TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

GPS Global Positioning System

UGV Unmanned Ground Vehicle

TCP/IP Transmission Control Protocol/Internet Protocol

CPU Central Processing Unit

Wi-FI Wireless Fidelity

LIST OF SYMBOLS

$\lambda 1$	longitude of way-point 1
$\lambda 2$	longitude of way-point 2
$\theta 1$	latitude of way-point 1
$\theta 2$	latitude of way-point 2
R	Earth's mean radius (6,371km)
d	Distance between two way-points

CHAPTER 1

INTRODUCTION

1.1 Robots in various fields

In the era of industry 4.0 robots play an important role in automating the work and at the same time achieving high precision and efficiency. With the increase in demand of industrial goods and products across the world requirement for smart and intelligent robot system has never been higher than before.

Companies like Amazon and Tesla are working their way to develop smart systems which are capable of navigating across the globe. While Amazon is trying to develop this system to deliver their products to the customers, Tesla is trying to bring automation in the on road vehicle industries.

1.2 GPS Way-Point Navigation Using UGV

Smart cars and bots capable enough to move their way around to reach their destination will create nuisance and traffic, down to many folds. If the smart vehicles would be able to communicate among themselves they can calculate their motion parameters to avoid traffic and accidents. With the introduction of such smart vehicles the traffic across the globe will be smooth and the number of causalities due to road accidents will decrease many times.

To achieve such results the problem of navigating around the given GPS coordinates needs immediate attention. These smart system must be able to calculate the angle by which they have to turn to face towards their destination location and further they must be able to calculate by how much distance they are behind their destination. All these calculation are to be done in real time and with extreme precision. Even the slightest miscalculation can cause road accidents and causalities.

The problem in computing all these simple data is that the earth isn't flat and all the planar geometrical concepts fail while calculating these parameters and therefore the theorems for spherical bodies or the geometry for curved surfaces is to be used to get the parameters required for successful navigation.

This project aims to design a two-wheel differential drive robot, to perform way-point navigation for a given set of GPS coordinates. The designed robot will be able to follow the GPS guided path using the information provided by the on-board sensors like GPS and magnetometer which allows it to navigate through the GPS way-points given by the operator. It uses an on-board microcontroller (Raspberry-Pi 3B) to compute the turn angle and the distance between the current position and destination GPS coordinates, using mathematical theorems and formulas.There are many theorems which are available to compute the required parameters for the navigating between the waypoints but all these theorems have their certain shortcomings. Some algorithms or theorems are very precise but take a lot of CPU time in computation while some are fast but loses their precision. These time delays and precision loss can accumulate over a certain time period causing vehicle to behave in unintended way.

In this project all these theorems and formulas are discussed in detail and an optimized solution is put forward which is also used to drive a robot chassis to follow certain GPS waypoints.

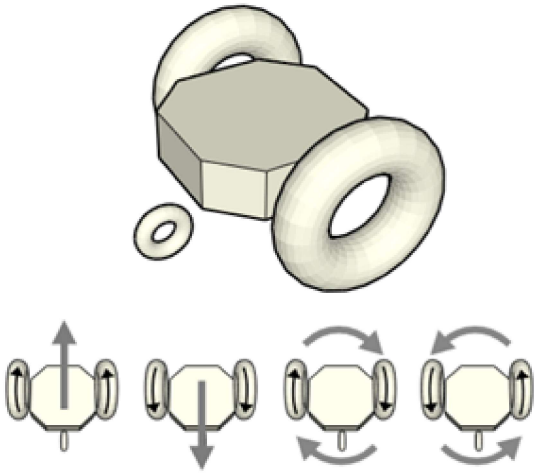

Figure 1.1: Differential Drive Locomotion

CHAPTER 2

LITERATURE SURVEY

2.1 Importance of GPS Guided mobile robots for inspection of farms

In paper [1] a small robot capable of GPS waypoint navigation is used for the surveillance of a farm. The robot is hard coded with the coordinates around the farm and therefore it continuously (in a loop) moves around the GPS waypoints and do the surveillance of the farm.

2.2 Geodetic distance calculation for automated vehicles

In paper [2] different formulas and algorithms are discussed with their mathematical complexities and assumptions to calculate the distance between two GPS coordinates (latitude and longitude).

2.3 Mobile robot following a GPS guided route using street maps

In paper [3] design of GPS guided robot is discussed which is capable of following GPS waypoints. User can plan a path using GPS coordinates and upload it to the system with the help of Bluetooth and a mobile app.

2.4 Development of an autonomous GPS guided mobile assisted robot

In paper [4], a mobile robot is designed which is capable of navigating in an urban environment. The robot followed a predetermined route by the use of a street map through GPS navigation.

2.5 GPS Based Autonomous Vehicle Navigation and Control System

In paper [5], autonomous route calculation algorithm is used to find the coordinates at each road intersection between any two input coordinates and control algorithm to navigate the prototype robot.

2.6 Automatic Semantic Waypoint Mapping Applied to Autonomous Vehicles

In paper [6], GPS and LIDAR sensor is used to detect the road surface and build a metric map and a route map processing is used to convert the GPS points into compact and suitable information for vehicle navigation.

2.7 Proposed work

Different algorithms and formulas are shortlisted based on their mathematical complexities and CPU run time to compute data for each iteration. Based on the results a final algorithm/formula is decided which is implemented on a two wheel differential drive robot which is capable of GPS waypoint navigation. Robot can be updated with GPS path over a wireless connection (IEEE 802.11) using TCP/IP protocol. Further

the robot can be continuously monitored over the range of Wi-Fi signals using a remote computer and a live surveillance feed. The robot can also be controlled manually to overcome the obstacle if experienced in path planned for the bot. The manual control is also achieved using the same remote computer and the control signals are transferred using TCP/IP protocol.

CHAPTER 3

SYSTEM MODEL, DESCRIPTION AND METHODOLOGY

3.1 Calculating distance between two GPS coordinates

Following are the three commonly used methods to calculate the distance between two GPS coordinates:

3.1.1 Equirectangular Method or Euclidean Formula:

This method is based on Pythagoras theorem and one of the major assumptions of this formula is that Earth can be considered flat for small distances. Main idea behind this formula is converting the entire globe into Cartesian rectangular grids with equal size, shape and area. The formula for the same can be given by as follows:

$$x = |(\lambda_1 - \lambda_2)| * \cos(0.5 * (\varphi_2 + \varphi_1))$$

$$y = |(\varphi_1 - \varphi_2)|$$

$$d = R * \sqrt{((x*x) + (y*y))}$$

Where R = 6731 i.e. radius of earth

3.1.2 Haversine Method:

The haversine formula provides the great circle distance (shortest distance between two points on sphere) between two geodetic points. Following is the haversine formula for calculating the distance.

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 * \cos\varphi_2 * \sin^2(\Delta\lambda/2)$$

$$c = 2 * \tan^{-1}(\sqrt{a} / \sqrt{(1-a)})$$

$$d = R * c$$

Here φ is the latitude, λ is longitude, R is the earth's mean radius (6,371km)

3.1.3 Spherical Law of Cosines:

This theorem depends on the sides and the angles of the spherical triangle and make use of the classic laws of cosines to calculate the distance between the two geodetic points. The formula for the same can be given by as follows:

$$d = \cos^{-1}(\sin(\varphi_1) * \sin(\varphi_2) + \cos(\varphi_1) * \cos(\varphi_2) * \cos(\lambda_1 - \lambda_2))$$

3.2 Calculation of bearing angle

The bearing formula gives us the angle by which we have to rotate/turn with respect to the earth's true north to face the target. To know the current heading of the chassis we are using a magnetometer which tells us the true heading or the direction faced by the chassis with respect to true north. Both the magnetometer and bearing formula outputs are compared to find the actual angle by which the chassis has to rotate from its existing heading to face toward the target.

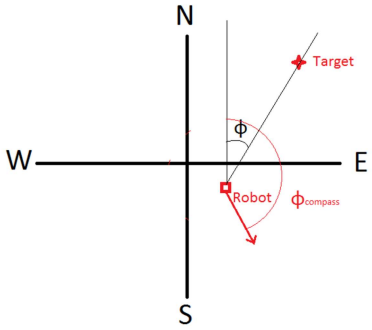

Figure 3.1: Calculation of Bearing

Following are the two commonly used methods to calculate the bearing between two GPS coordinates:

3.2.1 Haversine Method:

In our project we are using haversine bearing formula to calculate the angle by which we have to rotate to face the target point.

$x = \lambda_2 - \lambda_1$

$y = \varphi_2 - \varphi_1$

$a = \sin(y/2) * 2 + \cos(\varphi_1) * \cos(\varphi_2) * \sin(x/2) * 2$

$c = 2 * \tan^{-1}(\sqrt{a} / \sqrt{(1-a)})$

$b = R * c$

$bear = \tan^{-1}[(\cos(\varphi_1)*\sin(\varphi_2)-\sin(\varphi_1)*\cos(\varphi_2)*\cos(\lambda_2 - \lambda_2)] / [\sin(\lambda_2 -\lambda_1)*\cos(\varphi_2))]$

3.2.2 Rhumb Line method:

Rhumb line is an arc crossing all meridians of longitude at the same angle,that is a path with constant bearings measured relative to true or magnetic north.

$\varphi = \ln(\tan(pi/4 + \varphi_2/2) / \tan(pi/4 + \varphi_1/2))$

$bear = \tan^{-1}(\lambda / \varphi)$

3.3 Working

Raspberry-Pi acts as the main controller in the project which collects data from all the sensors interfaced with it i.e. magnetometer and GPS Module. The data is further processed based on an efficient algorithm. All the formulas are tested for CPU run time, delays and precision and the best one is selected. The results of all the computations made are then used to drive the motors of our two-wheel differential drive robot chassis using L298N motor driver.

Initially the user gives a set of GPS waypoints that they want the robot to follow. These waypoints (latitude and longitude) are saved in an array forming a map to be followed. The user communicates with the chassis over a Wi-Fi network from a remote computer.

After receiving all the waypoints from the user, Raspberry-Pi begins its interaction with on-board sensors (GPS and compass). GPS module (NEO-6M) communicates with Raspberry-Pi serially and provides with information such as devices'latitude, longitude, altitude with reference to the sea level, number of satellites available, current time and date etc.

Out of all these data our algorithm needs only the latitude and longitude of the chassis which extracted from the data set and the rest of the data is discarded. After retrieving the geodetic position of the chassis, Raspberry-Pi feeds this data to the part of code which calculates the distance between the two waypoints. The distance function takes two inputs i.e. current position of the chassis and the coordinates of the first/next waypoint. The python code uses the Spherical laws of cosines to compute the distance between the two waypoints. The formula takes about 11.699 us for each cycle of computation which way faster than other computation methods available.

For computing the angle of rotation, the haversine formula for bearing is used which gives the angle by which the chassis must rotate to face towards the target with respect to earth's true north. To know the actual heading of the chassis with respect to north we use magnetometer which gives the magnetic field strength in X, Y and Z axis. The magnetic strength is then converted into angle with respect to earth's north. The data from the magnetometer and the haversine bearing formula are compared and a final angle is calculated by which the chassis has to be rotated from its actual heading.

After computing both the bearing and the distance, Raspberry-Pi generates control signals for L298N motor driver to control the two-wheel differential drive.

The threshold for reaching a way point is set to 5 meters i.e. if the chassis is within 5 meters of the waypoint then the target is achieved. After reaching waypoint one, the code looks if there are more waypoints to follow, if there are more points than the same cycle is repeated otherwise the chassis stops at the last waypoint.

There is also an option for manually controlling the robot chassis over a Wi-Fi connection. At any point of time if the user realizes that the chassis is deviating from the course or there is an obstacle in the waypoint path the user can manually take over any time and control the chassis.

All the control signals for manual control are sent from a remote computer over Wi-Fi using TCP/IP protocol, also all the parameter i.e. current position of the chassis, it's current heading and its distance from the waypoint all can be observed on the remote system. Camera live feed is also available from the chassis.

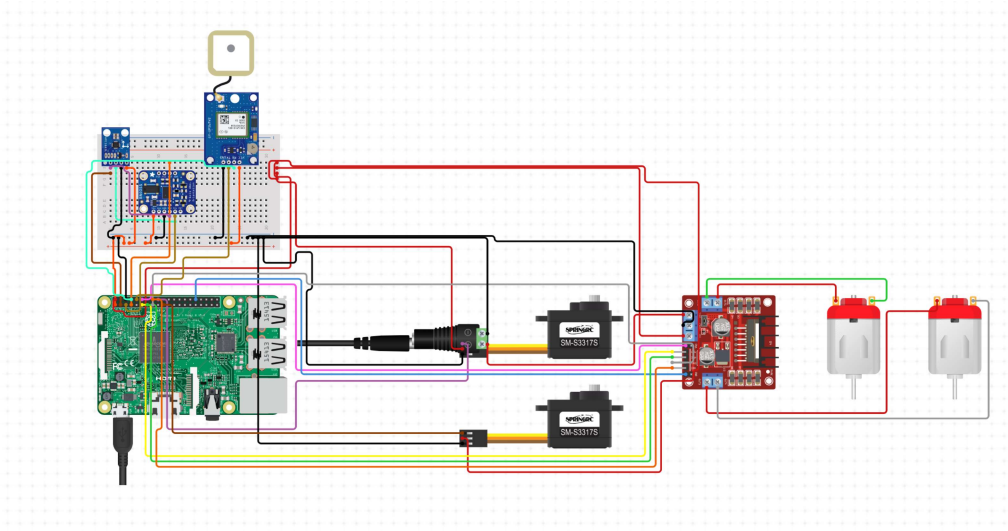

Figure 3.2: Circuit Diagram

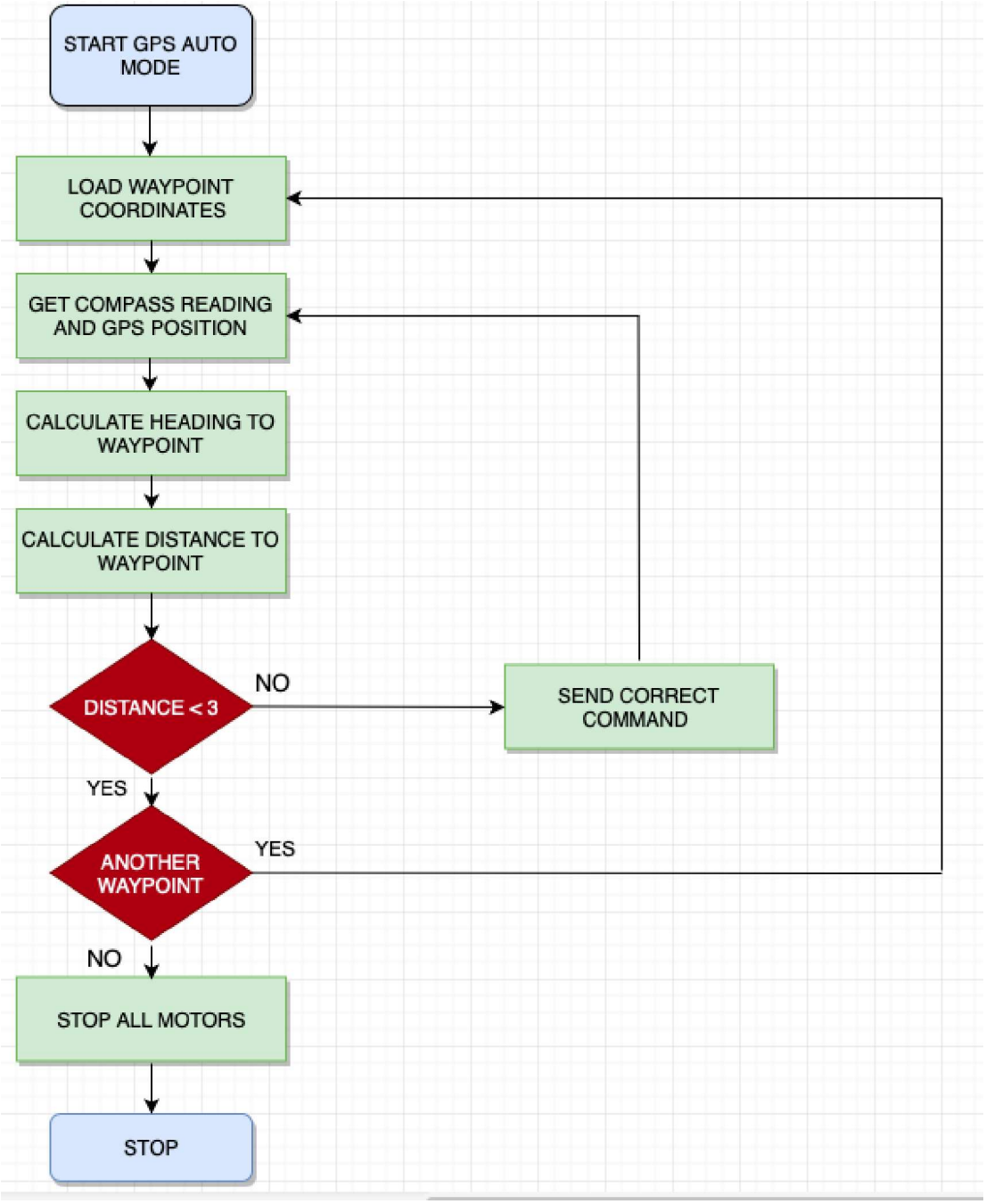

Figure 3.3: Flow chart

3.4 Experimental setup

The final setup of the Unmanned ground vehicle:

Figure 3.4: Inside view of the vehicle

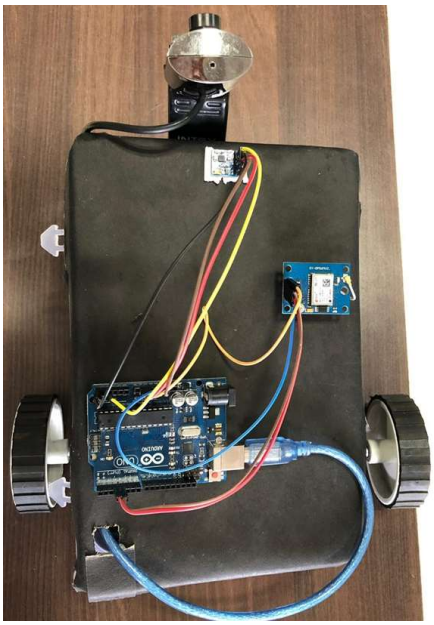

Figure 3.5: Top view of the vehicle

3.5 Hardware description

3.5.1 Raspberry-Pi 3B:

Raspberry-Pi is a low-cost computer which is used to design low cost projects and prototypes. In our project we are using it to read the sensor data and compute the bearing and distance between the waypoints so that the chassis can be directed in the right direction

Figure 3.6: Raspberry-Pi 3B

3.5.2 NEO -6M GPS Module:

NEO-6M is a GPS receiver which provides the geodetic position of a device in real time at frequency of 1 KHz. It provides the latitude and longitude of the device along with other information like altitude, time etc. In our project we are using it to determine the position of our chassis in real time to make GPS navigation successful.

Figure 3.7: NEO -6M GPS Module

3.5.3 HMC5883L Magnetometer:

Magnetometer HMC5883L measures the direction and magnitude of the earth magnetic field and hence is used for low cost compassing and magnetometry. It measures the magnetic field value along the X,Y and Z axes. It can be used as a compass to find direction or to find the direction of the heading of the device. We use this magnetometer in our project to determine the in which direction the robot chassis is facing with respect to true north.

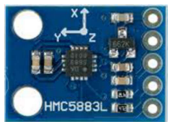

Figure 3.8: HMC5883L Magnetometer

3.5.4 L298N Motor Driver:

The L298N is an integrated monolithic circuit. It is a high current, high voltage dual full-bridge driver. Two enable inputs are provided to enable or disable the devices independently of the input signals.

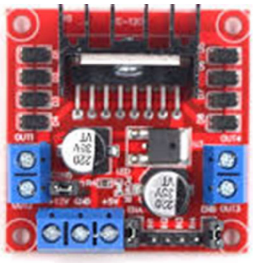

Figure 3.9: L298N Motor Driver:

3.5.5 12V DC motor 60 RPM:

60 RPM low cost, geared DC motor is used in our project to move the chassis around the waypoints. Although the motor is rated for 12V it works just fine for the voltage range of 7 to 12.

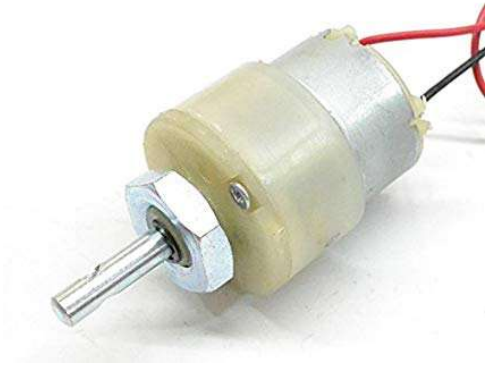

Figure 3.10: 12V DC motor 60 RPM

3.5.6 Remote computer:

A remote computer is being used to constantly monitor the movement of the chassis in real time. The chassis is equipped with camera which gives the live feed of the environment in which it is moving. The remote computer can also be used to manually control the chassis in case it is required. All the connections between the chassis and the remote computer is taking place over a Wi-Fi network using TCP/IP protocol.

3.5.7 LiPo battery:

Battery is used to power the complete system. The battery is capable of running the system for up to 30 minutes.

Figure 3.11: LiPo battery

3.6 Software description

3.6.1 Python:

Python is a high level, open source programming language which is used in our project to read sensor data as well as computing the control signals for the chassis.

3.6.2 Linux:

Linux is a family of free and open-source software operating systems based on the Linux kernel which is used in our project

CHAPTER 4

DATA ANALYSIS AND INTERPRETATION

4.1 GPS real time simulations

GPS receiver provides the geodetic position of a device in real time at frequency of 1 KHz. The GPS module communicates with Raspberry-Pi serially and provides with information such as devices'latitude, longitude, altitude with reference to the sea level, number of satellites available, current time and date etc. As we only need the latitude and longitude of the chassis, we discard the other received data.

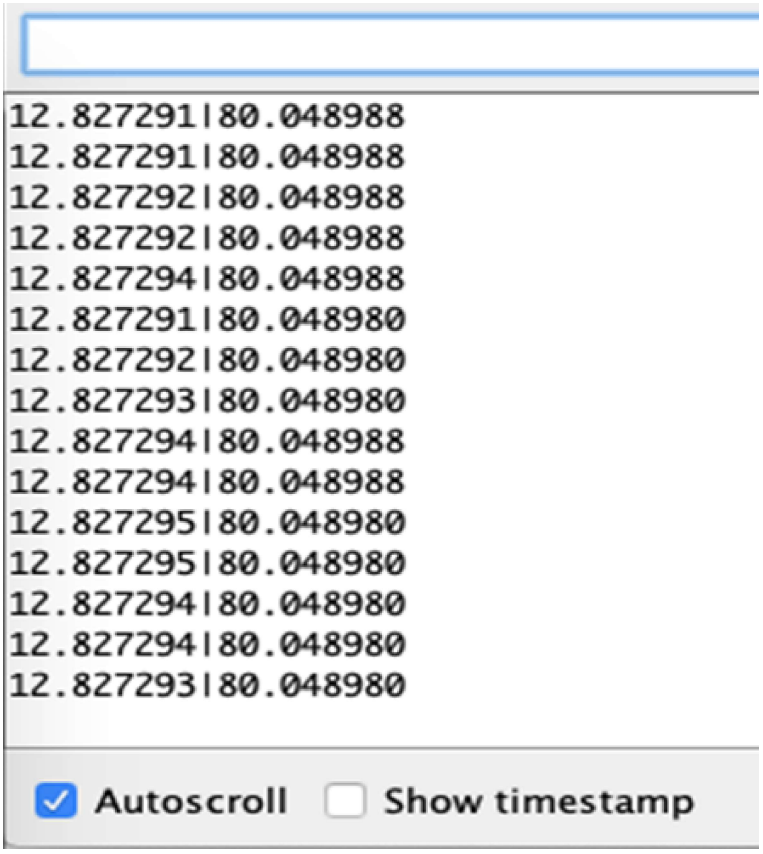

Figure 4.1: GPS Readings

4.2 Magnetometer real time simulations

The data from the magnetometer and the haversine bearing formula are compared and a final angle is calculated by which the chassis has to be rotated from its actual heading. The simulation results are:

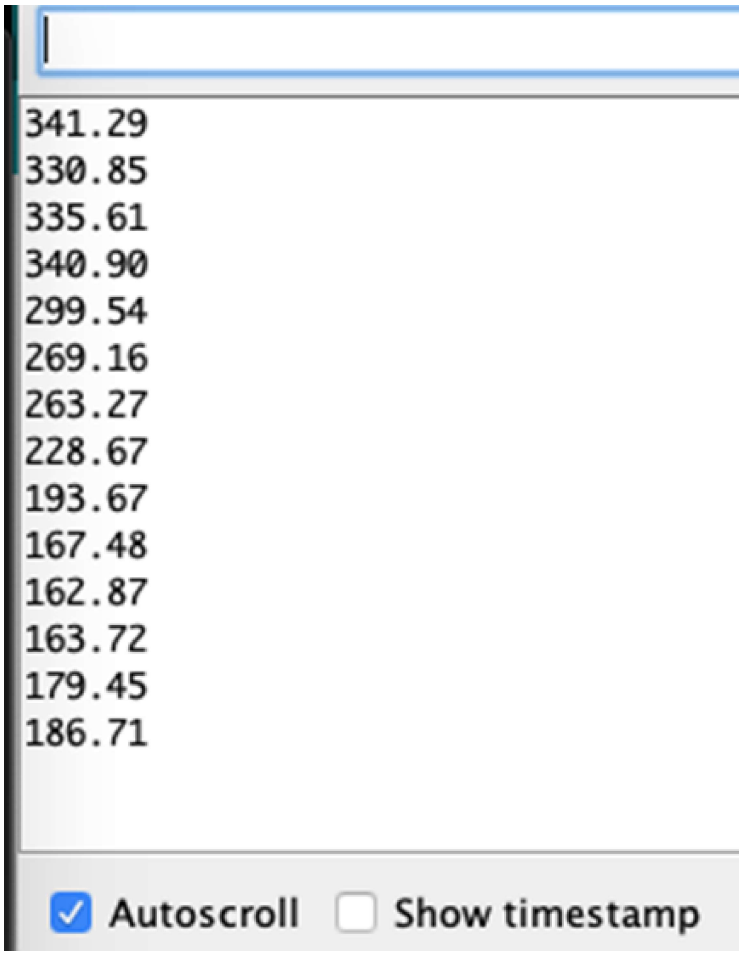

Figure 4.2: Compass Readings

4.3 GPS integrated with compass real time simulations

To move the chassis towards the destined GPS coordinates we need its own latitude and longitude as well as its bearing. Therefore, we combine the GPS and magnetometer readings together.

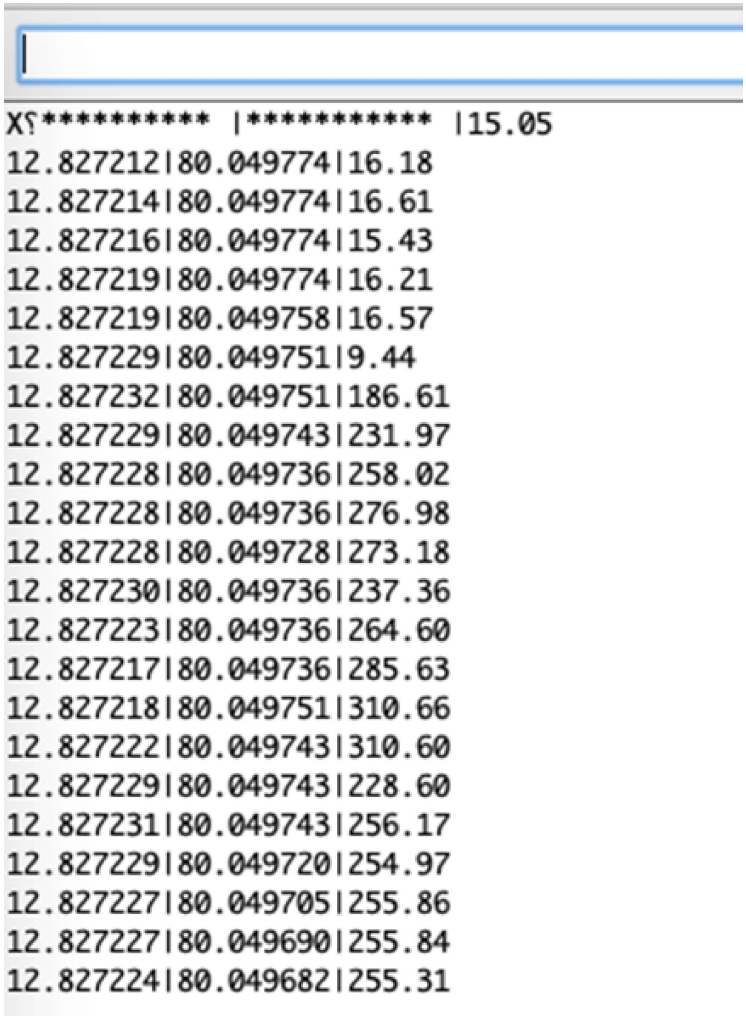

Figure 4.3: GPS Readings

4.4 Comparison of the discussed distance formula based on the CPU run time

To compare the above discussed formulas a python code was written which was iterated several times for the same set of values (latitudes and longitudes). All the three formulas were compared for same set of values and the CPU run time was noted.

Following are the results for the run time test.

```
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Euglidean distance is
126.969413938
0.0250000953674
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Euglidean distance is
126.969413938
0.00999999046326
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Euglidean distance is
126.969413938
0.0250000953674
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Euglidean distance is
126.969413938
0.0260000228882
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Euglidean distance is
126.969413938
0.0239999294281
>>>
```

Figure 4.4: Euclidean distance

```
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Haversine distance is
126.969413937
0.0239999294281
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Haversine distance is
126.969413937
0.0380001068115
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Haversine distance is
126.969413937
0.0209999084473
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Haversine distance is
126.969413937
0.0190000534058
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
enter longitude of destination77.675971
Enter latitude of origin12.853138
enter longitude of origin77.675492
Haversine distance is
126.969413937
0.0369999408722
>>>
```

Figure 4.5: Haversine distance

```
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
Enter longitude of destination77.675971
Enter latitude of origin12.853138
Enter longitude of origin77.675492
Distance using spherical laws of cosines
126.969350879
0.0340001583099
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
Enter longitude of destination77.675971
Enter latitude of origin12.853138
Enter longitude of origin77.675492
Distance using spherical laws of cosines
126.969350879
0.0169999599457
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
Enter longitude of destination77.675971
Enter latitude of origin12.853138
Enter longitude of origin77.675492
Distance using spherical laws of cosines
126.969350879
0.0139999389648
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
Enter longitude of destination77.675971
Enter latitude of origin12.853138
Enter longitude of origin77.675492
Distance using spherical laws of cosines
126.969350879
0.0200002193451
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\euglidean.py
Enter latitude of destination12.852096
Enter longitude of destination77.675971
Enter latitude of origin12.853138
Enter longitude of origin77.675492
Distance using spherical laws of cosines
126.969350879
0.0160000324249
>>>
```

Figure 4.6: Spherical Law of cosines distance

Based on the output it was noted that Spherical Laws of cosines took less time (0.017 seconds per 10,000 iterations) to calculate the distance as compared to other formulas. Based on the results it was decided that Spherical Laws of cosines will be used for further testing and implementation purpose.While haversine formula is much more precise but for small distances the output of both the formulas do not vary much and therefore speed of calculation is preferred over the accuracy in our project as we will be conducting the test for waypoints which are within 100 meters of each other.

4.5 Comparison of the discussed bearing formula based on the CPU run time

To compare the above discussed formulas a python code was written which was iterated several times for the same set of values (latitudes and longitudes). All the three formulas were compared for same set of values and the CPU run time was noted.

Following are the results for the run time test.

Figure 4.7: Haversine bearing

```
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\bearing-2 (1).py
Enter latitude of destination12.852127
Enter longitude of destination77.676000
Enter latitude of origin12.852757
Enter longitude of origin77.675602
bearing using Rhumb lines formula
-148.365681304
0.0160000324249
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\bearing-2 (1).py
Enter latitude of destination12.852127
Enter longitude of destination77.676000
Enter latitude of origin12.852757
Enter longitude of origin77.675602
bearing using Rhumb lines formula
-148.365681304
0.0239999294281
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\bearing-2 (1).py
Enter latitude of destination12.852127
Enter longitude of destination77.676000
Enter latitude of origin12.852757
Enter longitude of origin77.675602
bearing using Rhumb lines formula
-148.365681304
0.00999999046326
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\bearing-2 (1).py
Enter latitude of destination12.852127
Enter longitude of destination77.676000
Enter latitude of origin12.852757
Enter longitude of origin77.675602
bearing using Rhumb lines formula
-148.365681304
0.0220000743866
>>>
 RESTART: C:\Users\notan\OneDrive\Desktop\majorProject&Robotics\python\bearing-2 (1).py
Enter latitude of destination12.852127
Enter longitude of destination77.676000
Enter latitude of origin12.852757
Enter longitude of origin77.675602
bearing using Rhumb lines formula
-148.365681304
0.0299999713898
>>>
```

Figure 4.8: Rhumb line bearing

Based on the output it was noted that Haversine bearing formula took less time to calculate the bearing as compared to other formula. Hence, it was decided that Haversine bearing formula will be used for further testing and implementation purpose.

CHAPTER 5

FUTURE ENHANCEMENT

- For better results and precision diversity GPS receivers can be used to rule out GPS data error.
- Use of Extended Kalman Filter to remove or neutralize the burst error from GPS.
- Use of control systems to design controller for the chassis to achieve faster turning towards the targets.
- Introduction of obstacle avoidance to the basic code so that the chassis can avoid obstacle and re-plan the path to complete or achieve the waypoints.
- Design of a system GUI so that the user can interact with the bot with very much ease and track on a live map to increase user experience.
- Introduction of SLAM (Simultaneous Localization and Mapping) so that the bot not only do the surveillance in the unknown territory but also make a rough map of the area.

CHAPTER 6

CONCLUSION

This project presented the study and design of autonomous navigation of mobile robots, which is a step towards self-driving cars and navigation systems. Generally, the algorithms associated with the navigation system uses data from additional sensors combined with complicated filtering methods and control system. The requirement of additional and better sensors and controller will increase the cost of the system many times.

In this project we tried to implement autonomous navigation system without the use of additional sensors and controllers thus keeping the project cost in control and at the same time achieving good results.

REFERENCES

[1] Tang, Liqiong, and Phillip Abplanalp. "GPS guided farm mapping and waypoint tracking mobile robotic system." In *2014 9th IEEE Conference on Industrial Electronics and Applications,* pp. 1676-1681. IEEE, 2014.

[2] Esenbuga, Ozge Gizem, Alper Akoguz, Emre Colak, Beril Varol, and Bihter Erol."Comparison of principle geodetic distance calculation methods for automated province assignment in turkey." *In International Multidisciplinary Scientific Geo-Conference: SGEM: Surveying Geology mining Ecology Management2 (2016)*: 141-148.

[3] Nguyen, Tam, Josh Slonaker, and Mohammed Kadous. "Semi autonomous wireless controlled robot".*In Capstone Senior Design Project (2012).*

[4] Moreland, Scott, Ashley Wellman, and Andrew Goldenberg. "Development of an autonomous GPS guided mobile assisted robot."*In University of Toronto, Researchgate publication, March 22, 2007.*

[5] Ghazi, Irtsam, Ihtisham ul Haq, Muhammad Rashid Maqbool, and Sanaan Saud."GPS Based Autonomous Vehicle Navigation and Control System."*In 13th International Bhurban Conference on Applied Sciences and Technology (2016).*

[6] Zoccoler, Matheus, Patrick Y. Shinzato, Alberto Y. Hata, and Denis F. Wolf."Automatic Semantic Waypoint Mapping Applied to Autonomous Vehicles."*In Robotics, Springer, Berlin, Heidelberg, pp. 96-113, 2014.*

[7] Micheal E. Holden. "Low-Cost Autonomous Vehicles Using Just GPS".*In American Society for Engineering Education. San Francisco State University, 2004.*

[8] Honeywell International."Honeywell's HMC6352 2-Axis Digital Integrated Compass Solution". *In Honeywell International, 2003.*

[9] Jain, Kushwant and Venu Suluchana. " Design and Development of Smart Robot Car for Border Security". *In 2013 International Journal of Computer Applications 76, no. 7, 2013.*

[10] Da Xu, Li, Wu He, and Shancang Li. "Internet of Things in industries: A survey".In Industrial Informatics,A IEEE Transactions (2014).