GPS Guided Autonomous Rover

ELEN3017

Josh Berman 320856

Group: **16**

Partners: **Terry Bugai, Kent Salmon, Mayur Chiba**

Date: **24 May 2013**

School of Electrical & Information Engineering,
University of the Witwatersrand,
Private Bag 3,
2050,
Johannesburg,
South Africa

Abstract

This document provides the development and implementation of a GPS (Global Positioning System) guided rover. The rover is to compete in a race on a pre-defined track consisting of vertices made up of GPS coordinates. The design is comprised of a perspex base, battery pack, servo motors, Ublox GPS module and an ATMega328 Arduino microcontroller. The microcontroller receives the input from the GPS module in string format through serial communication. This data is then sent through our path finding algorithm and the direction is calculated. The rover runs through the course with minor mishaps. Improvement recommendations include alternate algorithms and additional sensors.

1. INTRODUCTION

Critical to this project, a Ublox GPS unit is mounted on the highest point on the rover's surface. Location data is converted to a usable form and a path is then calculated using our path finding Algorithm. The rover determines its location and direction in order to estimate how many degrees to rotate. It then rotates accordingly, drives forward and re-calibrates. This process continues until the rover has reached its destination.

Section 2 defines the constraints versus requirements of this autonomous rover. The physical circuit implementation is discussed in section 3 and the construction and implementation in section 4. The stability of the physical structure of the rover is discussed in section 5, the algorithm is discussed in section 6 with all results in section 7. The concluding statements with recommendations appear in section 8.

2. CONSTRAINTS AND REQUIREMENTS

2.1. General Assumption and Constraints

- The continuous servo motors given must be used to drive and steer the rover.
- The battery pack must be the sole power supply for the rover. The rover must hence be able to provide enough torque to carry the batter pack.
- The Ublox GPS module must be used to control the direction of the rover.
- The Perspex provided must be used in the construction of the rover
- No power tools are to be used throughout the construction of the rover.

2.2. Success Criteria

- Navigate through a track consisting of vertices made of GPS coordinates in the quickest possible time without veering off course.

2.3. Component Constraints

For the purposes of this autonomous rover, a standardised issuing of components to be used as power supplies and tracking modules forms the basic constraints of the subsequent system. It is around these components in table 1, that the subsequent design must meet all the requirements.

Table 1: Issued components and relevant ratings

Components	Relevant Ratings
2×Continuous Rotation Servo Motors	6V
1×Battery pack	6V 2.8Ah
$500mm$ ×300mm Perspex sheet	N/A
2×Rubber/plastic wheels (not issued)	9cm diameter
1×Track ball (not issued)	N/A

3. DESIGN APPROACH

3.1. General Design Approach

When comparing the constraints and requirements of the design the following two options for the drive mechanism of the rover are presented. One way is to use a single continuous rotation servo to produce the driving force acting on the rear axle of the rover and to use a 0-180° semi rotational servo to direct the front wheel which controls the direction. Another way is to use two continuous rotation servos to drive each wheel independently. This would allow us to turn on our axis by rotating the wheels in opposite directions. The latter option became the chosen one as the rover needs to make immediate turns when advancing passed coordinates.

The choice of the microprocessor used is dependent on a number of factors. The microprocessor must contain enough memory to house our entire code. The processing speed must be enough to calculate in real time before the rover crosses the next point. The micro must contain enough input/output pins so that the GPS and servo motors can be connected.

The Arduino ATMega328 fits all this criteria and is readily available at most electronic suppliers. The 'sketch' is coded in 'c' providing an easier platform than PIC's requiring 'assembly' or 'mipps'.

ASSEMBLY AND PHYSICAL FEATURES

3.2. Power supply

The Battery pack provided has an output voltage measured to be 6.3V when fully charged. This is the sole supply for the rover and must provide enough current to power the servo motors, GPS unit and Arduino microprocessor. The optimum voltage for maximum torque of the servos is 7.2V [1], but as little as a 4V supply can be used. The GPS module requires 3.3V [2] and is regulated through the Arduino. The microprocessor requires a 5V supply but anything up to 8V can be used as the PCB board contains its own 5Vdc regulator [3]. These values are measured experimentally using a controlled variable power supply and displayed in table 2 as minimum values for acceptable levels of performance.

Table 2: Tested load characteristics of rover under normal conditions.

Component	Voltage(V)	Load Current(A)	Load Power (W)
Arduino	5.0	80m	0.4
GPS	3.3	40m	0.13
Servos	6.0	190m	1.14
Rover in motion (under no load)	6.0	360m	2.16
Rover in motion (under load)	6.0	420m	2.52

Load currents are tested using a DC power supply and reading off the current drawn. These test results show that the rover requires at least a 6V supply at <1Amp.

3.3. Servo Motors

Two FS5109R standard continuous rotation servos are used to drive and steer the rover. A PWM signal generated by the microprocessor controls the angle of rotation. Since our servos are continuous rotation, the angle is used to determine the wheel direction once off, i.e. forwards or backwards. The servos continue to rotate until a new command is given as they do not contain a limiting potentiometer. In order to turn the rover, the servos must be set to rotate the wheels in opposite directions. The duty cycle of the PWM determines the direction of the rotation of the servos and the frequency determines the speed of rotation. Our frequency is set in order to get maximum speed of rotation while still providing enough torque to carry the battery pay load. This frequency is set to be 333.3 Hz rated at the maximum our plastic gears can handle. The normal operational frequency is between 50-60 Hz [4]. The rovers speed is noticeably faster.

Table 3: Servo Motor Ratings from datasheet

Components	Ratings	Ratings
Power	4.8V	6V
Speed	0.16sec/60°	0.16sec/60°
Torque	6.5kg/cm	9.5kg/cm
Weight	52g	N/A
Size	40.8*20.1*38.0mm	N/A
Application	robot	N/A

3.4. The Wheels

The diameter of the wheels is a crucial factor to the speed of the rover. As the servos are now rotating at a fixed speed, Assuming enough torque is provided through entire operation, we can assume the larger the wheels the greater the speed. This is limited to availability of attachable wheels and as mentioned the servo torque.

Different sets of wheels are tried and tested until the load on the servos is minimised and the speed of the rover is maximised. A final wheel of diameter 9cm is chosen. The wheels are taken from an old Mecano/Lego set and are fixed in place using rods. Glue is used to secure these rods. A table of varying size wheels and their respective speed is shown below.

Table 4: Wheel diameter and resultant speed/time

diameter	Perimeter	Rotations per meter	Time to cover a meter (sec/meter)
4cm	12.5cm	8	7.68
6cm	18.85cm	5.3	5.08
9cm	28.27cm	3.53	3.39

Speed is the Inverse of the time taken to cover a meter.

3.5. The Chassis

A 6mm thick sheet of Perspex measure to be 300x500mm is issued and must contribute to the structure of the rover. The Perspex is light in comparison to a metal structure but heavy when considering the added weight of the battery pack. Perspex is used as it can be easily cut using a saw or bored using a hand drill. Cut out pieces can be attached using a glue gun. Aesthetic additions such as a wind deflector are added for our own amusement. The Chassis is a single sheet of Perspex with side sections removed for the wheels to fit. Holes are drilled for the Arduino to be secured and the front track ball to be mounted. The base plate dimensions can be found in Appendix A, Figure 1.

3.6. GPS Module

The Global Positioning System comprises of multiple Satellite networks orbiting the earth. These satellites transmit data or "timestamps" depending on their position in space. The data is received by a GPS receiver where there is an unobstructed line of sight. This is accessible to anyone from military personnel to civilians. The most common GPS receivers are located in land mapping devices used in cars and cell phones [5].

The data set or information from GPS receiver is called NMEA, which stands for National Marine Electronics Association. These data strings based on ASCII are communicated at a rate of 38400 bits per second [6] which is equivalent to the baud rate of 38400 characters per-second. The data is transmitted in sentence codes and is decoded into string format which we then manipulate to find our required information. The GPS data is normally received and transmitted in a standard NMEA-0183 format. This GPS continuously outputs NMEA sentence codes such as RMC and VTG. In this case, the only concerned is the ($GPRMC) sentence, which represents the Global Positioning Recommended Minimum Specific GPS/Transit Data.

A serial communication port is set up in the software of the Arduino in order to receive the desired data. Once this data is received we can use string manipulation to extract the latitude and longitude. Checks need to be performed in order to make sure data is being received else the information is insufficient.

The GPS module takes roughly 30 seconds to begin receiving information and therefore initial loops need to be put in place to prevent the rover from moving forward before this information is received. The Latitude and longitude are displayed with five decimal places in order to depict an accuracy of 1.1 meters [6]. The accuracy during final tests was less than this and caused minor path variations for this rover. With a larger antenna and the storage of latitude and longitude in integer format as opposed to floats, we can reduce this error significantly. There was not enough time to incorporate this fix in the code and hence the error was relatively high. The GPS output can be found in Appendix B.

4. CONSTRUCTION

The rover is pieced together using assorted screws and a glue gun. The Servos are attached in opposite directions to the underside of the Perspex base plate. The wheels are then attached independently to each servo using the screws and the servo arms that are supplied with the servos. The track ball is fixed using the holes in the front of the chassis that were drilled previously. The battery pack is strategically placed and secured in the center of the chassis for optimum weight distribution. The GPS unit is placed on top of the battery pack and the TX and RX pins are connected to pin 3 and 5 of the Arduino respectively. The Vcc and GND are connected directly to the 3.3V output of the Arduino and the GND pin of the Arduino respectively. The Arduino is secured in front of the battery pack and the supply voltage is connected directly, via a power diode, into the power connector. The servo signal inputs are connected to PWM pins 6 and 10 of the Arduino and their respective ground and supply are connected to the battery supply rail. The rear spoiler is added for aesthetic reasons, and a rear leg is then secured. The construction of the rover is complete and the Arduino sketch containing the path finding algorithm can be uploaded.

5. DESIGN INSTABILITY

The performance of the rover is heavily susceptible to wind resistance and uneven surfaces. The small trackball in the front can get lodged in a crack in the surface of the track thus altering the direction of movement. More calculations need to be done in order to get the rover back on track as it is no longer moving along its desired path.

The Battery pack is large and heavy and puts a great deal of strain on the servos. The battery is located slightly forward on the chassis. Ideally one would like to have a 50:50 weight distribution but as a result of the vibration caused by the uneven surface, the weight is constantly shifted and the rover can topple backwards. A support bar

protruding from the rear of the rover prevents the rover from toppling backwards if the weight is shifted drastically.

The wheels need to be perfectly aligned as a slight degree of inaccuracy can cause a large change in direction when travelling over large distances. This takes copious trials and error. Additionally the servos can be tweaked by altering their variable resistors; this can slow down a servo independently allowing the rover to tend towards a certain direction. This is used for correction of alignment and compensation of independent speed inaccuracy.

6. PATH FINDING ALGORITHM

The path finding algorithm is initially drafted in *sudo code* to get a better understanding of the objective required. The code is then simulated in Matlab which provides a visual representation of the calculated angles and their respective paths. These simulated results can be seen in Appendix C. Once the algorithm is tried and tested on Matlab, it is then coded in 'c' and used on the Arduino to output the required results.

The algorithm begins with finding the direction required for the rover to move. This can only be done with two sets of reference coordinates. Thus the rover must move forward initially to retrieve these two sets of coordinates. The coordinates are passed to a function that calculates the angle of movement and the rover is rotated accordingly. The rover must continuously loop through this set of instructions until the desired point is reached. The rover then checks for the next vertices and re-evaluates its position.

7. RESULTS

7.1. Test day

Assumptions are made for the testing of the rover: the course is assumed to be smooth, the GPS module would constantly provide accurate information and that the power supply would be constant. This is an ideal case. During testing however these factors come into play and cause variations in the results. The rover completed the course but overshot one or two vertices on occasion. The speed and direction was hindered by the uneven surface which caused the rover to veer off the path towards the end. After much initial tests prior to the demonstration, the battery output voltage was lower than its initial value and the rover had to be connected directly to a PC in order to provide enough current for all the components. The rover did manage to complete the entire path at a relatively fast pace with all the constraints adhered to.

8. IMPROVEMENTS

The rover's improvements can be made in the form of additional sensors such as sonar or a gyroscope. The sonar sensor can measure the distance to an object in its path and hence trigger an object avoidance algorithm. An example of this algorithm is shown in Appendix D. The gyroscope can be used to measure the angle of rotation. This would be preferable as the current design requires the rover to move forward and retrieve new coordinates and hence calculate its angle.

The rover build quality was of high standard but the uneven surface can cause undesired results. The rover can be built with a wider base and thicker tires to absorb bumps in the track.

An antenna can be added to improve the GPS signal and therefore reduce the time taken to retrieve coordinates; this would noticeably speed up the rover's algorithm. To add theses improvements the connection diagram in Appendix E can be used for reference.

9. CONCLUSION

A GPS guided autonomous rover was designed, constructed and tested. The system followed a strict set of constraints with regards to the structure of the Rover as well as the implementation. The employed system autonomously navigates its way through a path set out in GPS coordinates powered by a single 6.3V battery pack. The accuracy of the GPS module was questioned and the desired outcome was hindered. The algorithm that was initially drafted in Matlab did not function as well as expected when converted to 'c', but the overall construction and implementation produced the required result. This project is a prime example of an intelligent autonomous vehicle.

.

REFERENCES

[1] Servo Voltage, FS5109R Data-sheet.
[2] GPS Operating Voltage, http://www.openimpulse.com/blog/products-page/product-category/ublox-neo-6m-gps-module/ ,June 2013
[3] Arduino Operating Voltage , http://arduino.cc/en/Main/arduinoBoardUno, June 2013
[4] Servo Frequency, http://pcbheaven.com/wikipages/How_RC_Servos_Works/ , June 2013.
[5] GPS information, http://en.wikipedia.org/wiki/Global_Positioning_System , May 2013.
[6] Implementation of GPS for Location Tracking , 2011 IEEE Control and System Graduate Research Colloquium, Shah Alam, Malaysia

Appendix A

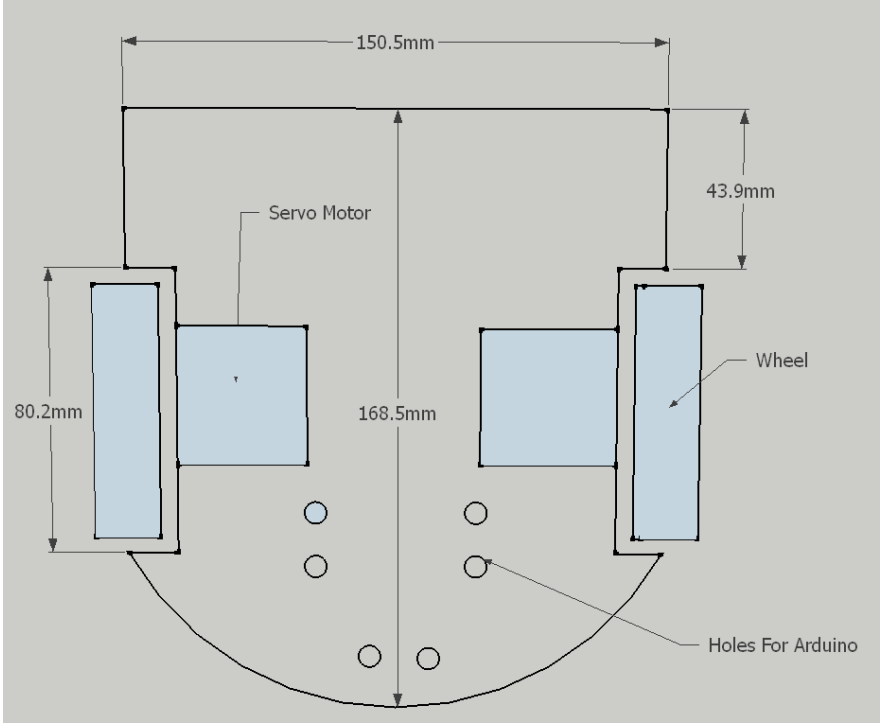
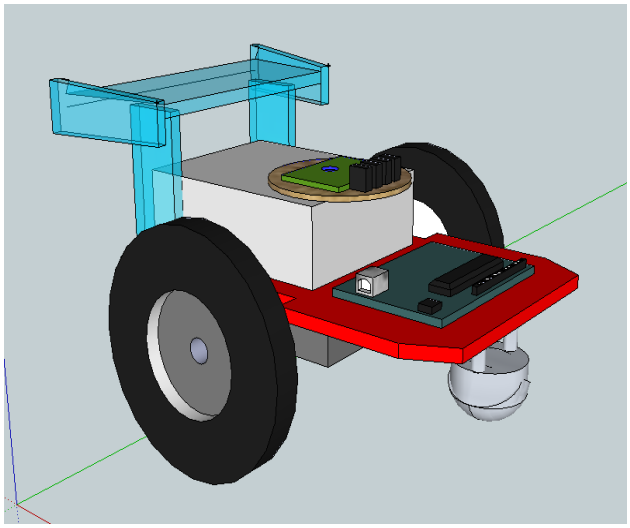
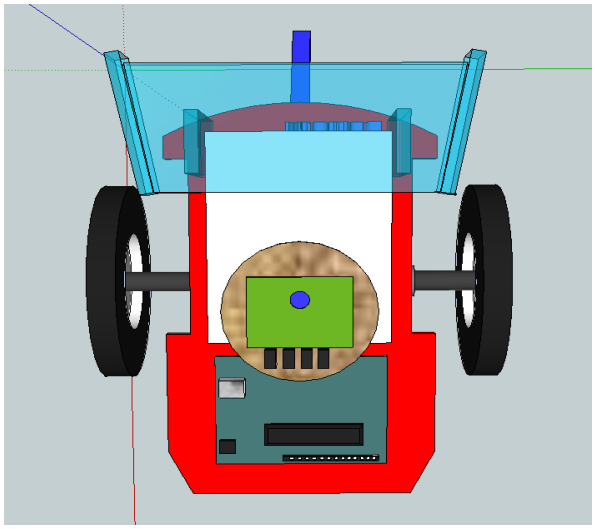


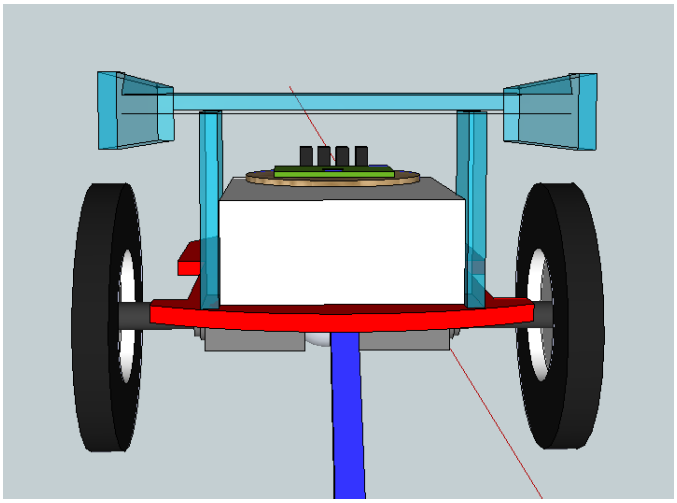
Figure 1: Perspex Chassis Dimensions



Figures 2: Side View of Rover



Figures 3: Top View of Rover



Figures 4: Rear View of Rover

Appendix B

```
5*2$GL6.79,8179,110A*1$Z,1000,,1,,*$GPRMC,115002.00,A,2611.83848,S,02801.67432,E,1.367,56.77,080513,,,A*4F
,M,1.367N23KAA$GA100,184S0063,13.675M3,,4
*GG,1115077441,446,242,F$GV,10,,6218199542,821,D$GV,1,,,0,,,2,,,14GL,184,0064,1020,7GD,0208023006
0,A,2611.83881,S,02801.67418,E,0.596,61.23,080513,,,A*4F
,M,0.596,10,*5$G,1002188,0064,102675M3,,C
*$G,11150,6446,441,242,E$PV,10,,6218198542,820,1
1*GG,188,00.1,10.0,7GD15308023006
0,A,2611.83869,S,02801.67373,E,0.246,62.78,080513,,,A*43
,M,0.246,,4,A1
,0,54.,30,4GSA2750,,,,726101$PV,11,,77051,051904227
02020027
5*0$G,6.89,8.33,540A*$Z,1000,,10,$GPRMC,115005.00,A,2611.83867,S,02801.67333,E,0.283,59.01,080513,,,A*47
,M,0.283,,5,A0
,0254.M20,*GS,2750,,,2726101$GV,1,,,73051,051,04227
02020207
,6.8,,8.73,500A*B$ZA10000,100*$GPRMC,115006.00,A,2611.83837,S,02801.67283,E,0.444,54.92,080513,,,A*41
,M,0.444N02K*B$GA100,183,0062,10267.M3,,4
*1$G,1015076441,441,242,F
02,502,E
,188,08.2,,060,7GD15600023006$GPRMC,115007.00,A,2611.83719,S,02801.67162,E,0.577,51.07,080513,,,A*47
,M,0.577,,0,,0
,0,54.,20M*GG,2751,,,,726101$PV,,,,,530,19051904227
```

Figure 1: GPS Output Stream Using Serial Communication

Appendix C

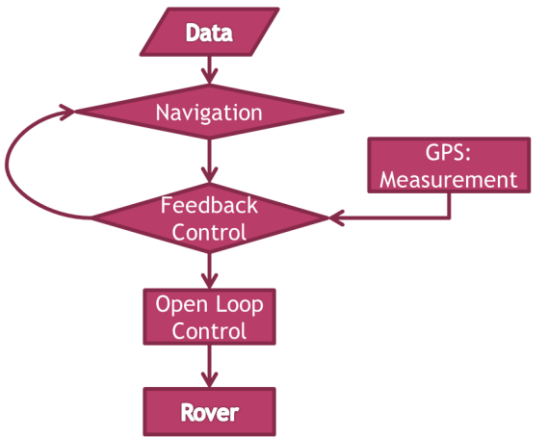


Figure 1: Flow Diagram of the Path Finding Algorithm

***Sudo code**:*

```
General order
Initialisation
        Find goal
        drive forward
        determine "direction"
        goalFind = 0
Running
        turn & drive
        if goalFind = 1
                find goal + 1
                goalFind =0
        end
        acquire position
        check angle
        check if success
        if success
                goal = goal + 1
                goalFind = 1
        end

Issue that may arise in order of probability:
###
Check angle after movement, else every turn/drive adds an error
###
Crossing goal latitude/longitude
if condition: latP > latS
        if latP >= latG
                latG = max(latC,latB)
        end
if condition: latP < latS
        if latP <= latG
                longG = min(latC,latB)
        end
similarly for longitude.
###
Crossing wall
Check proximity to wall AC
```

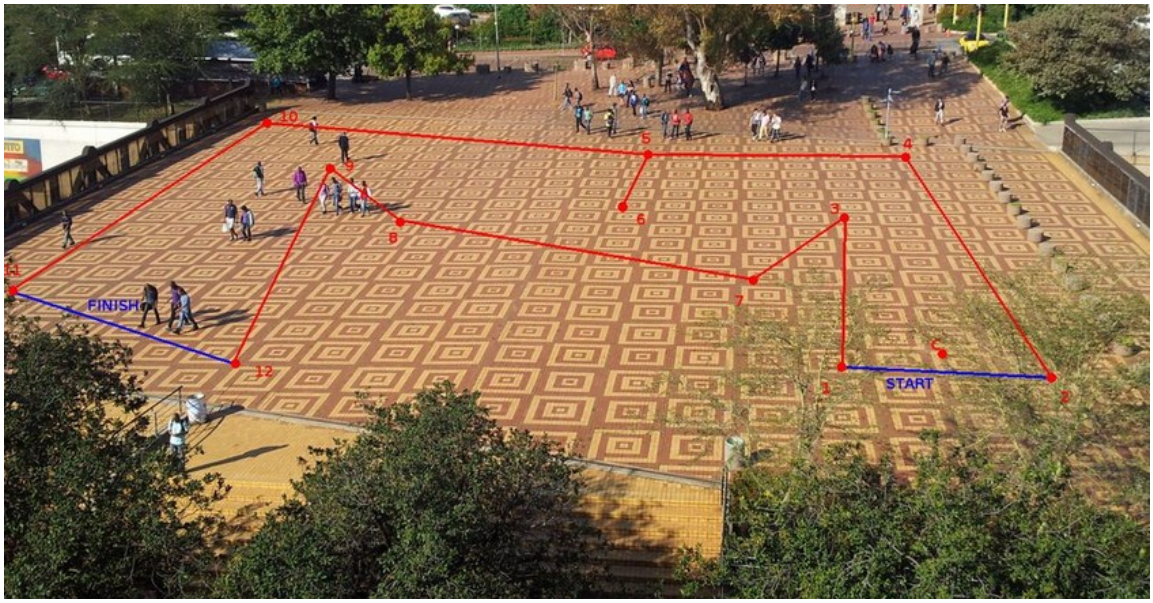


Figure 2: Map coordinates illistration

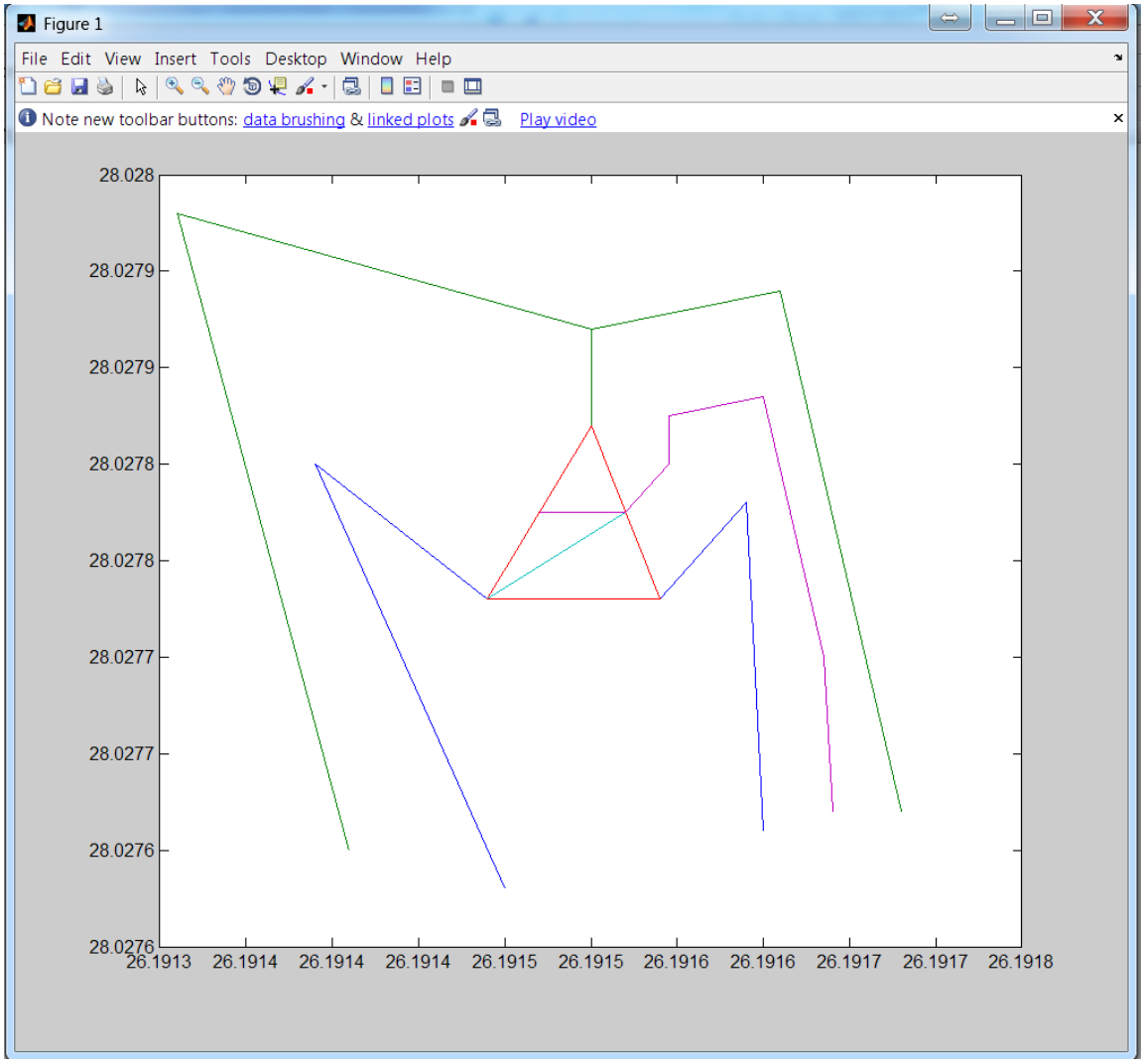


Figure 3: Visual Matlab Map Simulation

MATLAB code:

```
clc;
%close all;
%clear all;

latIn = [28.02766, 28.02783, 28.02778, 28.02778, 28.02785, 28.02763];
longIn = [26.19165, 26.19164, 26.19159, 26.19149, 26.19139, 26.19150];
latOut = [28.02767, 28.02794, 28.02792, 28.02787, 28.02792 ,28.02798, 28.02765];
longOut = [26.19173, 26.19166, 26.19155, 26.19155, 26.19155, 26.19131,
26.19141];
latP= 28.02767;
longP = 26.19169;
longVert =  [26.19165, 26.19173, 26.19164, 26.19166, 26.19155, 26.19155,
26.19159, 26.19149, 26.19155, 26.19139, 26.19131, 26.19141, 26.19150];
latVert = [28.02766, 28.02767, 28.02783, 28.02794, 28.02792, 28.02787, 28.02778,
28.02778, 28.02792, 28.02785, 28.02798, 28.02765, 28.02763];

longPath = longP;
latPath = latP;
longA = longVert(1,1);
latA  = latVert(1,1);
longB = longVert(1,2);
latB  = latVert(1,2);
longC = longVert(1,3);
```

```
latC  = latVert(1,3);
longD = longVert(1,4);
latD  = latVert(1,4);

longABC = [longA, longB, longC , longA];
latABC  = [latA , latB , latC  , latA ];
longPC  = [longP, longC];
latPC   = [latP , latC ];

if (longA - longD)^2 + (latA - latD)^2 < (longB - longD)^2 + (latB - latD)^2
    longB = longA;
    latB = latA;
end

longG    = abs(longB - longC)/2 + min(longB,longC);
latG     = abs(latB - latC)/2   + min(latB,latC);
longPath = [longPath, longG];
latPath  = [latPath, latG];
plot(longIn, latIn, longOut, latOut, longABC, latABC, longPC, latPC, longPath,
latPath);

vertN = 5;
vertM = size(longVert) + 1;
evalResponse = input('prompt');
while vertN < vertM(1,2)
    longA = longC;
    latA = latC;
    longC = longD;
    latC  = latD;
    longD = longVert(1,vertN);
    latD  = latVert(1,vertN);
    vertN = vertN + 1;
    longP = longG;
    latP = latG;

    longABC = [longA, longB, longC , longA];
    latABC  = [latA , latB , latC  , latA ];
    longPC  = [longP, longC];
    latPC   = [latP , latC ];
    longPD  = [longP, longD];
    latPD   = [latP , latD ];

    if (longA - longD)^2 + (latA - latD)^2 < (longB - longD)^2 + (latB - latD)^2
        longB = longA;
        latB = latA;
    end

    longG    = abs(longB - longC)/2 + min(longB,longC);
    latG     = abs(latB - latC)/2   + min(latB,latC);
    longPath = [longPath, longG];
    latPath  = [latPath, latG];

    plot(longIn, latIn, longOut, latOut, longABC, latABC, longPC, latPC,
longPath, latPath);
    evalResponse = input('prompt');
end


longP = longG;
latP = latG;
longA = longC;
latA = latC;
longC = longD;
latC = latD;
```

```
longABC = [longA, longB, longC , longA];
latABC  = [latA , latB , latC  , latA ];
longPC  = [longP, longC];
latPC   = [latP , latC ];
longPD  = [longP, longD];
latPD   = [latP , latD ];

longG    = abs(longA - longC)/2 + min(longA,longC);
latG     = abs(latA - latC)/2   + min(latA,latC);
longPath = [longPath, longG];
latPath  = [latPath, latG];

plot(longIn, latIn, longOut, latOut, longABC, latABC, longPC, latPC, longPath,
latPath);
evalResponse = input('prompt');
plot(longIn, latIn, longOut, latOut, longPath, latPath);
```

Appendix D

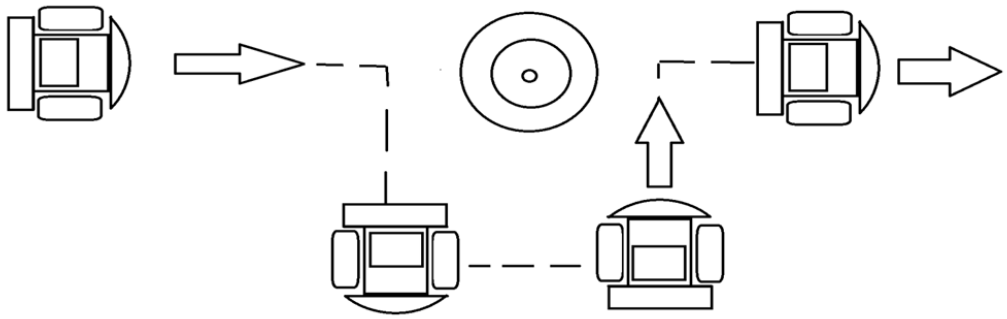


Figure 1: Depiction of Object Avoidance Algorithm

Appendix E

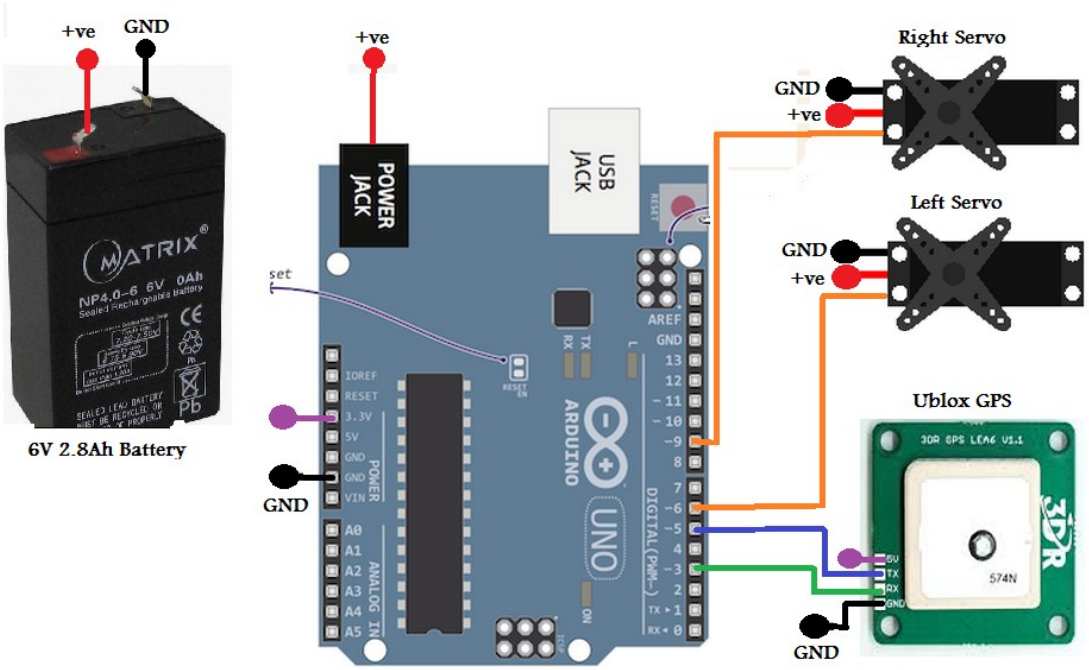


Figure 1: Connection Diagram of rover components