$\mathcal{U}$nmanned $\mathcal{V}$ehicle $\mathcal{S}$ystems $\mathcal{D}$evelopment

Final Project Report

Submitted by:

Kashish Dhal
Harsh Soni
Rohit Nagvenkar
Gopanraj Patel
Yash Gogirwar

(Team 2)

Submitted to:

Dr. Brian Huff
Dr. Manfred Huber
Dr. Kambiz Alavi

(Faculty)

A report submitted towards the partial fulfillment for the
Unmanned Vehicle Systems Development Course

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Objectives

Objective of this course is to give Introduction to the technologies needed to create an UVS (Unmanned Vehicle System) and to get hands on exposure on integration of these technologies (embodied as a set of sensors, actuators, and computing and mobility platform sub-systems) into a functioning UVS through team work. UVS could be designed to compete in a student competition sponsored by various technical organizations or to support a specific mission or function defined by the instructors.

Moreover, this course gives hands on exposure to the technologies and engineering methods used to develop and deploy Unmanned Vehicle Systems. To learn about cool technology from the perspective of multiple engineering disciplines like Computer Science, Mechanical, Aerospace, and Industrial Engineering Departments. Project experiences that would typically fall outside his/her main area of study challenging the student to explore the inherently multi-disciplinary nature of todays complex engineered systems.[1]

In this project we developed a UGV (unmanned ground Vehicle). An UGV is a vehicle that operates while in contact with the ground and without an onboard human presence. UGVs can be used for many applications where it may be inconvenient, dangerous, or impossible to have a human operator present. Generally, the vehicle will have a set of sensors to observe the environment, and will either autonomously make decisions about its behavior

or pass the information to a human operator at a different location who will control the vehicle through teleportation.[2]

We may find that UGV systems are used in applications in the fields of civil and military industry. It can be used by emergency services such as fire brigades, ambulances, police, etc Providing a great support for a wide variety of tasks, including: assistance to disabled people, fumigation, harvesting, transporting, patrol monitoring and detection, investigation, exploration and inspection at tunnels, buildings, etc

UGV systems are flexible robotic platforms developed to provide powerful multipurpose mobility support. Depending on their size and purpose, they can be extremely portables and they can even be able to avoid obstacles autonomously. In order to increase their capabilities

with latest technological advances, incremental improvements have been made in hardware, software, payloads, structure

In conclusion, UGV systems are very versatile platforms that permit operations in conditions where there are certain human risks, as in missions with biological, radiological and chemical emergencies or with risks of explosions, etc We cannot forget that Veronte Autopilot installed in Unmanned Ground Vehicles, improves their automation and safety in any application.[3]

Chapter 2

Waypoint Navigation in Outdoor Environment

First task of this course was to remotely control the rover using a radio and then calibrate it to perform the waypoint navigation with GPS signal using autopilot in Mission Planner. The hardware setup for this task is presented in figure 2.1.

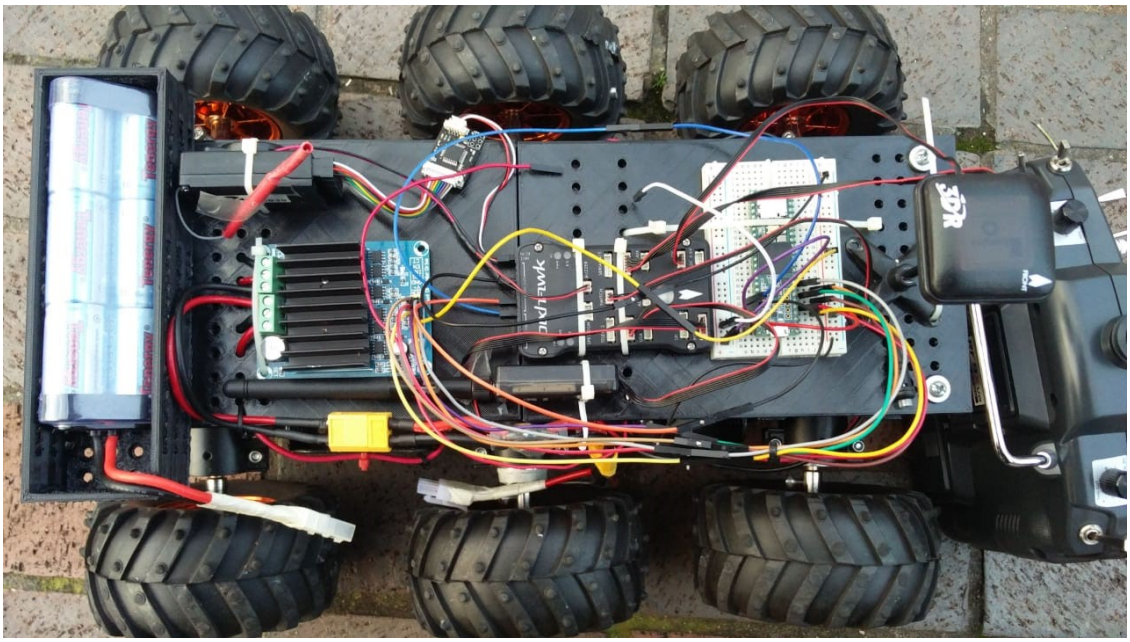

Figure 2.1: Initial setup of hardware

2.1 Rover Components

To accomplish the first task of waypoint navigation using GPS, the following components were required:

1. 6-wheel drive thumper
2. Dual channel H bridge
3. Nickle-metal hydride battery (7.2V)
4. Teensy
5. Teensy battery (5V)
6. Pixhawk
7. RC transmitter and receiver
8. GPS
9. Telemetry
10. Power module

2.2 CAD Model

A six-wheel drive thumper used for this project consists of individual suspension for each of its wheel, which allows overall traction on any uneven terrain. The wheels of the chassis are equipped with 6 centred shaft DC motors. The two middle motors are equipped with rotary encoders which keeps track of position and speed of the rover. The motors are powered using dual channel H-bridge through 7.2V nickle-metal hybrid battery.

With the aid of additive manufacturing, a battery box was fabricated using ABS and placed towards the rear end of the rover. Most of the components were mounted on the chassis using zip ties. Also, to increase the cushioning effect a platform was 3D printed using Ninja flex material and was placed on the chassis in order to support the components. The CAD model of the rover is presented in figure 2.2.

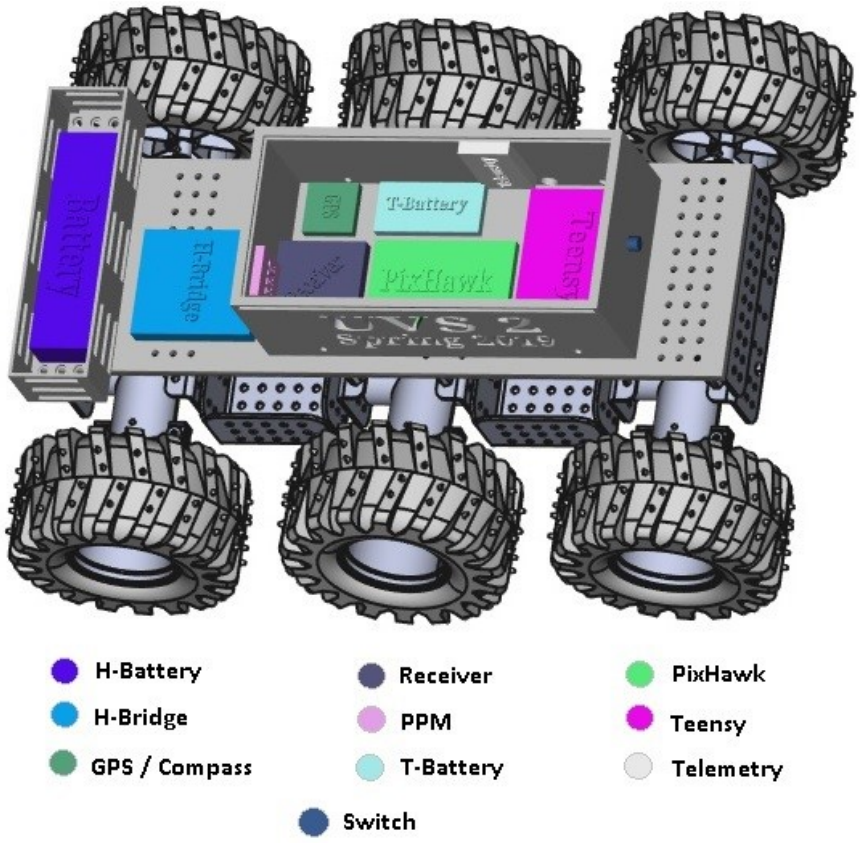

Figure 2.2: CAD Model of Rover with Hardware Mounted

2.3 Pixhawk

Pixhawk is a controller which comes an embedded GPS, compass, IMU and Gyros, capable of providing raw and fused navigation data. It was powered using NiMH battery supplied through power module to provide output voltage of 5V. It is then configured using mission planner by connecting through USB. Latest firmware 3.5 was automatically downloaded and installed but since the version used was pixhawk 2, old compatible firmware 3.4.2 was installed for appropriate calibrations.

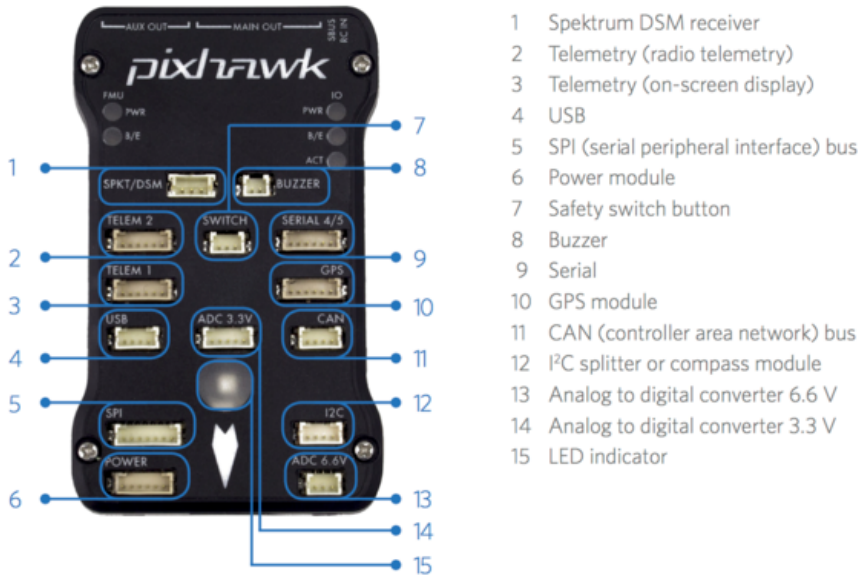

Figure 2.3: Pixhawk connections

2.4 Remote Controller

TURNIGY 9x remote controller(RC) was used for this task, which is equipped with overwhelming array of setting. The transmitter and receiver are configured before it transmits signal. To establish connection, the transmitter and receiver are binded together with specific procedure by transmitter with is switched off and the throttle being all the way down. The PPM signal from the transmitter of the RC is converted to PWM signal using Teensy 3.5 as presented in figure 2.4. The converted PWM signal is then used to control the power output of the motors using dual channel H bridge. The gear switch on the RC is also configured with Safety Override System in case of an emergency.

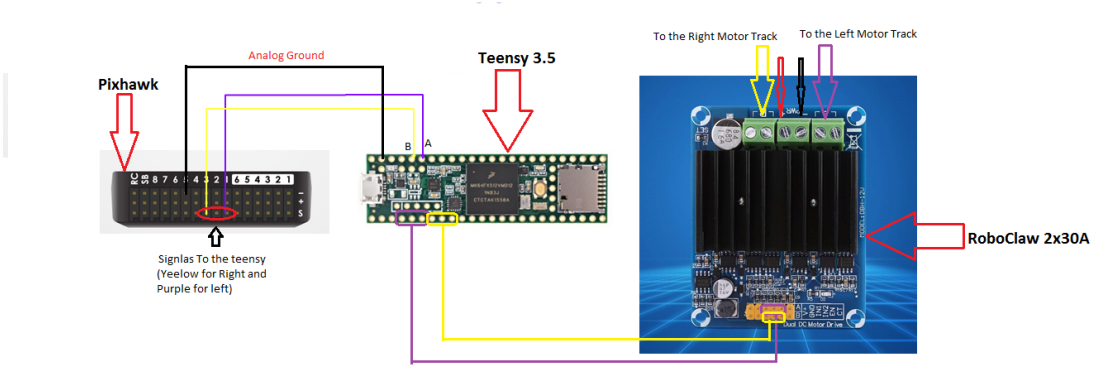

Figure 2.4: Circuit Diagram

2.5 Calibrations Required

After the establishment of the communication between Pixhawk and RC, the next step was to calibrate and tune the hardware to get the desired output for commanded input as required by the user. For this purpose, Accelerometer, Compass and RC were calibrated.

2.5.1 Accelerometer Calibration

With the help of the gravitational effect, accelerometer is calibrated through mission planner. For calibration, the rover is positioned on horizontal, vertical and on each side of the rover. A snapshot from the Mission Planner, displaying the procedure of calibration is presented in figure 2.5

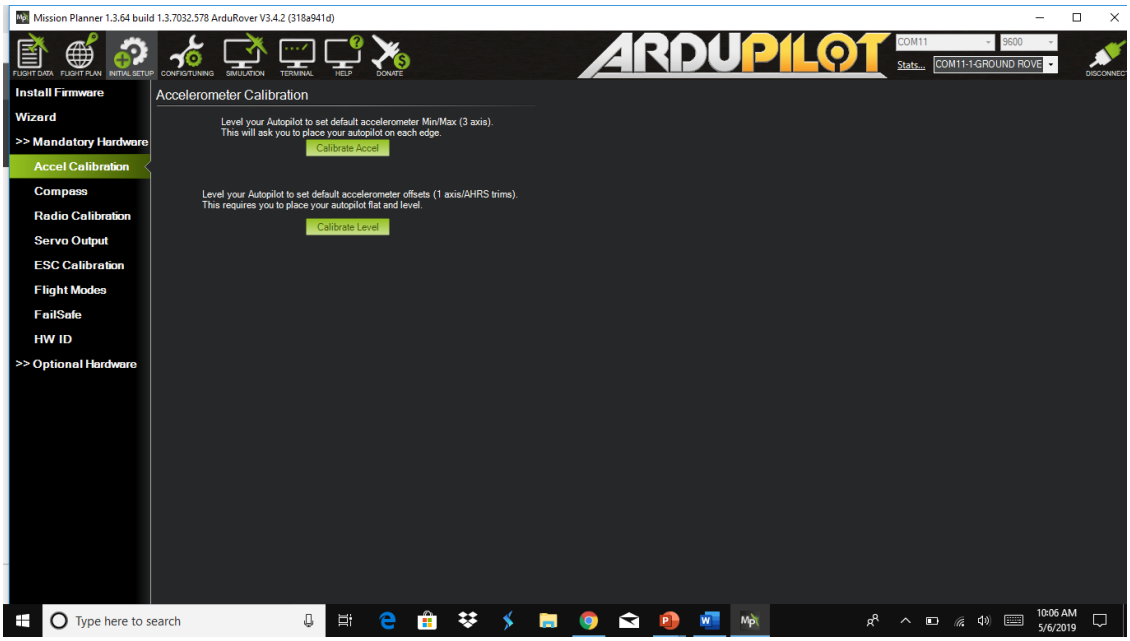

Figure 2.5: Accelerometer calibrating on mission planner

2.5.2 Compass Calibration

The rover is equipped with two compass, one of which is a Gyro based compass, present inside the Pixhawk and an external compass which was mounted with GPS. The calibration can be done using live or on-board calibration. In live calibration, a window pops up showing the current state of the calibration. The rover is rotated along all the 3 axes to hit each of the white dots to complete the calibration. The more accurate method is the on-board method which is also done on the flight controller. The rover is rotated along the 3 axes until it gathers all the calibration data. A snapshot from the Mission Planner, displaying the procedure of calibration is presented in figure 2.6

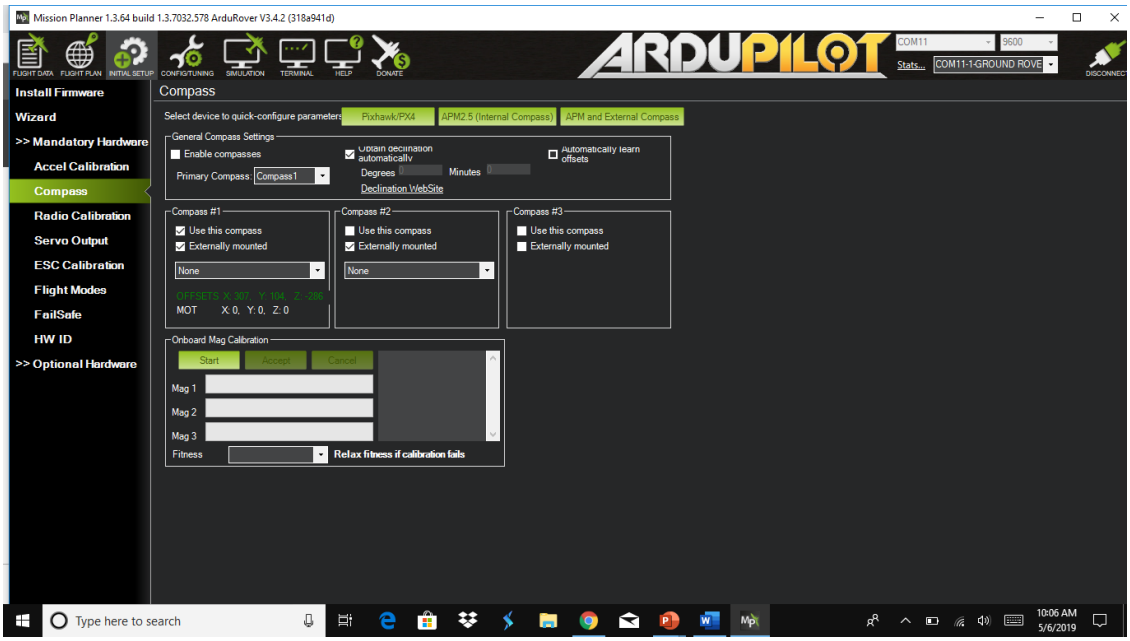

Figure 2.6: Compass calibrating on mission planner

2.5.3 Radio Calibration

After binding transmitter and receiver of the RC, the calibration is done by moving the control sticks of yaw, pitch, roll, throttle through their full range and record the maximum and minimum range. The gear switch is configured for safety override system. For ease, the position of the throttle was reversed from the servo output. A snapshot from the Mission Planner, displaying the procedure of calibration is presented in figure 2.7

2.6 Motor Test

For confirmation, another way to test the motors polarity is by using motor test option. For skid steer, Motor C and Motor D were assigned in the mission planner as shown in figure 2.8. By increasing the throttle to 50%, all motors were tested at once. Secondly, in another test, where individual side of the motors were commanded it, it was found that Motor D was running even with Motor C commanded. While debugging, it was found that inside the script of the teensy code, for the right side of the wheels, the command given was (pwm_cmdl)

instead of (pwm_cmdr). The signals from the left side of the wheels were provided to both side of the wheels. Afterwards, the code was corrected as presented in figure 2.9.

Figure 2.7: Radio calibration on mission planner

Figure 2.8: Motor Test on mission planner

```
128
129     // for RIGHT side wheels,
130     if (pwm_cmdr < 0)
131     {
132       analogWrite(InB1,0);
133       analogWrite(InB2,abs(pwm_cmdl));
134       digitalWrite(enabB,HIGH);
135     }
136     else if (pwm_cmdr > 0)
137     {
138       analogWrite(InB2,0);
139       analogWrite(InB1,abs(pwm_cmdl));
140       digitalWrite(enabB,HIGH); //writing pwm to motor
141     }
142     else
143     {
144       analogWrite(InB1,0);
145       analogWrite(InB2,0);
146       digitalWrite(enabB,HIGH);
147     }
148
149     delay(500);
150   }
151
```

Figure 2.9: PPM to PWM teensy code

2.7 Skid Steering

For rover to perform skid steering, the parameters can be set or enable from parameter list tree by changing SERVO1_FUNCTION = 73 to throttle left and SERVO3_FUNCTION = 74 to throttle right as presented in figure 2.10

2.8 Teensy

The signal pin 1 of the output on the Pixhawk is connected to the Pin A7 of the Teensy and signal pin 3 of the output is connected to the Pin A8 of the Teensy. Also, a common ground is established between Pixhawk, Teensy and the motor controller so that the voltages are measured with a common reference point. The pin layout is presented in figure 2.11.

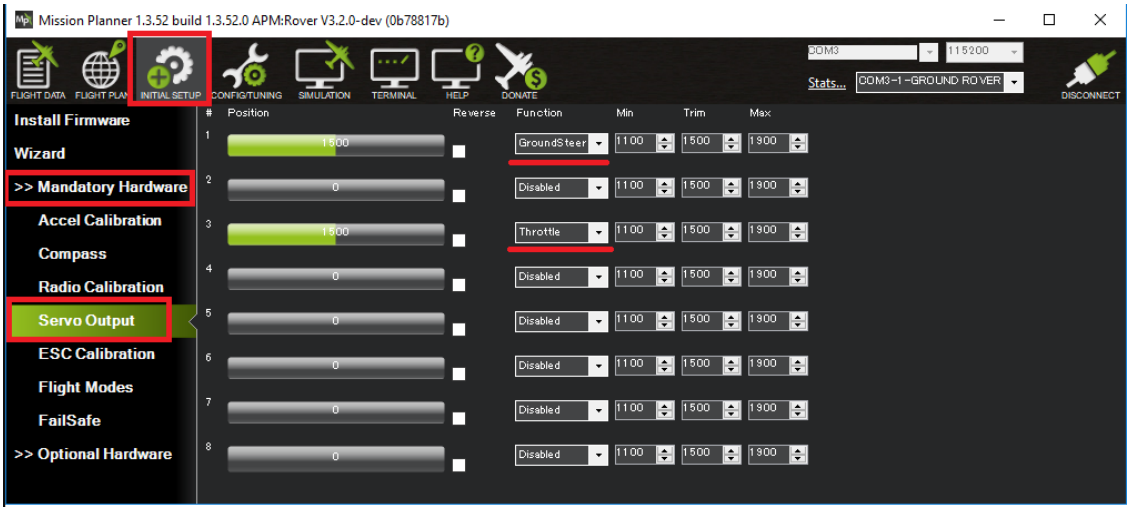

Figure 2.10: Skid Steering

Teensy 3.5 Pin Number	Pin on the Dual Channel H-Bridge
2	In2 – A
3	In1 – A
4	EN - A
5	In1 – B
6	In2 – B
7	EN - B

Figure 2.11: Pin Connections from Teensy to H-bridge

Initially, as per the code provided, the rover was set to a frequency of 50Hz as shown in figure 2.12, due to which, the motor operates in discontinuous mode and the rover would make fast random movements that were difficult to control by the system. So, to overcome this jerky movement of the rover, the PWM frequency was increased to 20kHz.

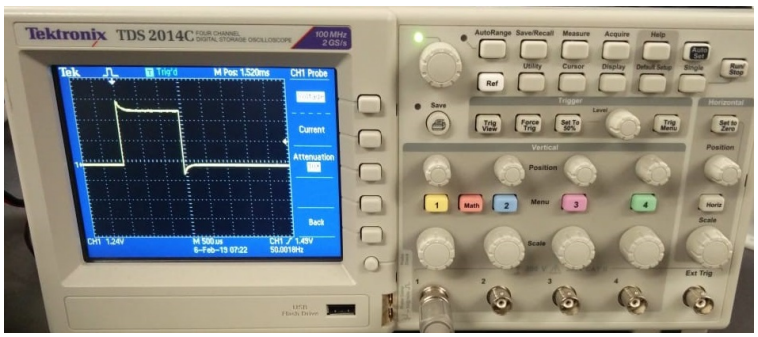

Figure 2.12: PWM frequency from H-bridge

2.9 H-bridge

An H-Bridge is an electronic component which regulates the voltage and/or switches the polarity as instructed. The motor controller used in this project can withstand very high current overload and since its a dual channel, it can supply constant voltage on both side of the rover. Figure 2.13 shows a picture of H-bridge.

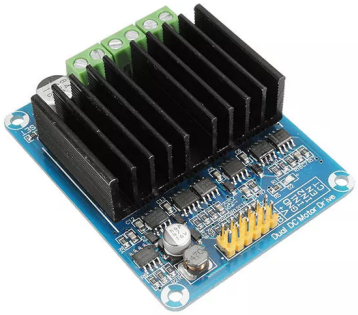

Figure 2.13: 50A dual Channel H-Bridge

2.10 Telemetry

For remote access of the pixhawk, telemetry was configured. To avoid the signals emitted from the telemetry to interfere with another unknown device, the frequency was set to 50Hz as presented in figure 2.14.

Figure 2.14: Telemetry Kit

2.11 Manual Control

The rover can also be controlled in the manual mode without using the GPS for its current position. It can be steered by controlling the output of the DC motors by steering and throttle sticks on the RC. The transmitter has multiple mode for different use of control sticks. For our rover, we used MODE 2 in TURNIGY 9x. Using the gear as a position switch, best way to setup one switch position on manual mode and other on auto mode.

2.12 Setting Waypoints from Mission Planner

With multiple options for custom imagery, Google maps was used for the mapping. When the GPS position is locked on the ground station or the pixhawk glows solid green led, it provides the current state of the rover in the map. Current heading along with the rover yaw can be seen in flight data. For rover to complete the task, it needs multiple waypoints between the start and goal position with enough space in between so that no two radii of waypoints interfere with one another. If the waypoint radius is too large or the waypoint is far near, the rover keeps moving in circular motion at its current waypoint. 17 waypoints with default WP radius of 2m was provided remotely using telemetry with enough space in between the two waypoints.

Chapter 3

Waypoint Navigation in Indoor Environment

In the previous task GPS was used to navigate. GPS signals were received by the Pixhawk and an on-board autopilot logic inside the Pixhawk drive the rover. But in this task, the objective was to do waypoint navigation in indoor environment where GPS signal is not accessible. For this task, a dead reckoning Simulink model was provided which uses odometry for navigation and as per the logic coded in the Simulink the rover can be driven. So, a on-board computer was needed for this task. In navigation, dead reckoning is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course.[4] An on-board computer (Intel NUC) was setup on the chassis of the rover and MATLAB/SIMULINK and ROS were installed on the Ubuntu OS running on Intel NUC. Detailed procedure for the setup is explained in the later sections.

3.1 Mounting New Components

Although only odometry was used in this task, however, the design was modified to accommodate the following components:

1. Li-Po battery
2. Step-up Transformer
3. Intel NUC
4. LIDAR
5. Camera
6. Sonar

The rover design used in previous was not able to accomodate NUC and LiDAR. So, a new design was proposed, and their positions were decided in such a way that the center of mass of the vehicle is as close to centroid of the platform as possible. The centroid helps the rover to find parameter necessary for angular motion. The rover consists of 4 layers in total. The bottom most layer contains the power distribution board to power six motors (three on each track), which gets the input signal from Dual H-Bridge, mounted on 2^{nd} layer of the rover. Power to rover wheels is supplied with Nickle Carbide battery through this H-bridge and besides this, a step-up transformer is mounted to power NUC. Followed by this, NUC was installed close to the center of the rover for equal weight distribution and Teensy was installed at the other end of the platform.

On the 3rd layer, the sonar was mounted at the front, followed by camera on the top most layer for the vision processing, pixhawk was located close to the center, GPS and 4 cell Li-Po battery were located near the rear end of the platform which is used to power NUC. A solid platform was 3D printed and installed on the 4th layer at an elevated height to mount the LIDAR on it so that it could get clear 360-degree view without any obstruction at the center. The whole setup is as shown in the figure 3.1

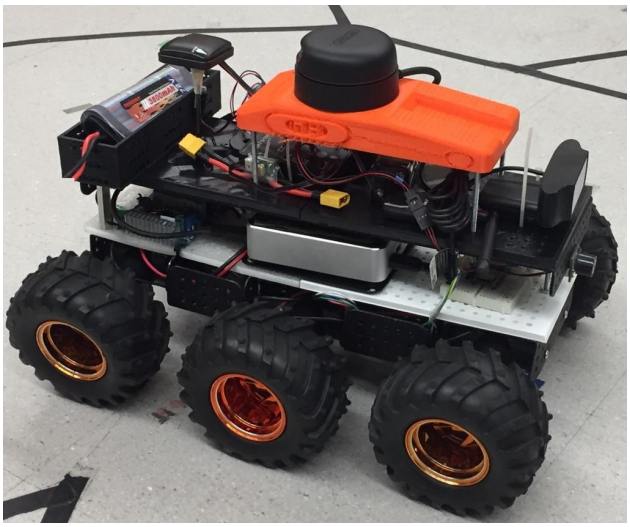

Figure 3.1: Rover after mounting additional hardware

Afterwards, NUC was powered and Ubuntu 16.04 OS was installed. Once the OS was installed, other required software, such as MATLAB/SIMULINK, ROS Kinetic, ROS Workspace, Mavros, ROSserial and Arduino IDE for Teensy were then installed. After VNC server was installed to remotely access the NUC from ground station. A screenshot of the UltraVNC Viewer together with the settings used is presented in figure 3.2

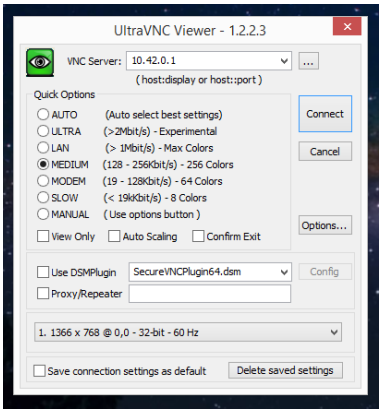

Figure 3.2: Accessing NUC via VNC remote access

Also, the Pixhawk was not required in this task so Teensy code and wiring was also modified as presented in figure 3.3

Teensy 3.5 Pin Number	Pin on the Dual Channel H-Bridge
2(for us 5)	In2 – A
3(for us 6)	In1 – A
4(for us 7)	EN - A
5(for us 8)	In1 – B
6(for us 9)	In2 – B
7(for us 10)	EN – B

Teensy 3.5 Pin Number	Encoder Wire	Wire Color
30	enco_R_B	White wire on the right encoder
29	enco_R_A	Yellow wire on the right encoder
35(for us 2)	enco_L_A	Yellow wire on the left encoder
36(for us 3)	enco_L_B	White wire on the left encoder

Figure 3.3: Rearranged Wiring of Teensy

3.2 Parameter Estimation

Since rover is not a point object, it has some parameters associated with it. To control it using an on-board computer, it is required to determine the system parameters, encoder counts per unit length and a parameter b which are the functions of dimensions of the platform and wheel radius and some other indeterminable factors.

3.2.1 eTick Parameter Estimation

When the experiments were conducted, the encoder counts were found to be inconsistent for same run distance. For instance, the encoder count of the right wheel was ranging from 8000 counts to 16000 counts and the encoder counts of the left wheel was ranging from 8000 counts to 20000 counts. After analyzing the encoder count data for the left and right wheels as shown in figure 3.4, it was concluded that both the encoders were damaged. Later, the encoders were dismantled from the motor to analyze them visually and it was found that the infrared sensor on the left wheel was loose and the encoder wires of the right encoder were burnt due to which the counts were inconsistent. This is presented in figure 3.5.

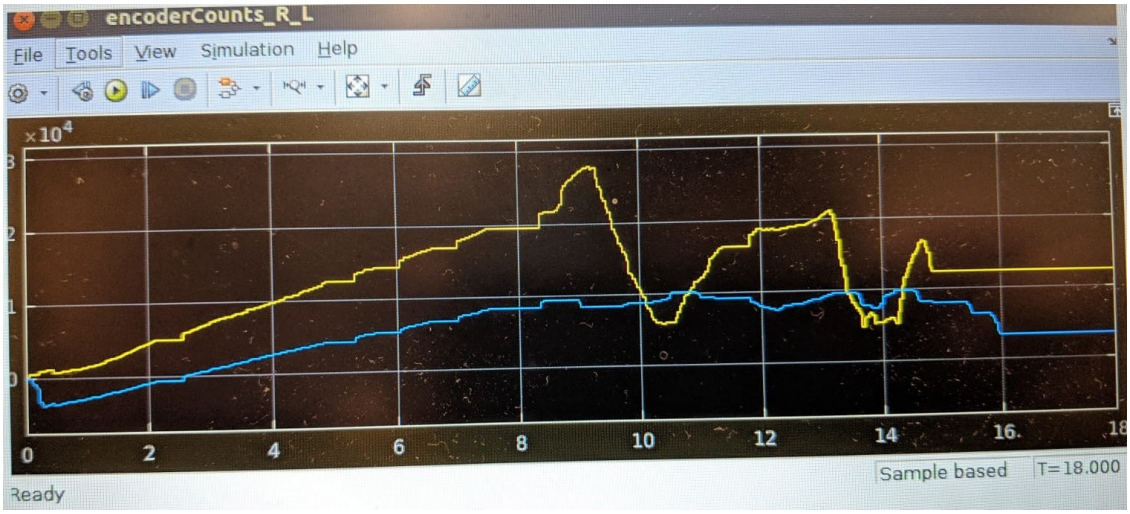

Figure 3.4: Encoder Counts for Faulty Encoders

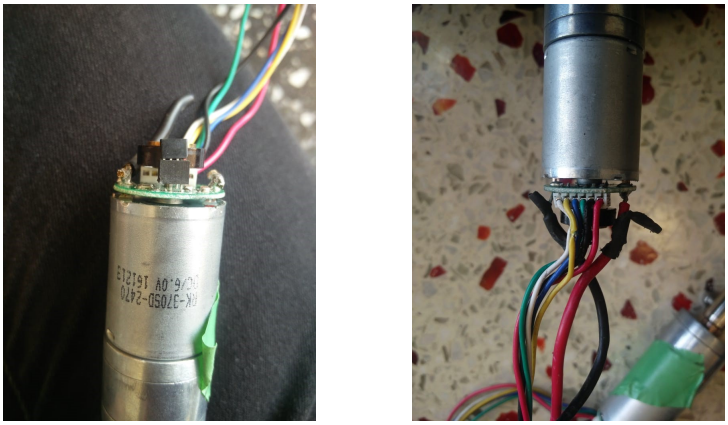

Figure 3.5: Faulty Encoders

The encoders were then replaced and this issue was resolved as is evident from the table 3.6. The encoder count for both the left and right wheels were published through the publisher block to the ROS topics and then accessed in SIMULINK as presented in figure 3.7. An average encoder count over ten iterations was taken for the sample time of 5 seconds and then used to calculate the value of parameter, eTick. It was calculated using equation 3.1 and was found to be 1050.

$$eTick = \frac{encoderCount}{distTravelled} \tag{3.1}$$

LEFT ENCODER COUNT	RIGHT ENCODER COUNT
8187	11446
11971	7471
20052	16556
10736	9712
9166	7989
9177	7933
9063	8361

Figure 3.6: Measured Encoder Counts for 5 seconds after Encoder Replacement

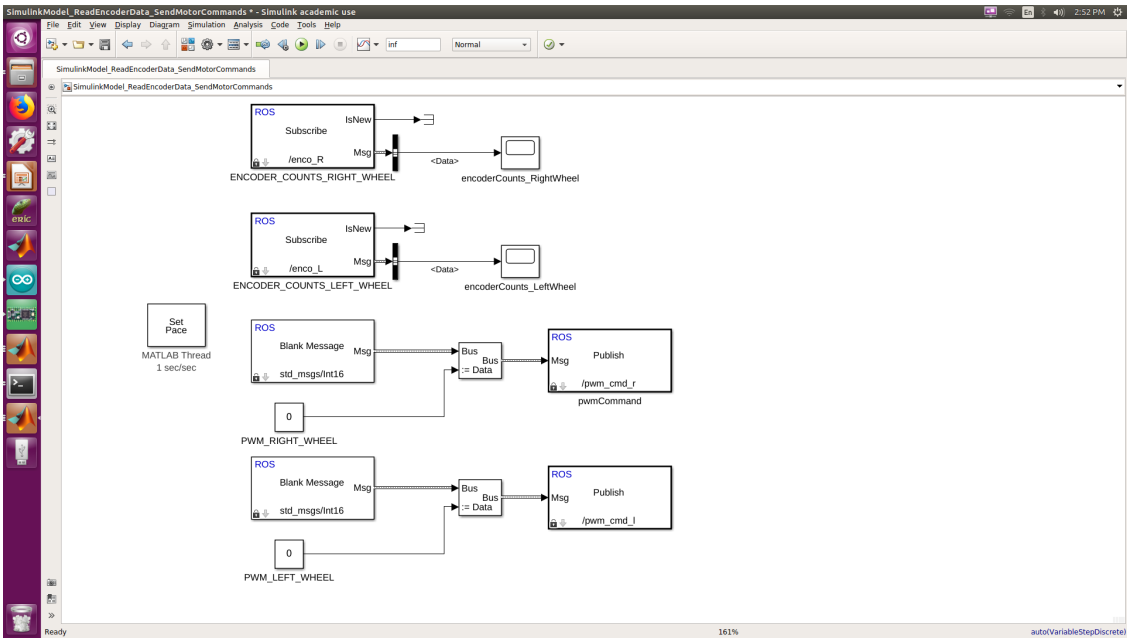

Figure 3.7: Simulink block to read the encoder counts

3.2.2 b

To estimate the value of b, the rover was commanded to rotate without any translation for a period of 2 seconds and the resulting angular displacement of the platform was recorded. Then equation 3.2 was used to calculate b which turned out to be 0.2993384 m

$$b = \frac{\Delta eCountL - \Delta eCountR}{\Delta\theta \cdot eTick} \tag{3.2}$$

Then, to verify, the distance between the middle wheels was measured and found to be 0.29 m which was close to our computed parameter value.

3.2.3 Lookup table

After plotting the PWM (ranging from -250 to 250) with the angular velocity,the dead band was found to be between -50 to 50 PWM cycle.

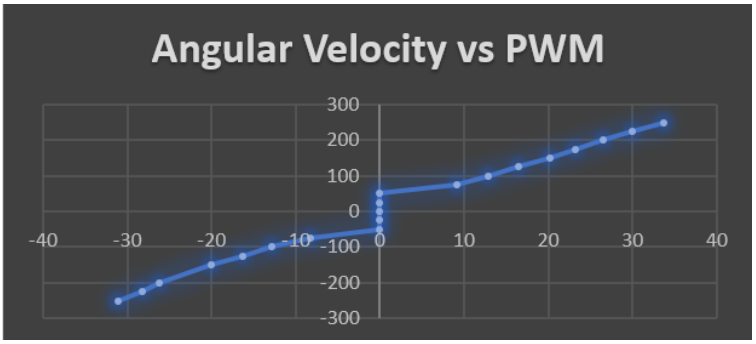

Figure 3.8: Angular velocity vs PWM (values from lookup table)

3.3 PID Tuning

Next step was to tune the PID gains. As the rover is a highly non-linear system it is impossible to find perfect values for the gains. However, some chosen gains may work for some defined local conditions. Using hit and trial PID gains for straight line motion and turning were approximated and then the rover was tested for these values on the same surface and chose the values which yields the best performance. Some choices for PID gains are presented in table 3.9.

Kp1`	10	10	10	10
Kp2	40	40	45	50
kI1	0.001	0.001	0.0001	0.0001
kI2	0.002	0.001	0.0001	0.0001
kD1	0	0	0	0
kD2	2.5	3.5	6	5.7

Figure 3.9: PID Tuning

Chapter 4

Obstacle Avoidance

Until the previous task, the vehicle is able to navigate through the waypoints using odometry. Now the next task is to avoid the obstacle on the way. Sonar and LIDAR were to be used to detect the obstacles and then a control algorithm was required to steer the vehicle away from the obstacle. This procedure is explained in detail in the following sections.

4.1 Sonar

The Sonar was mounted near the top front end of the rover as presented in figure 4.1 and was connected via pin no. 22, powered through Teensy. After the connection was established, the sonar data was published to ROS, msg_sonar as presented in figure 4.2. During the data analysis, a delay was found in the data published through Teensy to the ROS. To solve that issue the connection was changed from digital to analog connection as shown in the line 296 in the code in figure 4.3.

Figure 4.1: Position of Sonar

```
 97
 98  //sonar
 99  int sonar_pin = 22;
100  long duration;
101  std_msgs::Int16 msg_sonar;
102  ros::Publisher distance_sensor("distance_sensor", &msg_sonar);
103
```

Figure 4.2: Sonar Data Published to ROS

```
288  void loop()
289  {
290    msg_R.data = enco_R_pos;
291    msg_L.data = enco_L_pos;
292
293    enco_R.publish(&msg_R);
294    enco_L.publish(&msg_L);
295
296    duration = analogRead(sonar_pin);
297    msg_sonar.data = (duration * 5) / 25.4;
298    distance_sensor.publish(&msg_sonar);
299
300    nh.spinOnce();
301    digitalWrite(led, HIGH);
302    delay(10);
303
304  }
305
```

Figure 4.3: Modified Teensy Code

4.2 LIDAR Mapping

The LIDAR is powered directly from the NUC and is mounted at the highest point of the rover so that it does not obstruct while mapping. While running the LIDAR with the default speed of 600 rpm, it created a lot of vibrations in the rover. So, to avoid unnecessary

vibrations, the scanning speed of the LIDAR was reduced enough to scan the environment at lower rpm.

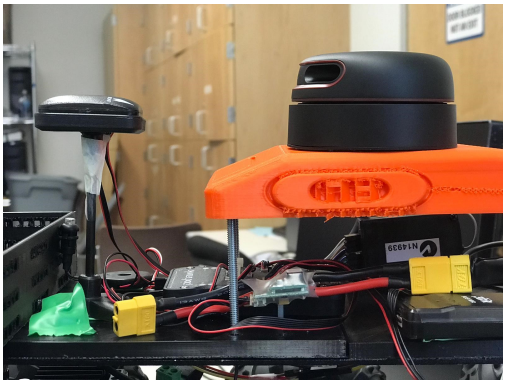

Figure 4.4: Position of LIDAR

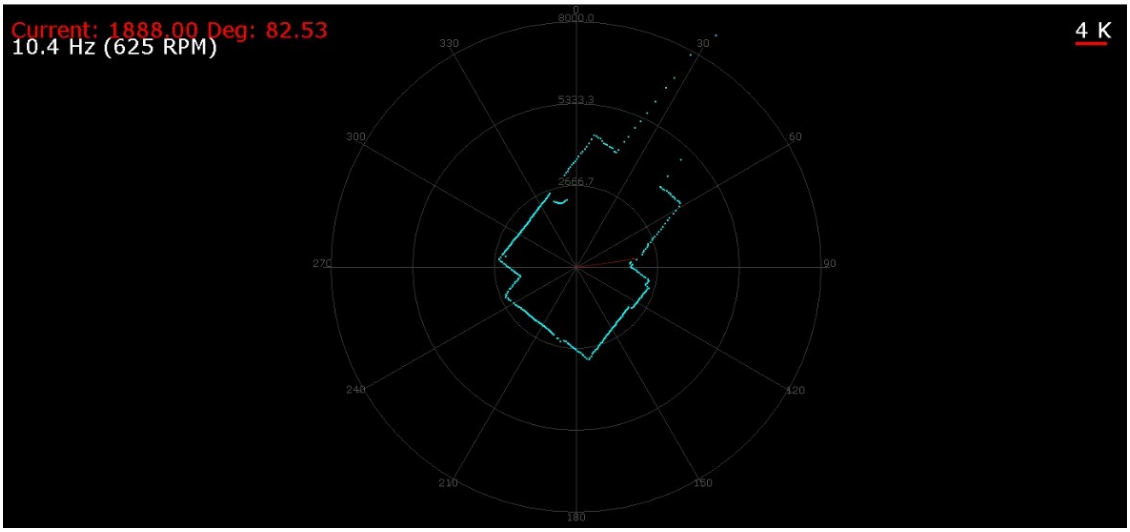

Figure 4.5: Point Cloud Generated by LIDAR

4.3 PCB

Due to excessive use of the breadboard, the connectors got loose inside the breadboard and the jumpers couldnt provide continuous supply of the data due to which the rover would act different from the actual path in the Simulink. So, we printed PCB and soldered the teensy due to which the rover would perform task close to its actual path.

4.4 Logic behind obstacle avoidance

The rover view was divided into 3 sections, one for sonar module which is in the dead center and the rest at the sides left or right for the LIDAR. The obstacle avoidance was done by changing the current heading angle of the rover. So suppose the sonar which is in the dead center sees the obstacle then the heading angle will change either negatively or positively and correspondingly the rover will move either left or right. Finally, if the LIDAR sections see the obstacle then the heading angle will change accordingly depending on the which section sees it, (+1 or -1).

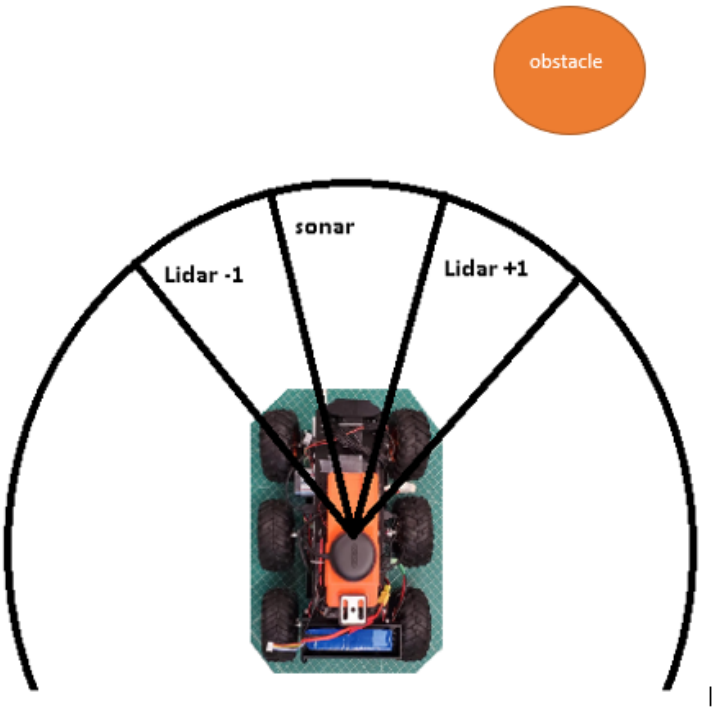

Figure 4.6: Area Scanned by LIDAR and SONAR

Chapter 5

Sensor Fusion

5.1 Extended Kalman Filter

Previously, the encoder readings were used (Dead reckoning) to calculate current position and heading of the rover with accuracy enough to reach the desired position of the rover. However, the accuracy of the rover usually decreases over time if its run for longer distance due to accumulation of error over the period of time. Due to the decrease in the accuracy, the rover shifts its orientation from the inertial frame of reference and with the passage of time with drift continues to increase. So, to maintain the accuracy, additional data from the sensors like compass, IMU, GPS, etc. can be used fused together with Extended Kalman filters[5] to get improved localization.

As described earlier, the rover is highly non-linear, so an Extended Kalman Filter was used to fuse the compass data with odometry. The EKF works by transforming the nonlinear models at each time step into linearized systems of equations. In a single-variable model, we can linearize the system using the current model value and its derivative, the generalization for multiple variables and equations is the Jacobian matrix. The linearized equations are then used in a similar manner as compared to regular Kalman filter.

The system dynamics for the rover are given by:

$$x_{k+1} = x_k + T_s V_k \cos(\theta_k) \tag{5.1}$$

$$y_{k+1} = y_k + T_s V_k \sin(\theta_k) \tag{5.2}$$

$$\theta_{k+1} = \theta_k + T_S \Omega_k \tag{5.3}$$

$$V_{k+1} = \frac{r}{2}(\omega_{L,k} + \omega_{R,k}) \tag{5.4}$$

$$\Omega_{k+1} = \frac{r}{b}(\omega_{L,k} - \omega_{R,k}) \tag{5.5}$$

Jacobian for the state transition matrix is given by:

$$F_{k+1} = \left.\frac{\partial f}{\partial x}\right|_{x_k, u_k} = \begin{bmatrix} 1 & 0 & -T_s V_k \sin(\theta_k) & T_s \cos(\theta_k) & 0 \\ 0 & 1 & T_s V_k \cos(\theta_k) & T_s \sin(\theta_k) & 0 \\ 0 & 0 & 1 & 0 & T_s \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.6}$$

Measurement Matrix is given by:

$$\begin{bmatrix} x_{GPS} \\ y_{GPS} \\ \theta_{\text{Mag}} \\ V_{GPS} \\ \Omega_{Gyro} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ V_k \\ \Omega_k \end{bmatrix} \tag{5.7}$$

However, for this task GPS was not used so our measurement matrix reduces to:

$$\begin{bmatrix} \theta_{\text{mag}} \\ \Omega_{Gyro} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ V_k \\ \Omega_k \end{bmatrix} \tag{5.8}$$

5.2 Accessing the sensor data from Pixhawk

Previously, the GPS was mounted in front of the Li-Po battery. The external compass present inside the GPS gets fluctuated as soon as it comes across any magnetic field (generally generated more by alternating current flowing through wire). As current is flowing through DC supply of Li-Po battery, to get stable values of compass, we decided to move the position of GPS just behind the camera where its not likely to get affected because there wont be any magnetic field generated around it. Fig below shows the current position of GPS.

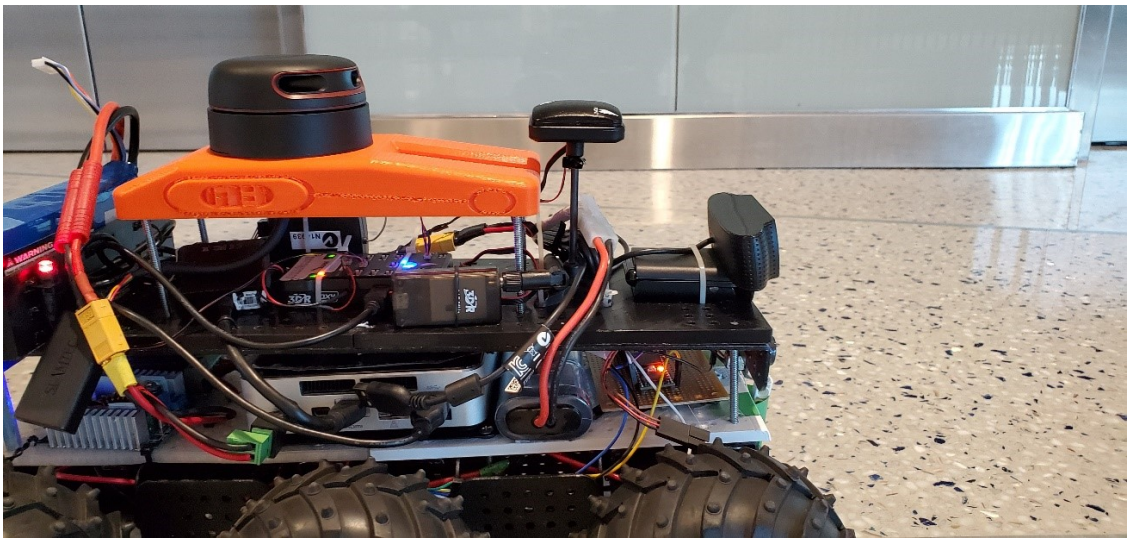

Figure 5.1: New Position of GPS

The internal and the external compass were calibrated as per the instructed procedure.

The rover was repeatedly run for same 4 waypoint for multiple loops without using the data from the filter and the drift in its orientation was negligible. Later, both the compass data were fused with the dead reckoning and it was found that the rover always deflects from its waypoint. Different trials used to measure the compass data at different positions (keeping the rover stationary) using /mavros/global_position/compass_hdg and /mavros/imu/mag command but the ROS always showed different values of the compass and IMU for same position of the rover. Another issue which we found with the compass data was, it always used to fluctuate, i.e. the readings were not consistent at all. So, to get stable values from the compass, we disabled the internal compass from the pixhawk and used only the data from the external compass. Later on, it was found that the reason for this was changing magnetic field. To resolve this issue, then only Gyro was used in the sensor fusion.

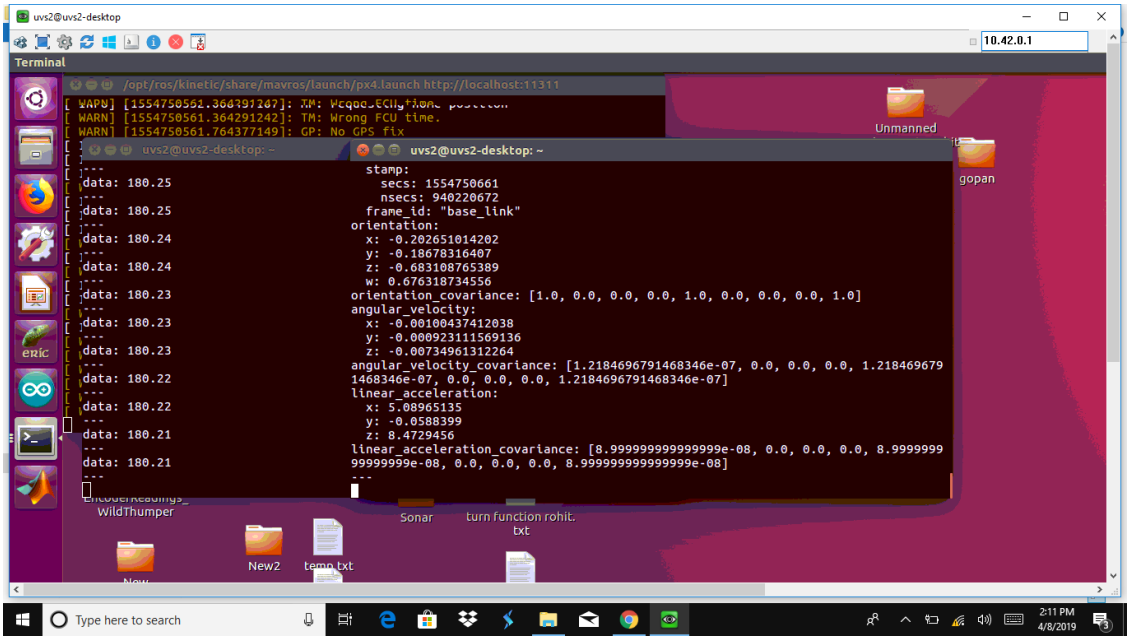

Figure 5.2: Different and inconsistent compass and IMU value for same position

To set the rover's initial position with zero compass value, the value of current compass was stored in persistent variable and subtracted from theta published by compass as presented in figure 5.3. So now, every time the rover was moved to any of its start position, the compass count will always start with zero value.

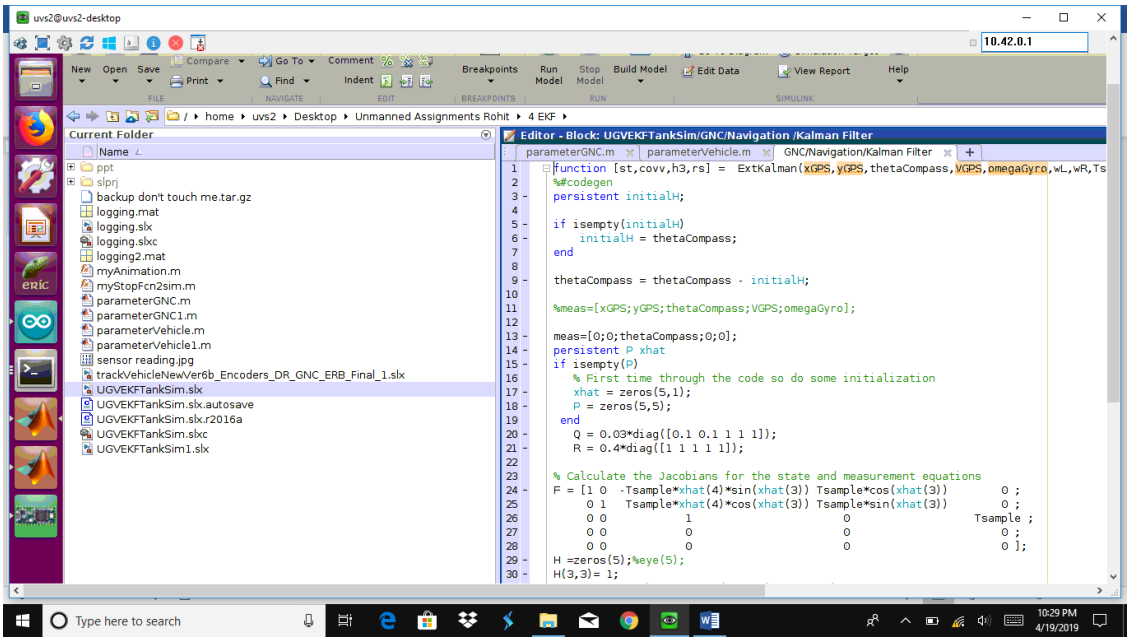

Figure 5.3: Initial Value of Compass

5.3 Q and R values

If the system is trusted more than the sensors, then multiplier value of R is increased and that of Q is decreased. Similarly, if the sensors are trusted more than the system, then multiplier value of Q is increased and that of R is decreased.

When the vehicle was steady, the compass data was recorded for 3 minutes and by using var function in the MATLAB, covariance was calculated from this data. However, the calculated covariance was then changed a bit based on the experimental observations and the behaviour of the system with an aim to minimize the drift in the orientation.

The heading mesured by the compass and the one calculated from odometer is mapped in the domain of [-π to +π]. The logic used for the conversion is demonstrated in the figure 5.4.

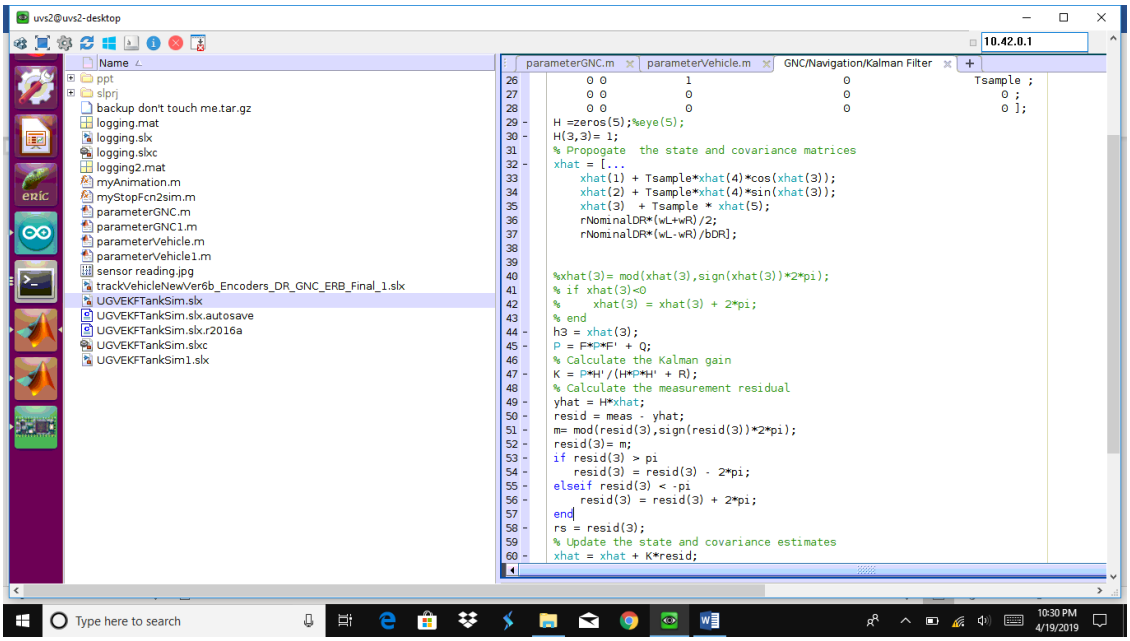

Figure 5.4: Logic for Mapping

Later, the error was calculated from the difference between the compass and odometer headings the scope of which shown in figure 5.5

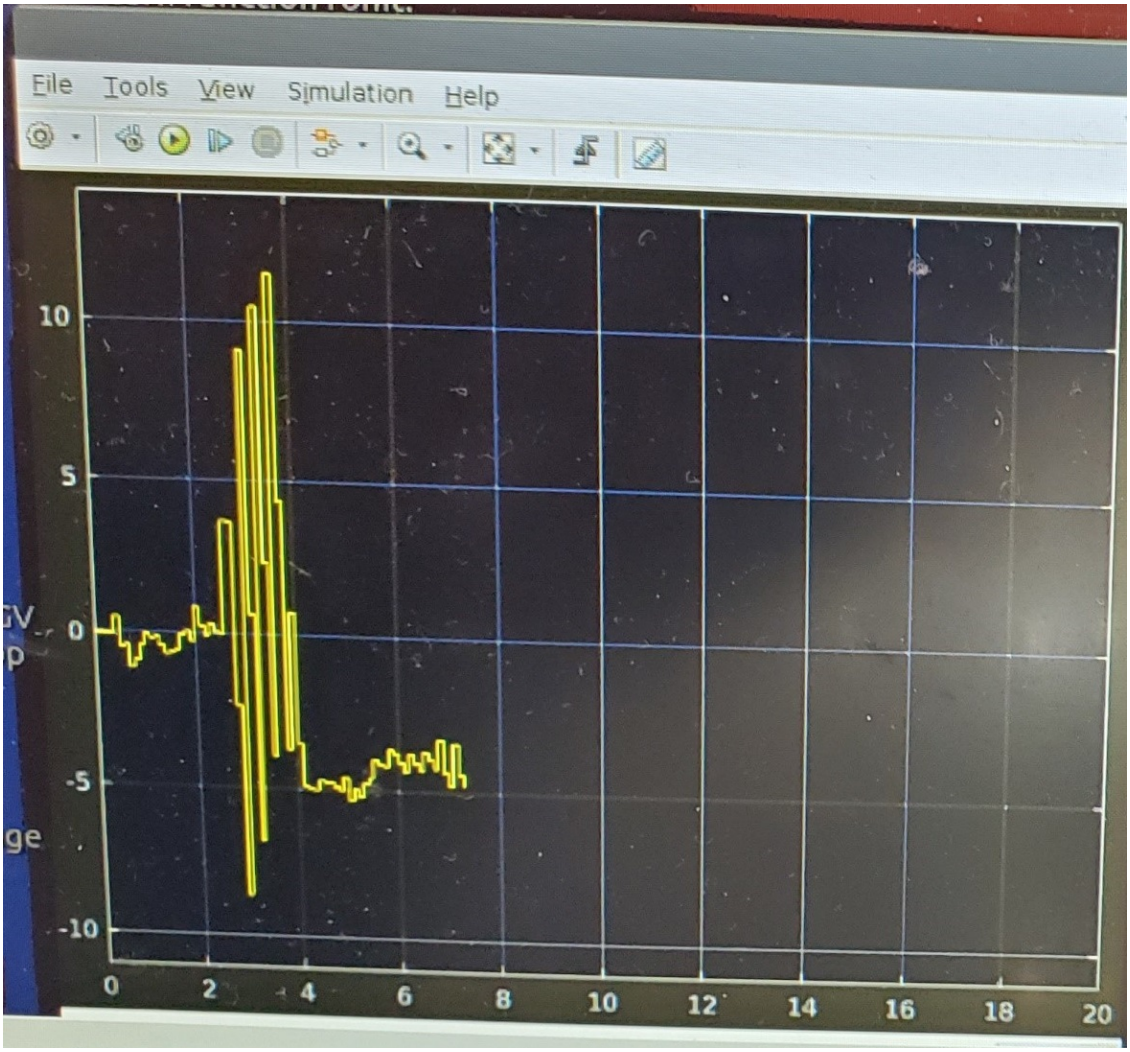

Figure 5.5: Error between odometry and compass

Chapter 6

Path Planning

Optimum path is a path which takes least time and distance to reach from point A to point B by maintaining the constraints on the vehicle. An overview of the environment and known position of start and goal pose along with the locations of obstacles are needed to know for the rover to complete the task without any obstacles. Basic path-planning algorithms, for example, A*, D*, RRT, Manhattan, Roadmap Approach, Approximate Voronoi, Cell Decomposition and so on. Every path planning method has different trajectories generated by them. As Manhattan distance approach can be used to generate the shortest paths with corners and very close from the obstacle, this make the path optimum. Hence, we will be using Manhattan approach of path planning for this task.

6.1 Manhattan Approach

Usually, it is not easy for any non-linear system to generate waypoints in such a way that no two waypoints interfere with one another. To make the path planning work, the guidance system of the rover should develop waypoints without two interfering with one another keeping the trajectory of the rover collision free from the obstacles. This planning should be done with the minimum knowledge of the location of the obstacles from the reference position of the environment.

With the known knowledge of the grid size, the guidance block generates the overall layout of the environment and position of the obstacles are provided through occupancy grid. For the rover to move towards its goal position, the path planner block from the guidance system must generate waypoints. With the known location of the starting position, ending position and obstacles on the occupancy grid, then comes the generation of the path planning through numbering. The cells around the goal position are numbers as 1 and neighboring cells are numbered with the increment of previous cell number propagating towards the start position. After generating the waypoints, the guidance system generates the smallest path from goal position to start position with avoiding obstacles in between. The red grid shows the obstacle position and the yellow marking on the occupancy grid denotes the extended part of the obstacle.

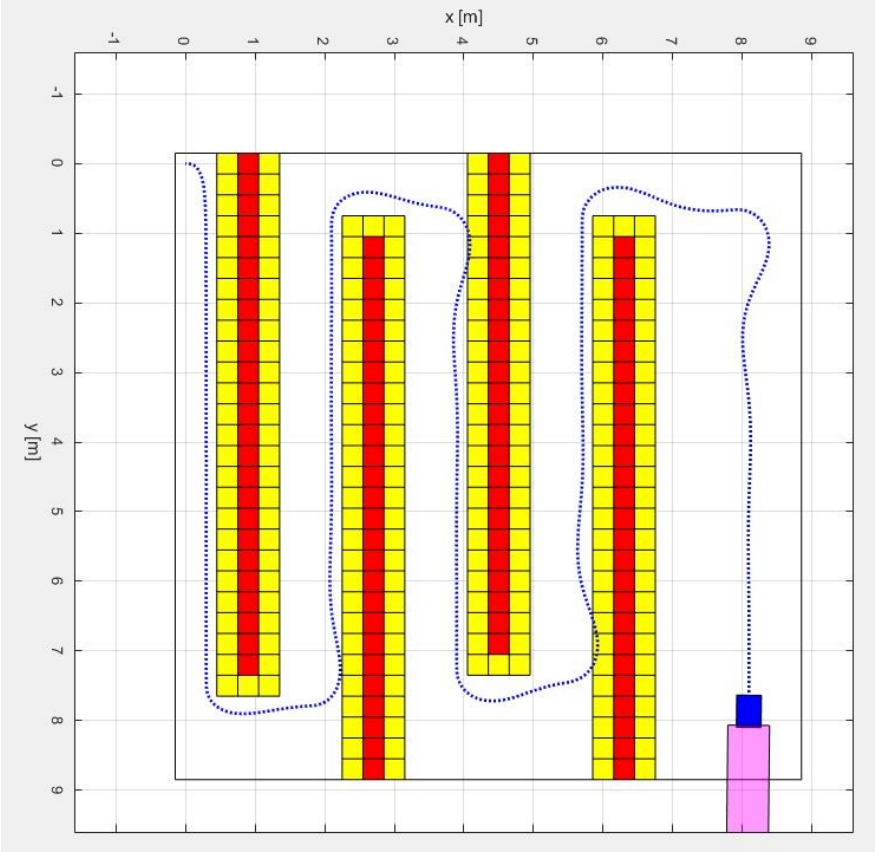

Figure 6.1: Simulation of Manhattan Path Planning

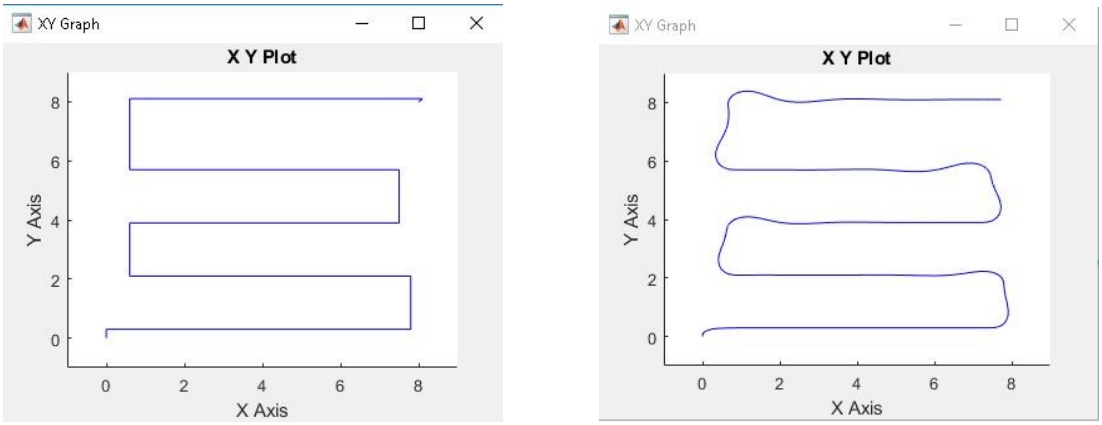

Figure 6.2: Estimate and Actual Path Planning

The estimated plot is the path generated by the Simulink model and the actual plot is the path generated by the rover.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	-1	139	138	137	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	115	116	117
2	139	138	137	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	114	115	116
3	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	112	113	114	115
4	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	111	112	113	114
5	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	110	111	112	113
6	87	86	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
7	86	85	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
8	85	84	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110
9	84	83	82	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
10	83	82	81	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
11	82	81	80	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
12	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	57	58	59	60
13	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	56	57	58	59
14	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	55	56	57	58
15	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	53	54	55	56	57
16	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	52	53	54	55	56
17	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	51	52	53	54	55
18	29	28	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
19	28	27	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
20	27	26	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
21	26	25	24	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
22	25	24	23	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
23	24	23	22	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1	1000	1000	1000	1000	1000	1000
24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	-3	1	2	3	4	5	6
25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	2	3	4	5	6	7
26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8
27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	4	5	6	7	8	9
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	5	6	7	8	9	10
29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	6	7	8	9	10	11
30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	7	8	9	10	11	12

Figure 6.3: -3:- Goal Position; -1:- Start Pose; 1000:- Obstacles; Total no. of Waypoints Generated:- 139

Figure 6.3 shows the number of waypoints generated automatically for path planning. The 1000 shows the position of the obstacles and its extension and the value -3 (red circle) shows the goal position where the rover must reach at the end of the path. The Manhattan path planning starts from the value -1. From the data, the system calculated total 139 waypoints and generates the optimum path. So, the rover starts from the 1st way point and

covers the trajectory by covering all the waypoint by avoiding obstacles if any.

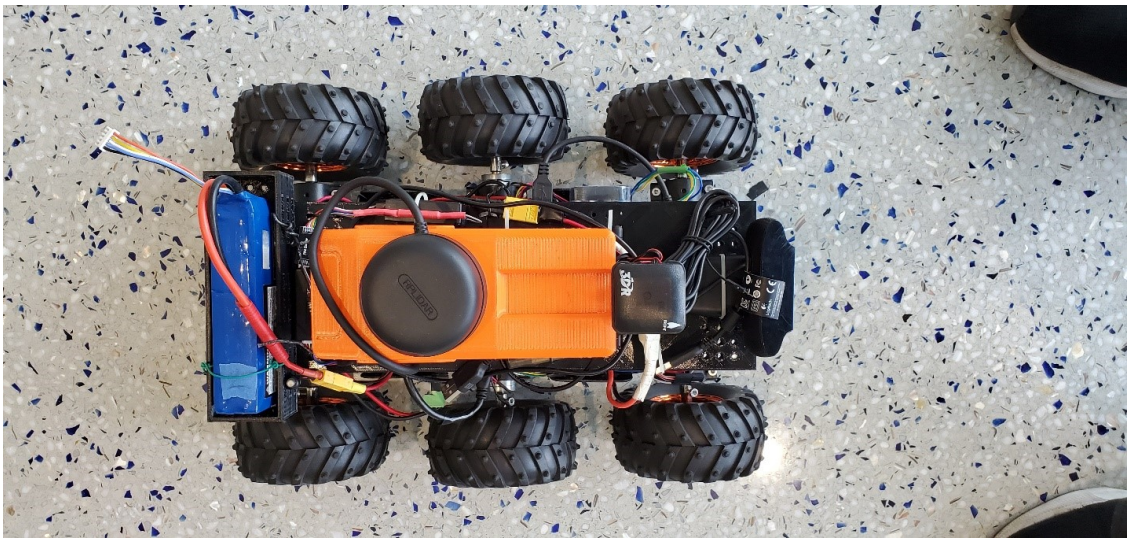

Figure 6.4: Accuracy while achieving Goal Position

Chapter 7

Path Planning with Obstacle Avoidance

The last task was to perform the path planning together with avoiding red colored obstacles using camera.

7.1 Toolbox Required

1. Camera Calibration Toolbox
2. DSP System Toolbox (Design and simulate streaming signal processing systems)
3. Image Processing Toolbox
4. Image Acquisition Toolbox
5. MATLAB Support Package

7.2 Camera Calibration

A fixed piece of chessboard is kept in-front of the rover for calibration. Using the camera calibration toolbox, the camera captures multiple images (20 for better result) from various direction and different angle. After defining the size of the chess box, the toolbox process

and sorts the image into accepted and rejected by scanning thus by defining X-axis, Y-axis and origin from accepted images.

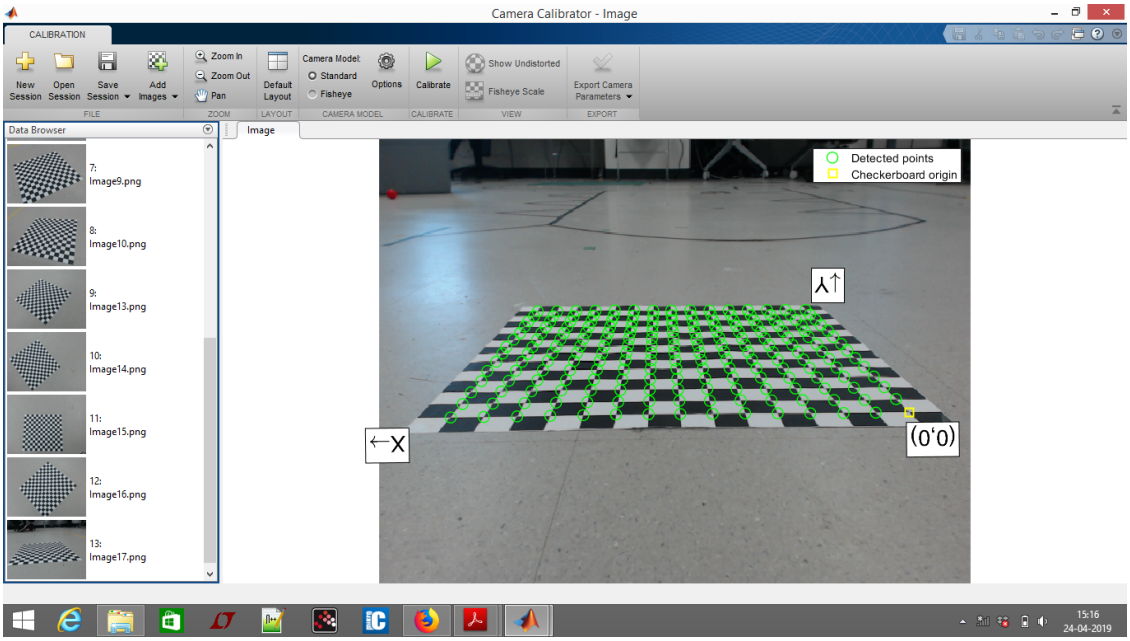

Figure 7.1: Accepted Images

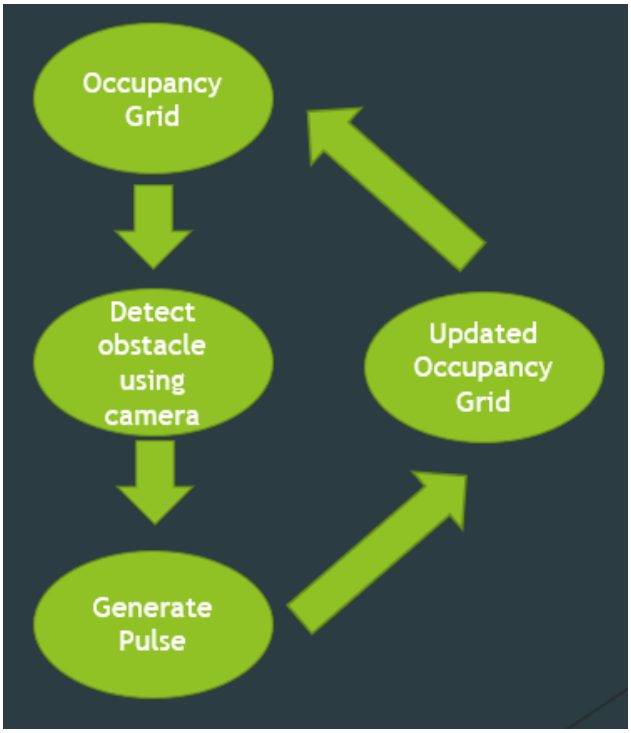

Figure 7.2: Process Flow

7.3 Hue, Saturation and Value

The HSV separates the image intensity from chroma or the color formation. This can be used to separate various color components depending on the intensity. This conversion to HSV is done by using filter function block from image processing toolbox. The red color defines the unknown obstacles and the green color states the goal position where the rover should reach. Different HSVs for different color were set to identify the obstacles and goal position.

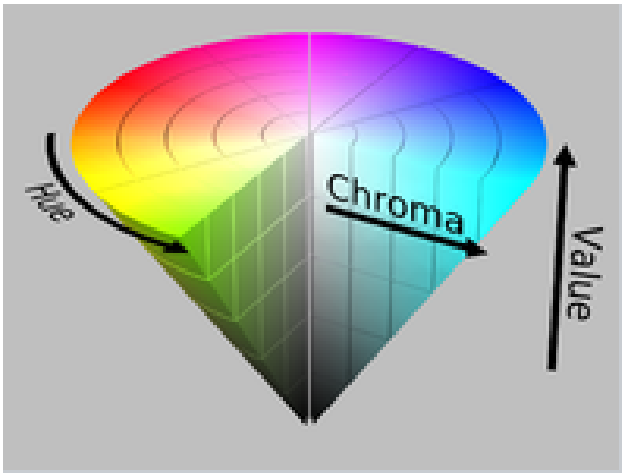

Figure 7.3: HSV value against Chroma

Unknown Obstacles (RED)

$0.52 < H < 0.6$

$S > 0.5$

$V > 0.2$

Goal Position (GREEN)

$0.01 < H < 0.1$

$S > 0.1$

$V > 0.5$

Chapter 8

Conclusions

We have successfully completed Waypoint navigation using GPS (Pixhawk), Waypoint navigation using encoders, Obstacle avoidance using Sonar and LIDAR sensor, and path panning tasks.

We were not able to complete EKF task because of the compass was not constant in any area of university where we had access. Also, Pixhawks gyro was not working properly. We were not able to do vision task because of two parallel Simulink model. When two Simulink model was running together it was causing problems in the movement of the rover.

Appendix A

Appendices

A.1 Precautions for Charging and Handling Li-Po Battery

1. For the best use of lithium polymer batteries, charger voltage should be less than or equal to the battery voltage (if not a balanced charger)

2. Do not charge them above their maximum safe voltage i.e, 4.2V per cell (Use on-cell protection circuit)

3. Do not discharge them below their minimum safe voltage i.e, 3.4V per cell (Use on-cell protection circuit)

4. Do not draw more current than the battery can provide (on-cell protection circuit)

5. Use balanced charger. (can control individual cell voltage)

6. Do not charge the batteries above or below certain temperatures (usually about 32F-122F) - sometimes handled by the charger, but often not an issue as long as the charge rate is reasonable.

7. Always store battery in Li-Polymer safe bag/bunker when not in use.

8. Do not short terminals.

9. Do not let battery go below minimum voltage when used.

A.2 Codes

All the codes will be icluded in the following github link:

https://github.com/kashishdhal/team2_uvs

Bibliography

[1] UTA, "Course."

[2] Wikipedia, "Uvg."

[3] Embention, "Uvg."

[4] Wikipedia, "Dead reckoning."

[5] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.