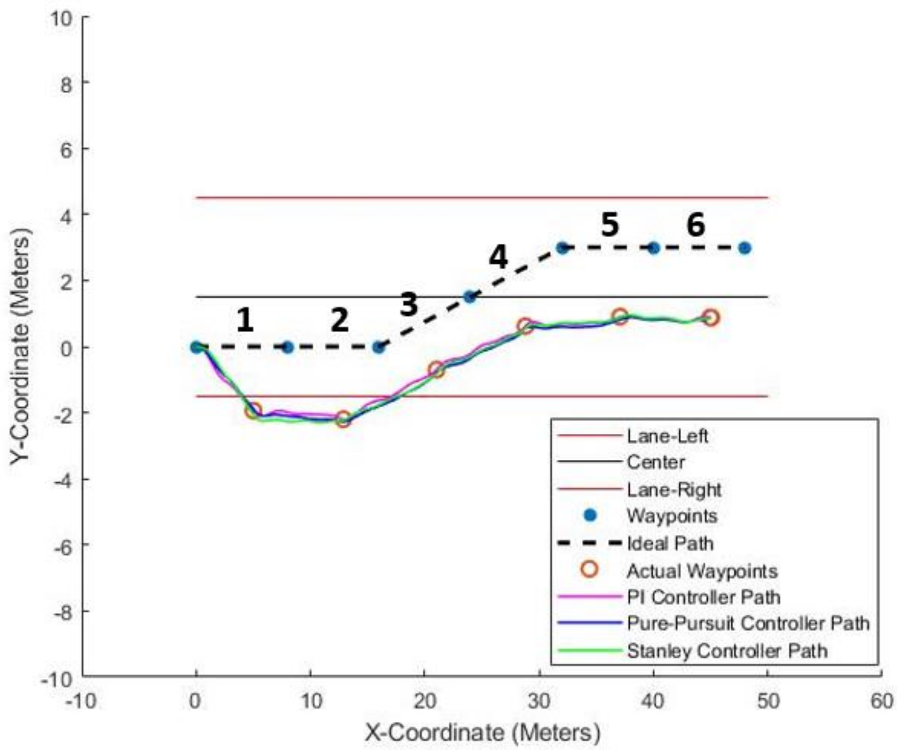

Figure 5-112: Effect of Sensor Errors on Navigation Performance

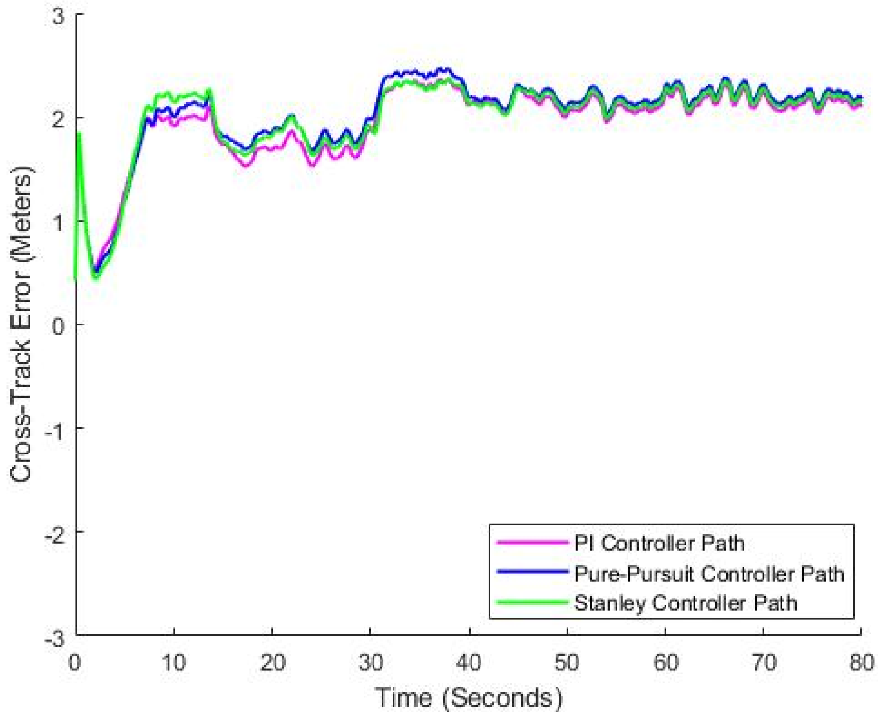

Figure 5-123: Effect of sensor error on Cross Track Error

From figure 5-13 and 5-14, the table 5-4 is derived which sums up maximum cross track error of the vehicle stop point from the actual path.

Table 5-4: Max. Cross Track Error (Meters) Results for Dynamic Lane Change Considering Sensor Errors

Segment	PI Controller	Pure Pursuit	Stanley
1	1.98	1.98	2.00
2	2.11	2.21	2.21
3	1.63	1.85	1.85
4	1.94	2.01	1.98
5	2.31	2.31	2.31
6	2.13	2.14	2.14

From table 5-4, and figures 5-13 and 5-14, it can be seen that the GPS systematic error pre-dominates over other sensor error. After deviation from the actual path, all the controllers maintain a similar offset from the path and follow the path trajectory.

6 IMPROVING NAVIGATION / WAYPOINT TRACKING USING STATE ESTIMATION APPROACH

As seen in the previous section, the error in GPS signal affects the path tracking performance of the vehicle. Also, the presence of stray noise affects the steering performance and causes the vehicle to behave abruptly as seen in section 5.2.1.

In order to improve the navigational performance of the vehicle, we need to improve the positional / localization accuracy of the vehicle. As discussed in section 2.2, GPS has systematic error as dominant error, hence, localization using position data from GPS will always have some offset from the true position. Sensory data from accelerometers and wheel speed sensors can be combined with GPS Data to improve the accuracy in navigation. This can be achieved by using the concept of sensor fusion. Using the laws of motion and by assuming constant acceleration at every time step, we can model the position equation as follows:

$$S_f = S_i + u\Delta t + \frac{1}{2} a\Delta t^2 \text{ ……………… (21)}$$

where S_f = Position at $t + \Delta t$, S_i = Position at time t, u = velocity at time Δt and a = acceleration at time step Δt.

It is also seen from the steering control laws, section 4.1, that vehicle heading sensed by the Magnetometer is an input to the controller, but it is affected by stray magnetic fields and sensor noise as seen in section 5.2.1. Hence, there is also a need to implement state estimation techniques, to generate noise free states for the controller. One method is to combine Magnetometer data with yaw-rate obtained from Gyroscope. The vehicle heading also known as Yaw has a linear relation with Yaw-Rate, given by,

$$\varphi_f = \varphi_i + \varphi' \, \Delta t \text{ ……………… (21)}$$

where ψ_f = Yaw or Vehicle Heading at time $t + \Delta t$, ψ_i = Yaw or Vehicle Heading at time t and ψ' is Yaw-Rate at time step Δt.

Combining sensory data allows choosing a state in between a measured value and state obtained by prediction from a model. For dynamic conditions, it is required to alter the weights at every time step depending upon the quality of measurement. If sensor data is good more weight should be given to it, else for poor sensor data, weightage is given to prediction. This can be achieved by the use of Kalman Filters. One might say, that using a model to predict the states should be sufficient, however system dynamics can never by modelled perfectly. Under such circumstances, even if the initial predictions are correct, the states would diverge from actual values due to non-linearities in the physical system. The use of measured value in the Kalman Filter prevents the predictions to diverge.

6.1 Kalman Filter Equations

The Kalman Filter consists of 2 Stages:

- Prediction – Uses model equations to predict the next system state based on current states.
- Update – Update the current states based on weights assigned to measured values and predicted values

The State Space equation is given by $\overline{x_{t+1}} = Ax_t + Bu_t$,where, $\overline{x_{t+1}}$ = Predicted System State at time t + 1 from previous state x_t, A = State Transition Matrix, B = Control Matrix, u_t = Input Matrix

The filter will not be used to generate control inputs, so B = 0

Hence, we get, $\overline{x_{t+1}} = Ax_t$ ………………..(22)

$\bar{P} = APA^t + Q$………………..(23)

$\bar{P}$ = Predicted State Co-Variance Matrix and P = State Co-Variance Matrix

Q = Process Noise or Noise in the Model

Equations 22 & 23 form the prediction stage

Residual, $y = Z - H\,\overline{x_{t+1}}$ ………………..(24), where Z = Measured states from sensor, H = Measurement function to scale predicted values as per Z

Uncertainty, in measurement $S = H\bar{P}H^t + R^{-1}$ ………………..(25), where R = Measurement Noise Vector

Kalman Gain K = $\bar{P}H^tS^{-1}$ ………………..(25), this is the step where the filter decides whether to give more weightage to measured value or predicted value. Higher the value of K, more value is given to measurement.

$x_t = \bar{x_t} + Ky$………………..(27), new estimated state based on the Kalman Gain

Updating the process co-variance, $P = (I - KH)\bar{P}$ ………………..(28)

Equations 24, 25, 26, 27 & 28 form the update stage of the filter where the filter estimates the new states from noisy measurements and x_t & P are used for the next prediction.

For the initial step / iteration the P and the x, matrices are required to initialize the filter. In the following iterations, the filter will estimate these values

6.2 Implementation of 1D – 2nd Order Kalman Filter for Improved Position Feedback in Straight Line Path

The following matrices where defined and initialized:

$x_t = \begin{bmatrix} s_t \\ v_t \\ a_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ since, at *t=0*, all the states start from *0*

$A = \begin{bmatrix} 1 & \Delta t & \Delta t^2/2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix}$ using the equations of motion discussed earlier

$P = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_a^2 \end{bmatrix}$, where the diagonals are the sensor variances for position, velocity and acceleration.

$Q = \begin{bmatrix} \Delta t^4/4 & \Delta t^3/2 & \Delta t^2/2 \\ \Delta t^3/2 & \Delta t^2 & \Delta t \\ \Delta t^2/2 & \Delta t & 1 \end{bmatrix} .* \text{Ø}^2$, this is the piece-wise model as discussed in [26] for constant acceleration at a given time-step and but differs at every step. A more accurate model as described in [26] is the continuous time noise model Q_c, which is used to find Q by integrating and for each time step using $Q = \int_0^{\Delta t} F\ Q_c\ F^T\ dt$. This process is more computationally intensive.

$Z = \begin{bmatrix} s_{meas} \\ v_{meas} \\ a_{meas} \end{bmatrix}$ and $H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$R = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_a^2 \end{bmatrix} .* MR$, where the diagonals are the sensor variances for position, velocity and acceleration.

MR and Ø, are used to set Q and R matrix and tune the filter.

It should be noted that high value of R, tells the filter that the measurement is noisy, and the filter will favor prediction at every step. A low value of Q tells the filter that the model defined in filter perfectly defines the system and to put more weights on the predicted value. A low value of R tells the filter that the measurement has less noise and the filter will favor sensor data at every step. A high value for Q tells the filter that the model is not accurate. Initially **Ø = 0.05 and MR=10**, since we know that the measurements are not perfect.

6.2.1 Filter Results for Various Controllers Under 1D conditions

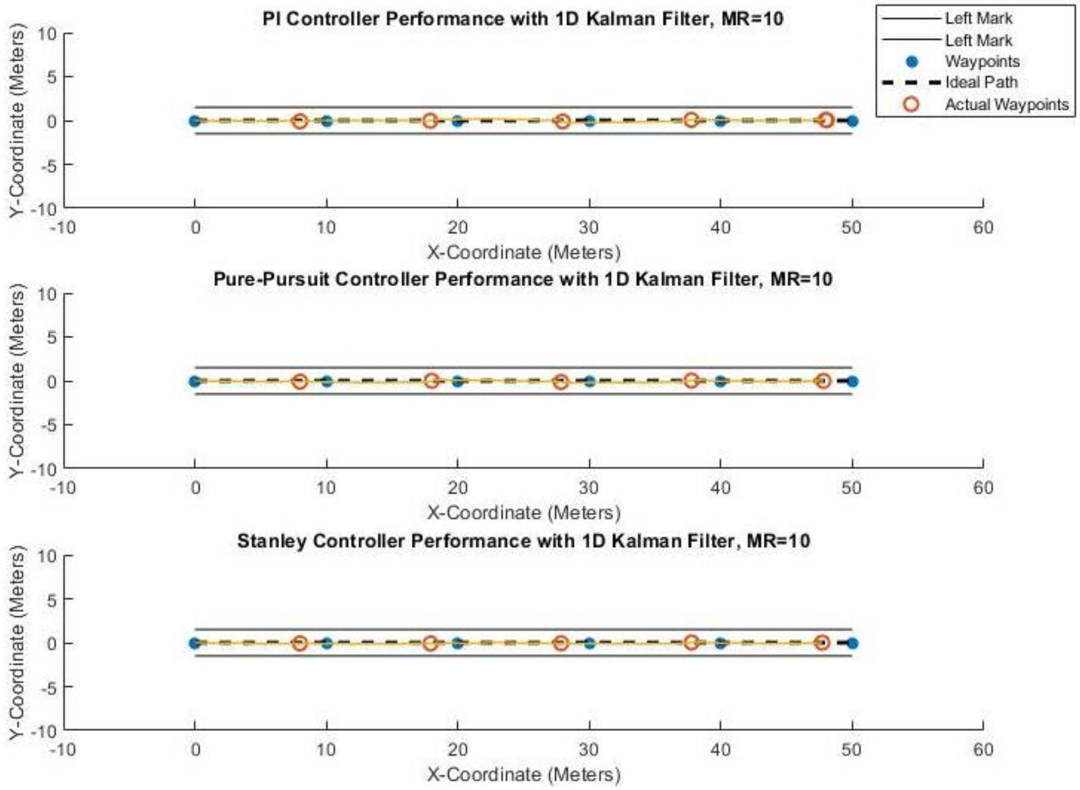

Figure 6-1: Controller Performance with 1D Kalman Filter, MR=10

Table 6-1: Difference in distance between vehicle stop point and waypoint for various controllers with Kalman Filter MR=10

Waypoint No.	PI	Pure Pursuit	Stanley
1	1.99 (33%)	1.99 (33%)	2.00 (33%)
2	2.09 (27%)	1.97 (29%)	2.07 (25%)
3	2.03 (36%)	2.15 (26%)	2.15 (27%)
4	2.24 (17%)	2.22 (24%)	2.22 (24%)
5	2.04 (32%)	2.19 (24%)	2.29 (21%)

Compared to the table 5-3, the filter is able to reduce the difference in distance between the vehicle stop point and waypoint. The percentage improvement is given in the parenthesis. However, the filter starts lagging behind due to the systematic error in GPS affecting the filter during residual calculation. The next step would be to include the GPS systematic error in the filter's Z matrix. This would allow the filter to have prior knowledge of the GPS systematic error.

$$Z = \begin{bmatrix} s_{meas} - gps\_sys\_x \\ v_{meas} \\ a_{meas} \end{bmatrix}$$

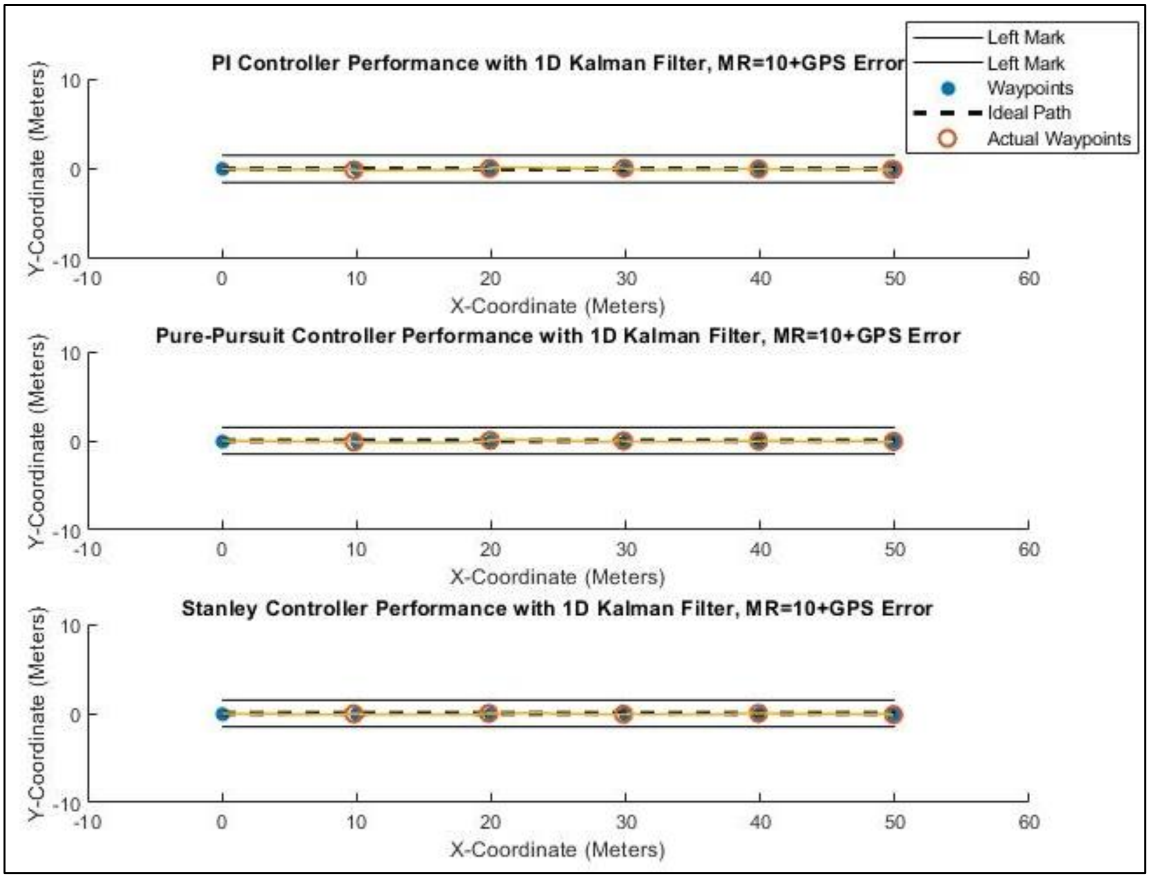

Figure 6-2: Controller Performance with 1D Kalman Filter, MR=10 with GPS Error included

Table 6-2: Distance (Meters) between vehicle stop point and waypoint for various controllers with Kalman Filter MR=10 with GPS Error included

Waypoint No.	PI	Pure Pursuit	Stanley
1	0.20	0.20	0.20
2	0.10	0.07	0.18
3	0.12	0.15	0.15
4	0.12	0.12	0.13
5	0.22	0.09	0.09

Compared to the results in Table 6-2, there is significant improvement in tracking performance and an accuracy at the centimeter level has been achieved. It should be noted that GPS systematic error depends on the satellite orientation and the signal quality, and this simulation shows a special case when the error in X&Y direction is 2.12 meters.

6.3 Implementation of 1st Order Kalman Filter for Vehicle Heading Improvement

$x_t = \begin{bmatrix} \varphi \\ \varphi' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$ using the linear relationship between yaw and yaw-rate

$P = \begin{bmatrix} \sigma_{\varphi}^2 & 0 \\ 0 & \sigma_{\varphi'}^2 \end{bmatrix}$ and $R = \begin{bmatrix} \sigma_{\varphi}^2 & 0 \\ 0 & \sigma_{\varphi'}^2 \end{bmatrix} .* MR,$

where the diagonals are the sensor variances for yaw and yaw-rate

$Q = \begin{bmatrix} \Delta t^4/4 & \Delta t^3/2 \\ \Delta t^3/2 & \Delta t^2 \end{bmatrix} .* \emptyset^2$

$Z = \begin{bmatrix} \varphi_{meas} \\ \varphi'_{meas} \end{bmatrix}$ and $H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

From figure 5-9, it can be seen that the magnetometer readings have less noise but is affected by stray magnetic fields. So, it can be assumed that the measurements are of good quality when there is no noise and correction is only needed when there is an external disturbance. Using trial and error, the value of $\emptyset$ **was chosen to be 4** and the value **of *MR* was chosen to be 0.5**. The filter performance was evaluated on the custom path for PI controller as it was affected the most by the stray noise.

6.3.1 Filter Implementation Results for Vehicle Heading Estimation

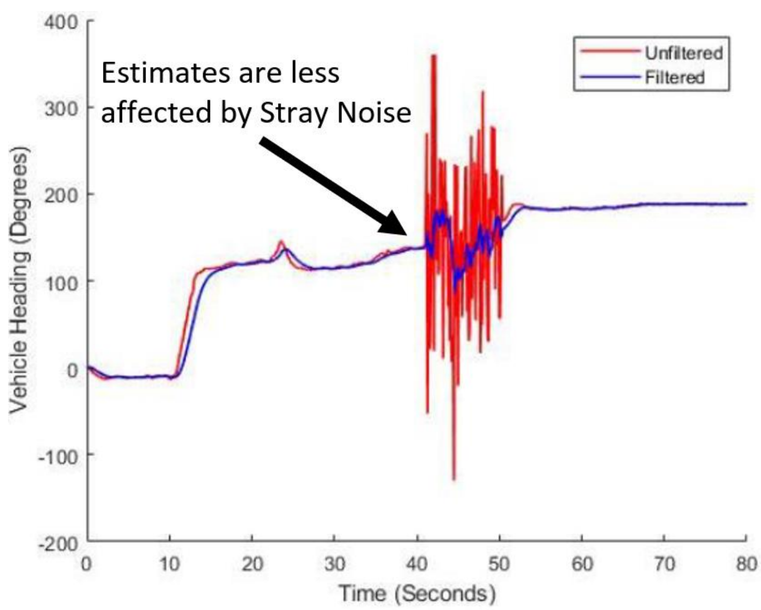

Figure 6-3: Filter Performance for Vehicle Heading Estimation

From the figure 6-3, it can be seen that the filter performs well in estimating the vehicle heading under noisy conditions.

As seen in figure 5-7 and 5-8, PI controller was affected the most by the stray magnetic fields. So, the navigational performance was also compared for PI controller for filtered and non-filtered condition.

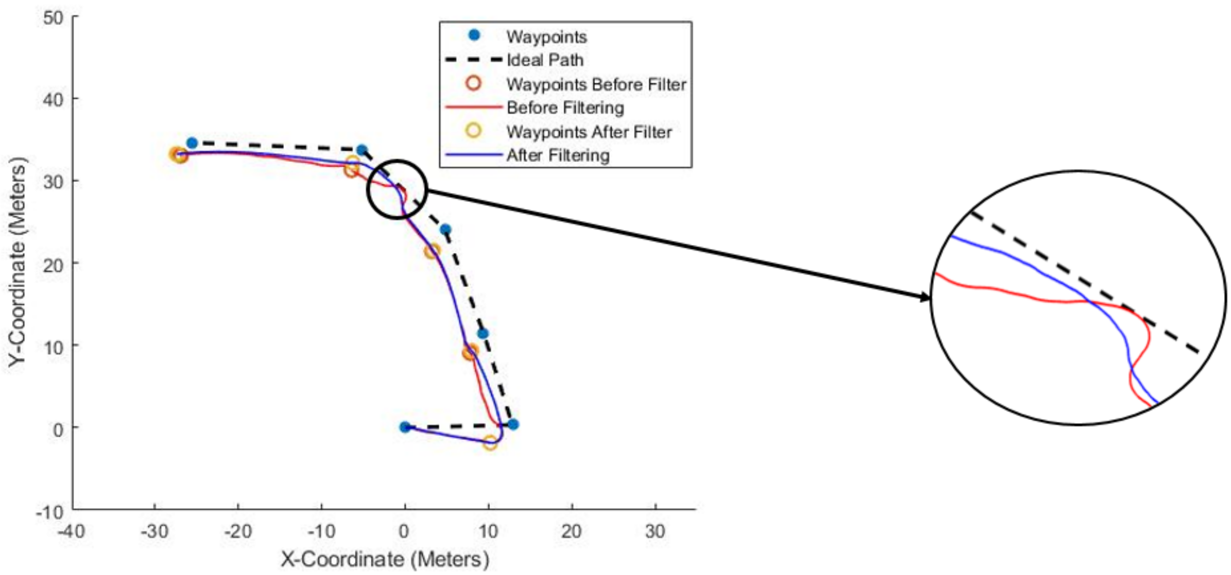

Figure 6-4: PI Controller Performance for Filtered Vehicle Heading

In figure 6-4, the highlighted portion shows that, although the vehicle deviates from path, the steering response is not abrupt in nature and is able to smoothly converge with the trajectory of the previous controller performance.

7 CONCLUSION AND FUTURE SCOPE OF WORK

In this report, the effects of actuator dynamics and sensor errors for autonomous navigation are analyzed for 3 different types of steering controllers. Initial analysis is done from experimental data and the factors for poor navigation are identified. Based on this, the need for model-based controller analysis was established.

Sensors, actuators and the vehicle kinematics were modelled based on actual component test data followed by the implementation of steering controls i.e. PI, Pure-Pursuit and Stanley controller along with Speed Controller, Navigation and Waypoint monitoring systems. These controllers were tuned for three different path conditions with cross-track error as the most important performance metric.

From the results, it can be seen that all the controllers deviated from the desired path and there was an offset between vehicle trajectory and the ideal path. It can be concluded that localization using GPS is highly biased by the presence of systematic error. When comparing the response or control action of the controllers, Stanley controller and Pure-Pursuit controller were superior in performance as compared to PI controller. However, all the steering controllers were affected by stray magnetic fields, PI controller being affected the most due to the absence of path dynamics in the control law.

It can be seen, that by the application of Sensor Fusion between GPS, Wheel Speed Sensor and Accelerometer via. 1D - 2nd Order Kalman Filter, the vehicle positional accuracy improves for 1D waypoint tracking, since, the filter was able to estimate the position of the vehicle from the noisy measurement. Also, by adding the knowledge of GPS systematic error in the filter, accuracy at centimeter level was achieved. It is also seen that by applying sensor fusion between Gyroscope and Magnetometer, the yaw or vehicle heading output is improved as the estimates are less affected by the stray magnetic fields.

In the future, a learning-based technique will be developed to provide the GPS systematic error input for the Kalman Filter under various satellite and climatic conditions. This would be followed by the implementation of a 2D Kalman Filter for position estimation and localization in X&Y direction. After successful simulation work, the model will be modified for implementation on a real time vehicle ECU.

8 REFERENCES

1.) Rafael Vivacqua, Raquel Vassallo, Felipe Martins. “A Low-Cost Sensors Approach for Accurate Vehicle Localization and Autonomous Driving Application”, MDPI Journal of Sensors, 2017.

2.) Jun Yang, Hong Bao, Nan Ma, Zuxing Xuan. “An Algorithm of Curved Path Tracking with Prediction Model for Autonomous Vehicle”, 13th International Conference on Computational Intelligence and Security, 2017, pp. 405 – 408.

3.) Jarrod M. Snider. “Automatic Steering Methods for Autonomous Automobile Path Tracking”, MS Thesis, Carnegie Mellon University, 2009.

4.) Byung-Hyun Lee, Sung-Hyuck Im, Moon-Beom Heo and Gyu-In Jee. “Curve Modeled Lane and Stop Line Detection based GPS Error Estimation Filter “, 2015 IEEE Intelligent Vehicles Symposium, Seoul, Korea, 2015. pp. 406 – 411.

5.) Stefan F. Campbell. “Steering Control of an Autonomous Ground Vehicle with Application to the DARPA Urban Challenge”, MS Thesis, Massachusetts Institute of Technology, 2007.

6.) Mertcan Cibooglu, Umut Karapinar, Mehmet Turan Söylemez. “Hybrid Controller Approach for Autonomous Ground Vehicle Path Tracking Problem”. 25th Mediterranean Conference on Control and Automation, 2017, pp. 584 - 588

7.) Pierre Pettersson. “Estimation of Vehicle Lateral Velocity”, MS Thesis, Lund University, 2008, pp. 2 – 13, 18 – 19, 24.

8.) Md. Rashedul Islam, Jong-Myon Kim. “An Effective Approach to Improving Low-Cost GPS Positioning Accuracy in Real-Time Navigation”. The Scientific World Journal Volume 2014, Article ID 671494.

9.) Francois Caron, Emmanuel Duflos, Denis Pomorski, Philippe Vanheeghe. “GPS/IMU data fusion using multi-sensor Kalman filtering: introduction of contextual aspects”. Elsevier Journal, pp. 221 – 230, 2004.

10.) Hang Guo, Min Yu, Chengwu Zou, Wenwen Huang. “Kalman filtering for GPS/magnetometer integrated navigation system”. Elsevier Journal, pp. 1350 – 1357, 2010.

11.) Teawon Han, Yanghyun Kim, Kisung Kim. “Lane Detection & Localization for UGV in Urban Environment”. IEEE 17th International Conference on Intelligent Transportation Systems (ITSC), 2014, Qingdao, China, pp. 590 – 596.

12.) Chris J. Ostafew, Angela P. Schoellig, Timothy D. Barfoot. “Learning-Based Nonlinear Model Predictive Control to Improve Vision-Based Mobile Robot Path-Tracking in Challenging Outdoor Environments”. IEEE International Conference on Robotics & Automation (ICRA) Hong Kong Convention and Exhibition Center, 2014, pp. 4029 – 4036.

13.) Malavika Panicker, Tanzeela Mitha, Kalyani Oak, Ashwini M. Deshpande. “Multisensor Data Fusion for an Autonomous Ground Vehicle”. Conference on Advances in Signal Processing (CASP) Cummins College of Engineering for Women, Pune, 2016, pp. 507 – 512.

14.) J. Pérez, J. Godoy, V. Milanés, J. Villagrá, E. Onieva. “Path following with backtracking based on fuzzy controllers for forward and reverse driving”.

Intelligent Vehicles Symposium Alcalá de Henares, Spain, June, 2012, pp. 1108 – 1113.

15.) Martin Lundgren. “Path Tracking for a Miniature Robot”. MS Thesis, Umeå University, 2003.

16.) M. Pe´rez-Ruiz, J. Carballido, J. Agu¨era, J. A. Gil. “Assessing GNSS correction signals for assisted guidance systems in agricultural vehicles”. Springer Science, Precision Agric , 2011, pp. 639 – 652.

17.) Aidan O'Dwyer. “PI and PID controller tuning rules: an overview and personal perspective”. Dublin Institute of Technology, 2006.

18.) Fredrick S. Solheim, Jothiram Vivekanandan1, Randolph H. Ware, Christian Rocken. “Propagation Delays Induced in GPS Signals by Dry Air, Water Vapor, Hydrometeors and Other Particulates”. Journal of Geophysical Research, 104, 9663-9670, 1999.

19.) Hanieh Deilamsalehy, Timothy C. Havens. “Sensor Fused Three-dimensional Localization Using IMU, Camera and LiDAR”. IEEE Journal, 2016

20.) SAE Levels of Automated Driving - https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles

21.) Typical System Architecture- https://www.slideshare.net/KiranKumar1455/autonomous-cars-61077075

22.) Ford Heart Rate Monitoring Seat - http://www.medtees.com/content/ecg_seat_fact_sheet_2.pdf

23.) Zhiping Liu, Mingjing Zhu. “Calibration and Error Compensation of magnetometer”. 26th Chinese Control and Decision Conference, 2014, pp. 4122 – 4126

24.) Minha Park and Yang Gao. “Error and Performance Analysis of MEMS-based Inertial Sensors with a Low-cost GPS Receiver”. MDPI Journal of sensors,2008

25.) MEMS Technology - https://www.memsnet.org/mems/what_is.html

26.) Concept Behind Sensor Fusion - https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/04-One-Dimensional-Kalman-Filters.ipynb

27.) Effect of Look Ahead Distance on Pure Pursuit Controller - https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html

28.) Forward and Inverse Kinematics - https://www.mathworks.com/matlabcentral/fileexchange/66586-mobile-robotics-simulation-toolbox

29.) Zero Order Hold - https://en.wikipedia.org/wiki/Zero-order_hold

30.) WGS84 Model - http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf

9. APPENDIX

9.1 Python code used for initial vehicle test and analysis

```
#Import Libraries
import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import serial
import math
import Adafruit_BBIO.UART as UART
import time
from time import sleep

#Initialization of UART
UART.setup("UART1")  #Initialize UART1
UART.setup("UART4")  #Initialize UART4
ser=serial.Serial('/dev/ttyO1',19200) #Initialize Serial Port at 19200 for Garmin GPS
ser1=serial.Serial('/dev/ttyO4',115200) #Initialize Serial Port at 115200 for UM7

#Assign DI, DO and PWM
start_button="P8_8"
okled_pin="P8_10" #red LED
runled_pin="P8_12" #yellow LED
esc_pin = "P9_21"
ser_pin = "P8_13"
GPIO.setup(okled_pin, GPIO.OUT)
GPIO.setup(runled_pin, GPIO.OUT)
GPIO.setup(start_button, GPIO.IN)
GPIO.output(okled_pin, GPIO.LOW)
GPIO.output(runled_pin, GPIO.LOW)

#Reset PWM to default conditions
dc_fbeep = 13.93
dc_stop=11
ser_dc = 26.2
esc_f=90.9
ser_f=181.2
PWM.start(esc_pin, dc_fbeep, 90.9) #starting frequency and duty cycle for esc_pin
time.sleep(3)
PWM.start(ser_pin, ser_dc, 181.2) #starting frequency and duty cycle for ser_pin
time.sleep(0.1)
PWM.set_duty_cycle(esc_pin,float(dc_fbeep))
PWM.set_duty_cycle(ser_pin,float(ser_dc))
```

```
PWM.stop(esc_pin)
PWM.stop(ser_pin)

#Initialize ESC and servo
PWM.start(esc_pin, dc_fbeep, 90.9)
time.sleep(3)
PWM.start(ser_pin, ser_dc, 181.2) #starting duty cycle for ser_pin
throttle=30 #Percentage of throttle
throttle_dc=(0.0107*throttle)+13.93 #Throttle to duty cycle Linear Map
steering_angle=0 # Initial steering position
ser_dc=(0.1667*steering_angle)+26.2 #Steering Angle to duty cycle Linear Map

#Open file for write
f=open("Test.txt","a")
f.write("LoopTime WaypointNo. CurrLat CurrLong TargetLat TargetLong TargetHeading
CurrentHeading HeadingError distanceToTarget Speed Yaw_rate Ax Ay Az Magx Magy
Magz\n")

# Way point/map parameters
WAYPOINT_DIST_TOLERANCE = 2
HEADING_TOLERANCE = 10
TarLat = [47.169502,47.169640,47.169795,47.169917,47.169934]
TarLong = [-88.507541,-88.507583,-88.507640,-88.507768,-88.508037]
x0 = 47.169502    # Vehicle start point
y0 = -88.507711
n=4 #number of waypoints, zero position being the first waypoint
i=0
t1=0
t2=0

#Empty the serial buffers for serial input
ser.flushInput()
ser.flushoutput()

#General Parameters
gpscount=3
count=1
d0=0    #starting point distance
d_cal=0
delta=0
starttime=0
looptime=0
z=1
speed=0
integral=0
```

#Class for GPS Data Read

```
class GPS:
	def read(self):
		ser.flushInput()
		ser.flushOutput()
		while ser.inWaiting()==0:
			pass
		self.NMEA1=ser.readline()
		while ser.inWaiting()==0:
			pass
		self.NMEA2=ser.readline()
		NMEA1_array=self.NMEA1.split(',')
		NMEA2_array=self.NMEA2.split(',')

		if NMEA1_array[0]=='$GPGGA':
			self.latDeg=NMEA1_array[2][:-8]
			self.latMin=NMEA1_array[2][-8:]
			self.latHem=NMEA1_array[3]
			self.lonDeg=NMEA1_array[4][:-8]
			self.lonMin=NMEA1_array[4][-8:]
			self.lonHem=NMEA1_array[5]
			if NMEA1_array[7]==' ' or NMEA1_array[7]==0:
				self.sat=0
			else:
				self.sat=NMEA1_array[7]

		if NMEA2_array[0]=='$GPRMC':
			self.latDeg=NMEA2_array[3][:-8]
			self.latMin=NMEA2_array[3][-8:]
			self.latHem=NMEA2_array[4]
			self.lonDeg=NMEA2_array[5][:-8]
			self.lonMin=NMEA2_array[5][-8:]
			self.lonHem=NMEA2_array[6]
			if NMEA2_array[7]==' ' or NMEA2_array[7]==0:
				self.speed=0
			else:
				self.speed=NMEA2_array[7]

		if NMEA2_array[0]=='$GPGGA':
			self.latDeg=NMEA2_array[2][:-8]
			self.latMin=NMEA2_array[2][-8:]
			self.latHem=NMEA2_array[3]
			self.lonDeg=NMEA2_array[4][:-8]
			self.lonMin=NMEA2_array[4][-8:]
			self.lonHem=NMEA2_array[5]
```

```
			if NMEA2_array[7]==' ' or NMEA2_array[7]==0:
				self.sat=0
			else:
				self.sat=NMEA2_array[7]

		if NMEA1_array[0]=='$GPRMC':
			self.latDeg=NMEA1_array[3][:-8]
			self.latMin=NMEA1_array[3][-8:]
			self.latHem=NMEA1_array[4]
			self.lonDeg=NMEA1_array[5][:-8]
			self.lonMin=NMEA1_array[5][-8:]
			self.lonHem=NMEA1_array[6]
			if NMEA1_array[7]==' ' or NMEA1_array[7]==0:
				self.speed=0
			else:
				self.speed=NMEA1_array[7]

#Class for IMU Data Read
class UM7():
	def read(self):
		ser1.flushInput()
		ser1.flushOutput()
		ser1.flushInput()
		ser1.flushOutput()
		time.sleep(0.1) #Time delay to serial input / output buffers

		while ser1.inWaiting()==0:
				pass
		self.NMEA3=ser1.readline()	#Read NMEA1
		NMEA3_array=self.NMEA3.split(',')

		while ser1.inWaiting()==0:
				pass
		self.NMEA4=ser1.readline()	#Read NMEA2
		NMEA4_array=self.NMEA4.split(',')

		while ser1.inWaiting()==0:
				pass
		self.NMEA5=ser1.readline()	#Read NMEA3
		NMEA5_array=self.NMEA5.split(',')

		while ser1.inWaiting()==0:
				pass
		self.NMEA6=ser1.readline()	#Read NMEA4
		NMEA6_array=self.NMEA6.split(',')
```

```
                    if NMEA3_array[0]=='$PCHRP': # Statement to check the condition of first NMEA sentence
                              if NMEA3_array[0]=='$PCHRP':
                                        self.yaw=NMEA3_array[7] #Yaw or current heading

                              if NMEA4_array[0]=='$PCHRS':
                                        self.yaw_rate=NMEA4_array[5] #Yaw Rate

                              if NMEA5_array[0]=='$PCHRS':
                                        self.ax=NMEA5_array[3] #Acceleration in X Direction
                                        self.ay=NMEA5_array[4] #Acceleration in Y Direction
                                        self.az=NMEA5_array[5] #Acceleration in Z Direction

                              if NMEA6_array[0]=='$PCHRS':
                                        self.magx=NMEA6_array[3] #Mag Sensor value in X Direction
                                        self.magy=NMEA6_array[4] #Mag Sensor value in Y Direction
                                        self.magz=NMEA6_array[5] #Mag Sensor value in Z Direction


myGPS=GPS()
imu=UM7()
time.sleep(1)
lat=0
sat=0
flag=0
total_gain=0
j_max=100
sum_yaw_rate=0
sum_ax=0
sum_ay=0
sum_az=0

# Self routine having 100 iterations to check for GPS and IMU data integrity
for j in range(0,j_max):
          myGPS.read()
          imu.read()
          latprev=lat
          myGPS.latMin=float(myGPS.latMin)
          myGPS.latDeg=float(myGPS.latDeg)
          myGPS.latMin = myGPS.latMin * 0.01666667      #Convert Minutes to Degrees for latitude
```

```
        lat = myGPS.latDeg + myGPS.latMin
        GPIO.output(okled_pin, GPIO.LOW)
        status=0


if lat-latprev!=0: #Check GPS Data before proceeding
        GPIO.output(okled_pin, GPIO.HIGH)
        flag=1
else:
        GPIO.output(okled_pin, GPIO.LOW)
        flag=0

while(status==0 and flag=1): #Wait for the start button to be switched on
        status=GPIO.input(start_button)
        GPIO.output(okled_pin, GPIO.HIGH)
        old_status=status
        time.sleep(0.5)

# Accelerometer and Gyroscope Self-Calibration routine
for j in range(0,j_max):
        imu.read()

        yaw_rate=float(imu.yaw_rate)
        sum_yaw_rate=sum_yaw_rate+yaw_rate

        ax=float(imu.ax)*9.81
        sum_ax=sum_ax+ax

        ay=float(imu.ay)*9.81
        sum_ay=sum_ay+ay

        az=float(imu.az)*9.81
        sum_az=sum_az+az

yaw_rate_cal=sum_yaw_rate/j_max
ax_cal=sum_ax/j_max
ay_cal=sum_ay/j_max
az_cal=sum_az/j_max

#Main loop
while(i<=n and status==1):
        GPIO.output(okled_pin, GPIO.LOW)
        GPIO.output(runled_pin, GPIO.HIGH)
        PWM.set_duty_cycle(esc_pin,float(throttle_dc))
        t1=time.time()
```

```
imu.read()
curr_hdng_deg=float(imu.yaw)
if curr_hdng_deg<0:
    curr_hdng_deg=curr_hdng_deg+360
yaw_rate=(float(imu.yaw_rate))-yaw_rate_cal
ax=(float(imu.ax)*9.81)-ax_cal
ay=(float(imu.ay)*9.81)-ay_cal
az=(float(imu.az)*9.81)-az_cal
magx=float(imu.magx)
magy=float(imu.magy)
magz=float(imu.magz)

if z==1:
    x = x0   # Vehicle start point #center point of the APSRC road
    y = y0
    d = d0
    z=z+1
else:
    myGPS.read()
    myGPS.latMin=float(myGPS.latMin)
    myGPS.lonMin=float(myGPS.lonMin)
    myGPS.latDeg=float(myGPS.latDeg)
    myGPS.lonDeg=float(myGPS.lonDeg)
    speed=round(((float(myGPS.speed))*0.514444),2)
    sat=float(myGPS.sat)

    myGPS.latMin = myGPS.latMin * 0.01666667        #Convert Minutes to Degrees for latitude
    myGPS.lonMin = myGPS.lonMin * 0.01666667        #Convert Minutes to Degrees for longitude
    CurrLat = myGPS.latDeg + myGPS.latMin
    CurrLong = myGPS.lonDeg + myGPS.lonMin
    if myGPS.latHem=='S':        #Convert latitude to -ve if in southern hemisphere
        CurrLat = CurrLat * -1
    if myGPS.lonHem=='W':        #Convert longitude to -ve if in western hemisphere
        CurrLong = CurrLong * -1
    x=CurrLat
    y=CurrLong

#Now calculations for Distance to Target
TarLat1 = math.radians(TarLat[i])
TarLong1 = math.radians(TarLong[i])
CurrLat1 = math.radians(x)
```

```
CurrLong1 = math.radians(y)
delta = CurrLong1 - TarLong1
sdlong = math.sin(delta)
cdlong = math.cos(delta)
slat1 = math.sin(CurrLat1)
clat1 = math.cos(CurrLat1)
slat2 = math.sin(TarLat1)
clat2 = math.cos(TarLat1)
delta1 = (clat1 * slat2) - (slat1 * clat2 * cdlong)
delta1 = math.pow(delta1,2)
temp = clat2 * sdlong
delta1 = delta1 + math.pow(temp,2)
delta1 = math.sqrt(delta1)
denom = (slat1 * slat2) + (clat1 * clat2 * cdlong)
delta2 = math.atan2(delta1, denom)
distanceToTarget = delta2 * 6372795

#Now calculations for Target Heading
dlon = TarLong1-CurrLong1
a1 = math.sin(dlon) * math.cos(TarLat1)
a2 = math.sin(CurrLat1) * math.cos(TarLat1) * math.cos(dlon)
a2 = math.cos(CurrLat1) * math.sin(TarLat1) - a2
a2 = math.atan2(a1, a2)
if a2 < 0.0:
	a2 = a2 + (2*math.pi)
targetHeading = math.degrees(a2)

#Calculate heading error for PID controller
headingerror = targetHeading - curr_hdng_deg

# adjust for compass wrap
if headingerror < -180:
	headingerror = headingerror+360
if headingerror > 180:
	headingerror = headingerror-360

# Steering system PID controller
p_gain = (headingerror*0.4)
integral = integral + headingerror*looptime
i_gain = 0.001*integral
# i_gain=0
total_gain=p_gain+i_gain
if distanceToTarget > WAYPOINT_DIST_TOLERANCE:
	if abs(headingerror) <= HEADING_TOLERANCE:
```

```
                    steering_angle=0 # -30 Degrees is extreme left and +30 degrees is
extreme right
                    ser_dc==(0.1667*steering_angle)+26.2
            else:
                    steering_angle = steering_angle + ((total_gain))
                    ser_dc==(0.1667*steering_angle)+26.2

            # Logic to Saturate the duty cycle within operating range
            #21 being extreme left and 31 being extreme right # If heading error is
negative turn servo to left and vice versa
            if ser_dc<=21:
                    ser_dc = 21
            if ser_dc>=31:
                    ser_dc = 31
            PWM.set_duty_cycle(ser_pin,float(ser_dc))
            time.sleep(0.1)
        elif distanceToTarget <= WAYPOINT_DIST_TOLERANCE:
            PWM.set_duty_cycle(esc_pin,float(dc_stop))
            time.sleep(3)
            i=i+1

        #Calculation of loop-time
        t2=time.time()
        looptime=t2-t1

        # Write to file
        f.write("%0.2f %0.1f %0.8f %0.8f %0.8f %0.8f %0.2f %0.2f %0.2f %0.2f %0.2f
%0.4f %0.2f %0.2f %0.2f %0.2f %0.2f %0.2f\n"
%(looptime,i,x,y,TarLat[i],TarLong[i],curr_hdng_deg,targetHeading,headingerror,distan
ceToTarget,speed,yaw_rate,ax,ay,az,magx,magy,magz))

        # Monitor Emergency Stop Button Status
        newstatus=GPIO.input(start_button)
        if newstatus==0:
            GPIO.output(okled_pin, GPIO.HIGH)
            GPIO.output(runled_pin, GPIO.LOW)
            time.sleep(1)
            break
while True:
        PWM.set_duty_cycle(esc_pin,float(dc_stop))
        PWM.set_duty_cycle(ser_pin,float(26.2))
        PWM.stop(esc_pin)
        PWM.stop(ser_pin)
        PWM.cleanup()
        f.close()
```

9.2 Hardware Specifications

9.2.1 Controller Specification

Figure 9.2.1.1: Beaglebone Black Micro-Controller - https://beagleboard.org/black

Hardware Details:

- Processor: AM335x 1GHz ARM® Cortex-A8
- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers
- USB client for power & communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

Software Details:

- OS: Debian / Ubuntu
- Coding: C /C++ / Python

9.2.2 Sensor Specifications

<u>GPS – Global Positioning System</u>

Figure 9.2.2.1: GPS - Garmin 18x - 5Hz - https://buy.garmin.com/en-US/US/p/13195#overview

PHYSICAL CHARACTERISTICS	
Size	Φ 61mm , H=19.5mm
Weight	161.6 grams
ELECTRICAL CHARACTERISTICS	
Input Voltage	4.0 – 5.5 V
Input Current	65mA @ 5.0V
Signal Output Levels	Asynchronous Serial, RS 232
Supported Baud Rates	4800, 9600, 19200, 38400 bps
ENVIRONMENTAL CHARACTERISTICS	
Operating Temperature	-30°C to +80°C
Storage Temperature	-40°C to +90°C
GPS PERFORMANCE	
Reacquisition Time	< 2 seconds
Update Rate	5 Hz
Accuracy: GPS Standard Positioning Service (SPS)	< 15 mtrs. 95%
Accuracy: Wide Area Augmentation System (WAAS)	< 3 mtrs. 95%

Table 9.2.2.1: GPS - Garmin 18x - 5Hz Specification - http://static.garmin.com/pumac/GPS_18x_DoC.pdf

<u>Commonly Used Output Data</u> – Latitude, Longitude, Hemisphere, GPS Fix Type, No. of Satellites, Speed

<u>Output Type</u> – NMEA Sentences or Binary Output

IMU – Inertial Measurement Unit

Figure 9.2.2.2: IMU - Redshift Labs UM7 - https://www.redshiftlabs.com.au/sensors/um7

PHYSICAL CHARACTERISTICS	
Dimensions	27mm x 26mm x 6.5mm
Weight	11 grams
ENVIRONMENTAL CHARACTERISTICS	
Operating Temperature	-40°C to +85°C
PERFORMANCE	
Max. Binary Packet Output Rate	255 Hz.
Max. NMEA Packets Output Rate	100 Hz
HEADING SPECIFICATIONS	
Static Accuracy – Pitch and Roll	± 1 Degree *
Dynamic Accuracy – Pitch and Roll	± 3 Degree *
Static Accuracy – Yaw or Current Heading	± 3 Degree *
Dynamic Accuracy – Yaw or Current Heading	± 5 Degree *
Repeatability	0.5 Degree *
Resolution	< 0.01 Degree *
GYROSCOPE SPECIFICATIONS	
Rate Noise Density	0.005 deg/s/rtHz *
Total RMS Noise	0.06 deg/s-rms *
Dynamic Range	± 2000 Deg/s
Non-Linearity	0.2%
ACCELEROMETER SPECIFICATIONS	
Rate Noise Density	400 μg / rtHz *
Dynamic Range	± 8 g

MAGNETOMETER SPECIFICATIONS	
Initial Scale Factor Tolerance	± 4%
Initial Bias Tolerance	± 300 µT
Dynamic Range	± 1200 µT
ELECTRICAL SPECIFICATIONS	
Input Voltage	5 V
Current Consumption	50mA @ 5V
Signal Output	3.3V TTL UART, 3.3V SPI
Default Baud rate	115200 bps

Table 9.2.2.2: IMU - Redshift Labs UM7 - Technical Specification - https://www.redshiftlabs.com.au/files/index/download/id/1471348551/

* Data taken from catalog, actual parameters depend on installation and other operating conditions. Always perform tests on sensors to analyze data before using it for experimentation. Other specs. can be taken from the datasheet

Commonly Used Output Data – Euler Angles (Yaw), Gyro Data, Accelerometer Data

Output Type – NMEA Sentences or Binary Output

9.2.3 Test Vehicle Specification

Figure 9.2.3.1: Test Vehicle - https://www.horizonhobby.com/desert-buggy-xl-e--1-5th-4wd-eletric-rtr---black-los05012t1

- Vehicle Type – 1/5 Scale RC Car, 4WD, Electric, 13.8 Kg. (30.5 lbs.), 844 x 501 x 308mm
- Motor – Non-Sensor Brushless Type, 800Kv, built in 160A Electronic Speed Controller ESC, Motor Gear Ratio – 3.33:1
- Drivetrain – 4WD, Final Drive Ratio – 12.81 : 1
- Steering Servo – Torque: 30 kg-cm @ 6.0V
  Response: 0.27 Sec / 60 Degree (On Dirt)