

ELE494-08

Autonomous Robotic Systems

Project Final Report

Nasir Khalid

Yousif Khaireddin 63618

Dr. Shayok Mukhopadhyay

**Introduction**

Throughout the project we initially aimed to build a robot that can survey a room for the location of maximum intensity. The goal was to continuously measure the light intensity which the robot goes around a room. This information would be fed back to a server which would plot all this information in real-time. Once the room has been completely inspected, the robot would then settle at the position of maximum light. Although we weren’t able to complete the entirety of this project, we were able to essentially accomplish a great deal of work that lay the foundations for the rest of the project. To measure the position of the robot we decided to utilize a complementary filter to combine the readings of an encoder and an accelerometer for a better estimate. We would then utilize Ackerman’s steering to get the robot to move as required.

**Objective**

**WRITE OBJECTIVE**

**Methodology**

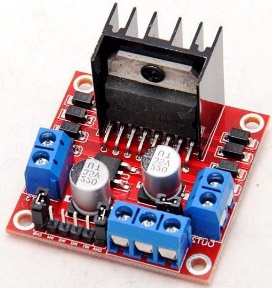
**Hardware Specifications**

For the actual robot chassis and construction, we bought and built the small car that can be seen in figure 1 and decided to repurpose it for our project.

**

*Figure 1:Robot Chassis, Wheels, & Motors*

To be able to control the direct as well as speed of rotation of each of the wheels, we used a Dual H-Bridge which can be seen in figure 2. This was adequate since it provided the had the required current rating for the motors stall torque current draw.

**

*Figure 2: L298N Dual H-Bridge motor controllers*

As for the onboard controller, we had initially started by using a regular Arduino; however, once we decided to incorporate the real-time tracking of information, we then decided to switch to the NodeMCU that can be seen in figure 3. The reason behind this will be explained in the following sections.

**

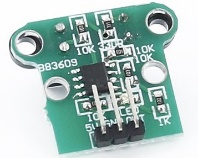
*Figure 3: ESP8266 based NODEMCU microcontroller*

Next major piece of hardware required would be the actual accelerometer we decided to use. Due to the limited number of ports on the NodeMCU, we had to choose a very small accelerometer that required a maximum of three pins. Luckily, we were able to find the one shown in figure 4 which perfectly meets these criteria.

**

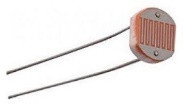
*Figure 4: MPU6050 3 axis accelerometer/gyroscope*

As previously discussed, we hope to combine the reading of the accelerometer with that of an encoder. For this, we decided to buy the encoder that can be seen in figure 5. The speed measuring module attaches to the rotary encoder and emits a pulse periodically which can be used to give an estimate of speed.

** 

*Figure 5: HC-020K Speed Measuring Module and Rotary Encoder*

Of course, for our actual project to have taken place we would require something that can give us an understanding of light intensity in a certain area. For this, we decided to use the LDR sensor shown in figure 6.

**

*Figure 6: Photoresistor LDR CDS 5mm*

Finally, to power everything, we decided to use the power bank shown in figure 7 due to availability.

**

*Figure 7: Huawei 6700 mAH power bank*

**Robot Assembly**

Once all the parts have been tested separately, it was time to combine everything together to build the final robot. This took some time to ensure that all the wiring and soldering was correct and that the all the parts were oriented correctly and still operating as required. The final construction can be seen in the figures below.

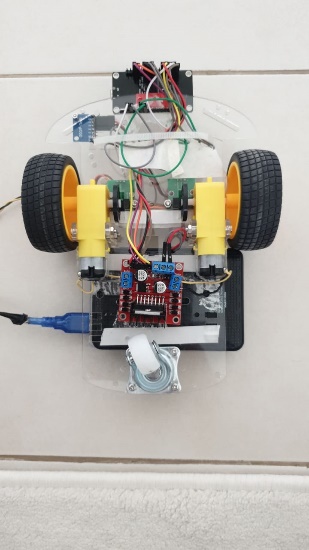
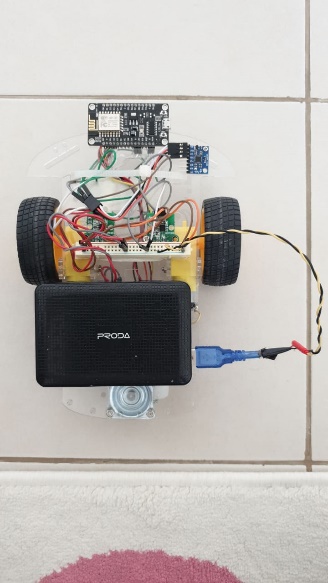
 

Figure 10: Final Robot

**Position Estimation**

Regarding the actual localization of the robot, we have decided to try and construct a complementary filter and combine our encoder and accelerometer readings for a better estimate of position. This will be done by reading each of them separately, changing their readings into position through the required integrations necessary, then performing some sort of weighted average to combine the results. The difficulty here will be trying to understand which of the two sensors performs better to properly choose the weights.

**Robot Movement**

Once the robot has successfully learned how to locate its position relative to an axis, it must also learn to move to any specific point, as required. For this, we decided to implement Ackerman’s steering since it eliminated the need for any PID gain tuning and should still provide satisfactory results. It is important to note that the route taken from point A to point B will be completely random and that there’s no guarantee that it will be the same every time. For our application, however, this shouldn’t be an issue.

**Real-time Communication**

Finally, as for the real-time aspect of the project, we hope to connect to the NodeMCU wirelessly and execute a command that will continuously retrieve required information about the robots parameters and plot this information in real-time. This was the reason we decided to switch from an Arduino to the NodeMCU. Its inbuilt functionality which allows it to become an access point which a computer could connect to and easily access data will vastly facilitate the transfer of data between the robot and our server.

**GitHub Repository**

One important thing to note is that throughout the project, we ensured to create a GitHub repository which helped us organize and keep track of each of our work and contributions as well as help us organize the project at whole.

This can be found here <https://github.com/NasirKhalid24/ELE494-08-Project>

**Experimentation**

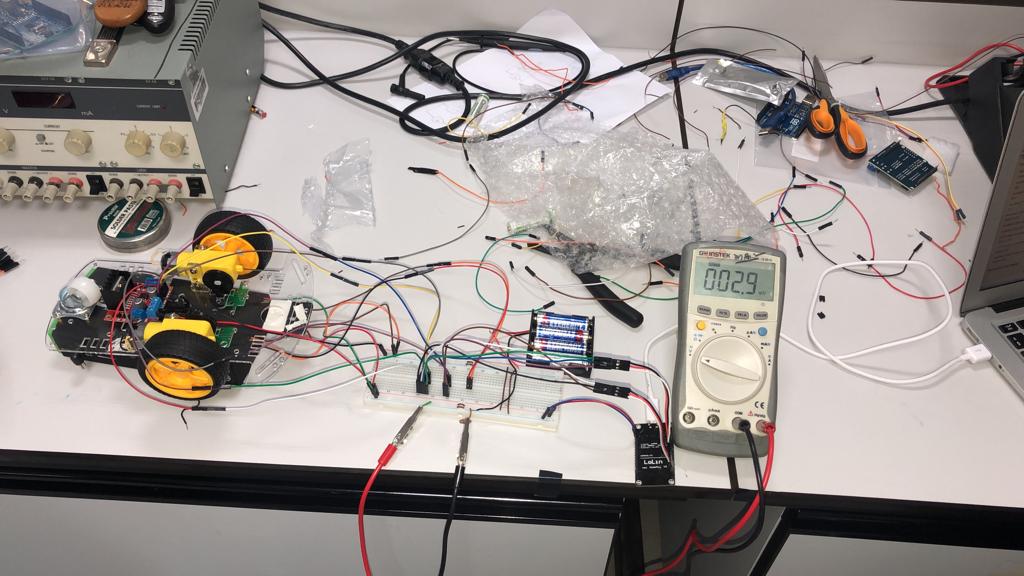
**Parts Operation**

Once all the parts have arrived, it was essential to test that each of them was actual operating properly. For this, we ensured to construct different testing procedures that will help us ensure that the operation of each part is sensible and appropriate and this can be seen in the following.

* Motors and Motor Driver

For this we started by applying a voltage to the motors ensuring their proper rotation in both directions. We then varied these voltage inputs to ensure that the motors are in fact responding to changes in voltage values. This is important as it enables us to perform speed control.

Once that has been completed, we then connected the motors to the motor driver and fed the motor drive the required pin configurations to turn the motors in both directions as well as ensured to test that different inputs to the enable pins would cause a change in motor speed; thus, further confirming our ability to perform speed control. The setup for these tests can be seen in figure 8.



*Figure 8: Preliminary Testing*

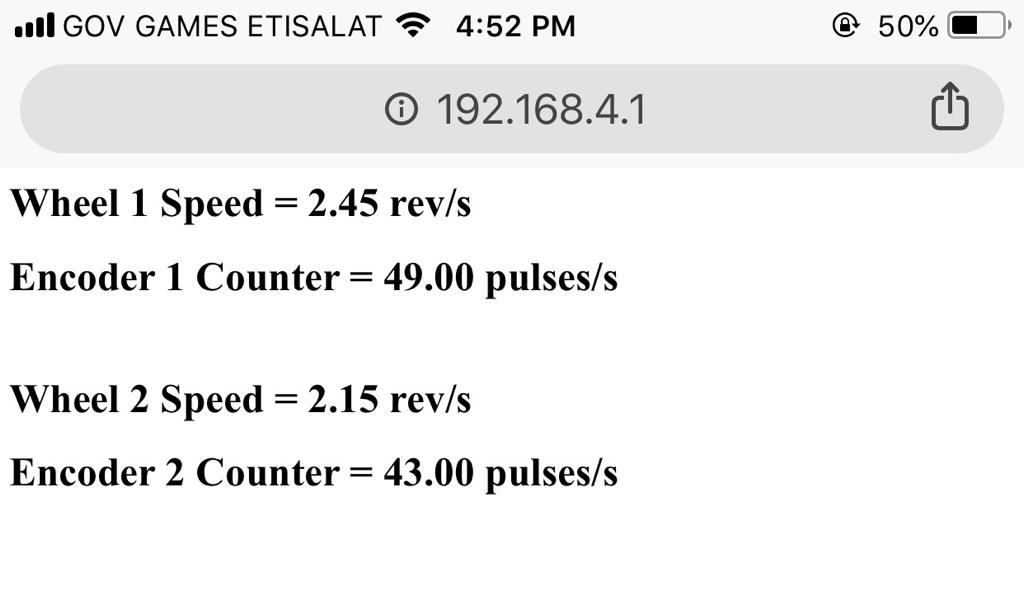
* Sensors

One of the first test we made was to ensure that accelerometer values were sensible. This was done by keeping it constant in all directions and placing a different face pointing downwards. The goal here was to ensure that whenever a specific side faced downwards, its acceleration was roughly 9.81 m/s2 while everything else was roughly 0 m/s2 (no motion).

Once that has been completed, it was important to test the encoder readings, this was done by manually rotating the wheel through 1 rotation and ensuring that the counter value reaches 20. This is the case since there are 20 slots in our rotary encoder as can be seen in figure 5.

* Data Transfer

The next major section we had to test is the ability for us to connect to the NodeMCU wirelessly and extract data in real-time. This was done by running a loop that sent some values over to the computer and cross checking that these values were as expected. This is illustrated in figure 9.



*Figure 9: Data Transfer*

**Calibration**

Due to mismatches in design motor design, imperfect weight distributions, as well as a number of other factors, we were required to find a ratio between the wheel pwm inputs that would ensure the robot behaves as required. Without this step, feeding both motors the same voltage will cause the car to tilt towards one direction rather than go straight. For this, a number of tests and trials were done while varying a scale factor until we found the correct one which will ensure proper movement. The code showing this factor can be seen below.

//note: the 0.9 factor is just used for calibration since the motors aren't identical

analogWrite(EN1, pwm\_l\*0.9);

analogWrite(EN2, pwm\_r);

**Speed Control**

Once the above has been completed, we then moved on to running a number of tests to ensure that we can control each motors speed through altering the PWM input which will in turn give us the ability to turn left and right as required.

**Position Estimation**

As previously discussed, to properly detect the position of our robot, we decided to combine the readings of an accelerometer and an encoder using a complementary filter. In this section, we were faced by a number of choices. One of the first choices we had to make is which of the following two methods to utilize when designing the filter:

Method 1:

* integrate the accelerometer once
* merge that reading with the encoder speed to obtain a *speed estimate*
* integrate the weighted average to obtain a measurement of position

Method 2:

* integrate the accelerometer twice
* integrate the encoder once
* merge the results using a weighted average to obtain a *position estimate*.

One way to resolve this is by just choosing the method that has the least number of integrations before yielding the result. This argument is purely based on the discussions we had in class discussing the inaccuracies and problems that direct integration can result in. However, as can be seen above both methods need at least two integrations so they are identical in this aspect.

To resolve this, we decided to implement both options to see which would yield better results. Unfortunately, since both our encoder as well as our accelerometer readings were extremely inaccurate and very noisy to begin with, neither of them gave us any concrete results, so we decided to just go with method 2 since its slightly more convenient because the final output of the filter is directly position.

Of course, the next choice we had to make was choosing the weights of this filter. To do this, we varied them across several different cases including some extremes of (0.95 & 0.05) just to see the behavior of the car. Although still not great due to the extremely faulty hardware, our best results were obtained using 0.7 on the encoder and 0.3 on the accelerometer. This kind of makes sense since the accelerometer undergoes two accelerations while the encoder only goes through one.

One important thing to note is that when we had initially started testing, we initially assumed a constant timestep of 100ms for the integrations which is very wrong especially since at each loop, depending on certain values there are specific delays placed which can significantly contribute to this timestep. For this, we later ensured to internally measure this timestep and fix these wrong assumptions.

**Ackerman’s Steering**

Even though our estimate of position was very wrong, we still decided to carry on and build the code required for Ackerman’s steering. In this section we had to decide the choice of h and K. Initially, we had chosen a h value of 0.5 and a K matrix of [0.5 0 ; 0 0.5].

As per our discussion with Dr. Shayok, we have been told that a larger h and a smaller K would lead to a smooth convergence. For this, we then ended up with an h of 1 and a K of [0.2 0; 0 0.2].

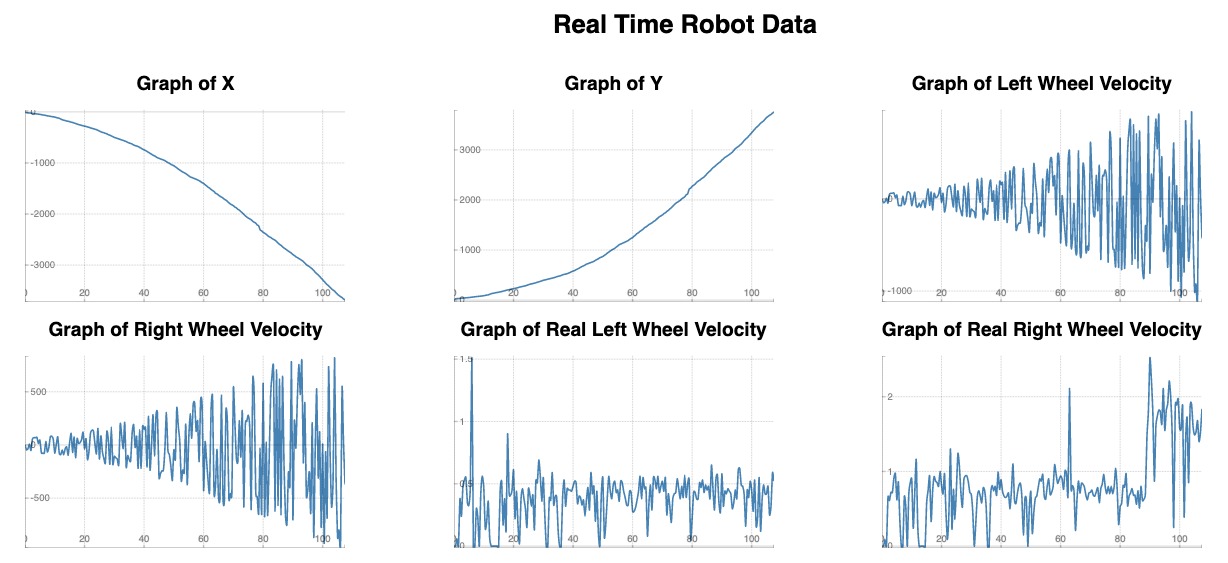
Of course since our position estimates were not correct by any means, we were never able to verify the actual workings of this code; however, by cross referencing with other online sets of code as well as our homework solutions, we are confident it would. This is the case since the robot does react to errors as can be seen in figure XX where as the x and y values get further and further from the point they are intended to reach, the wheel velocities get higher and higher. This indicates that there really is a reaction to error.

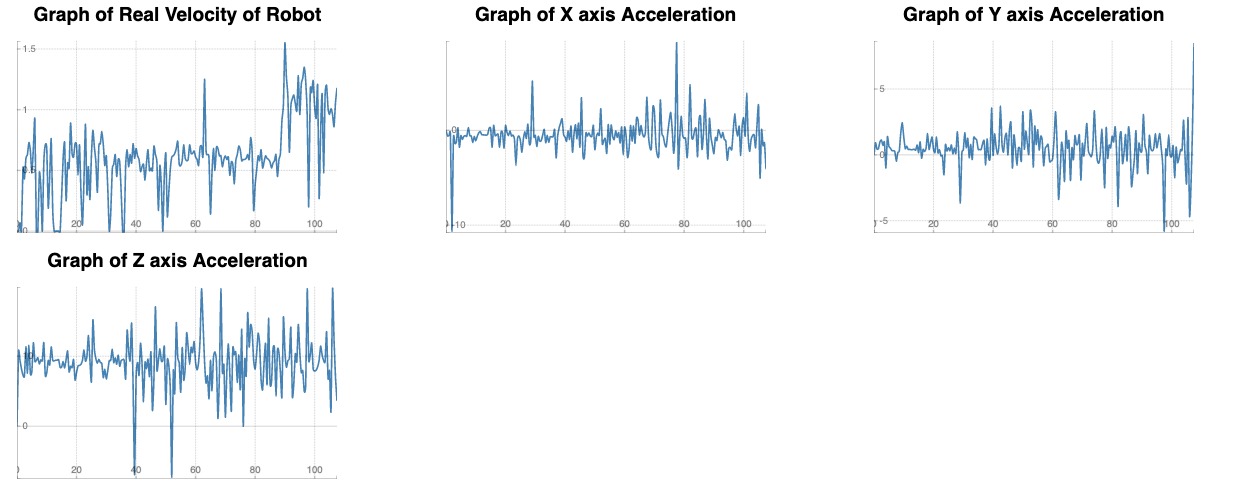
**Data Transfer**

NASIR YOU NEED TO WRITE IN THIS PART AND TALK ABOUT ALL THE JAVASCRIPT STUFF ANS SHOW DIFFERENT THINGS YOU TRIED

**Results**

Since we had only gotten to building Ackerman’s steering and the complementary, there wasn’t much we could show in terms of results. However, as can be seen in the plots below we are definitely tracking the values the robot is sending in real-time and are accurately plotting them as time passes. In this run, the robot was required to go to point (200, 200) which it obviously did not. Had the complementary filter output been somewhat accurate we strongly believe that it would’ve been possible.





*Figure ??: Results*

**Conclusion**

**WRITE THIS**

**Appendix**

**Robot Operation Code**

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <ESP8266WebServer.h>

#include <Wire.h>

#include <ESP8266HTTPClient.h>

#include <time.h>

#define PI 3.1415926535897932384626433832795

//PIN LAYOUT

//EN1 --> D3

//IN1 --> RX

//IN2 --> TX

//IN3 --> D0

//IN4 --> D7

//EN2 --> D4

//ENC1 -->D1

//ENC2 --> D6

// D8 IS BAD. DO NOT USE!

//Pin Definitions

#define ENCODER1 5 //[D1]

#define ENCODER2 12 //[D6]

#define EN1 0 //[D3]

#define IN1 3 //[RX]

#define IN2 1 //[TX]

#define EN2 2 //[D4]

#define IN3 16 //[D0]

#define IN4 13 //[D7]

#define PI 3.1415926535897932384626433832795

#define radius 0.032955

//requirements to run robot

**double** count1 = 0;

**double** count2 = 0;

**double** rev1 = 0;

**double** rev2 = 0;

**double** v1, v2, v\_actual;

**double** ts = 0;

**double** tremove = 0;

**bool** flag\_p\_l = 1;

**bool** flag\_p\_r = 1;

//requirements to run ackremenas steering

#define h 1

#define k1 0.2

#define k2 0.2

#define l 0.157

#define w1 0.05

#define w2 0.95

#define w3 0.05

#define w4 0.95

//initial conditions

**double** x\_1= 0;

**double** x\_2= 0;

**double** y\_1= 0;

**double** y\_2= 0;

**double** Vx = 0;

**double** Vy = 0;

**double** x = 0;

**double** y = 0;

**double** theta = 0;

**double** v = 0;

**double** w = 0;

**double** xh = 0;

**double** yh = 0;

**double** e1 = 0;

**double** e2 = 0;

**double** vl = 0;

**double** vr = 0;

**double** pwm\_l = 0;

**double** pwm\_r = 0;

//final point

**double** xr = 200;

**double** yr = 200;

// Connection credentials

**const** **char**\* WIFI\_NAME = "Daedalus";

**const** **char**\* WIFI\_PASS = "flightoficarus";

**const** **char**\* host = "192.168.43.177";

// Select SDA and SCL pins for I2C communication

**const** uint8\_t scl = 4; //[D2]

**const** uint8\_t sda = 14; //[D5]

// MPU6050 Slave Device Address

**const** uint8\_t MPU6050SlaveAddress = 0x68;

// sensitivity scale factor respective to full scale setting provided in datasheet

**const** uint16\_t AccelScaleFactor = 16384;

**const** uint16\_t GyroScaleFactor = 131;

// MPU6050 few configuration register addresses

**const** uint8\_t MPU6050\_REGISTER\_SMPLRT\_DIV = 0x19;

**const** uint8\_t MPU6050\_REGISTER\_USER\_CTRL = 0x6A;

**const** uint8\_t MPU6050\_REGISTER\_PWR\_MGMT\_1 = 0x6B;

**const** uint8\_t MPU6050\_REGISTER\_PWR\_MGMT\_2 = 0x6C;

**const** uint8\_t MPU6050\_REGISTER\_CONFIG = 0x1A;

**const** uint8\_t MPU6050\_REGISTER\_GYRO\_CONFIG = 0x1B;

**const** uint8\_t MPU6050\_REGISTER\_ACCEL\_CONFIG = 0x1C;

**const** uint8\_t MPU6050\_REGISTER\_FIFO\_EN = 0x23;

**const** uint8\_t MPU6050\_REGISTER\_INT\_ENABLE = 0x38;

**const** uint8\_t MPU6050\_REGISTER\_ACCEL\_XOUT\_H = 0x3B;

**const** uint8\_t MPU6050\_REGISTER\_SIGNAL\_PATH\_RESET = 0x68;

int16\_t AccelX, AccelY, AccelZ, Temperature, GyroX, GyroY, GyroZ;

**double** Ax, Ay, Az, T, Gx, Gy, Gz;

**void** High\_Callback() {

count1 += 1;

}

**void** Low\_Callback() {

count2 += 1;

}

**void** setup() {

Serial.begin(115200);

Serial.print("STARTED ROBOT");

//Connect to WiFi

Connect2Wifi();

// Initialize MPU

Wire.begin(sda, scl);

MPU6050\_Init();

//Defining PIN directions

pinMode(EN1, OUTPUT);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(EN2, OUTPUT);

pinMode(IN3, OUTPUT);

pinMode(IN4, OUTPUT);

pinMode(ENCODER1, INPUT);

pinMode(ENCODER2, INPUT);

delay(1000);

analogWrite(EN1, 1024\*0.9);

analogWrite(EN2, 1024);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

attachInterrupt(digitalPinToInterrupt(ENCODER1), High\_Callback, RISING);

attachInterrupt(digitalPinToInterrupt(ENCODER2), Low\_Callback, RISING);

}

**void** loop(){

**if**(WiFi.status() != WL\_CONNECTED){

Connect2Wifi();

}

//obtaining the timestep between loops

ts = (millis() - tremove)/1000;

tremove = millis();

//obtain the speed of wheels in revolutions/sec using the count values from the interrupt functions

rev1 = count1/(20\*ts);

rev2 = count2/(20\*ts);

//change these speeds into m/s for each wheel

v1 = rev1\*2\*PI\*radius;

v2 = rev2\*2\*PI\*radius;

//obtain actual robot speed

v\_actual = (v1 + v2)/2;

//obtaining positions using integration of encoder velocity

x\_1= x\_1+ ts\*v\_actual\*cos(theta);

y\_1= y\_1+ ts\*v\_actual\*sin(theta);

theta = theta + ts\*w;

//reset counter values in preperation for next v measurement

count1 = 0;

count2 = 0;

//obtain accelerometer and gyroscope values

Read\_RawValue(MPU6050SlaveAddress, MPU6050\_REGISTER\_ACCEL\_XOUT\_H);

//divide each with their sensitivity scale factor

Ax = (**double**)AccelX\*9.81/AccelScaleFactor;

Ay = (**double**)AccelY\*9.81/AccelScaleFactor;

Az = (**double**)AccelZ\*9.81/AccelScaleFactor;

T = (**double**)Temperature/340+36.53; //temperature formula

Gx = (**double**)GyroX/GyroScaleFactor;

Gy = (**double**)GyroY/GyroScaleFactor;

Gz = (**double**)GyroZ/GyroScaleFactor;

//obtaining velocities using integration of accelerometer values

Vx = Vx + ts\*Ax;

Vy = Vy + ts\*Ay;

//obtain position using second integration of accelerometer

x\_2= x\_2+ ts\*Vx;

y\_2= y\_2+ ts\*Vy;

//obtaining an estimate of position using complementary filter

x = x\_1 \* w1 + x\_2 \* w2;

y = y\_1 \* w3 + y\_2 \* w4;

//implementing ackremans steering

xh = x + h\*cos(theta);

yh = y + h\*sin(theta);

e1 = xh - xr;

e2 = yh - yr;

v = (k1\*cos(theta) + k2\*sin(theta))\*e1;

w = ((-k1\*sin(theta)/h) + (-k2\*cos(theta)/h))\*e2;

//obtain required speed for each motor

vl = v + (l\*w/2);

vr = v - (l\*w/2);

//scaling the voltage values between 650 - 1024

pwm\_l = ((1024 - 750) \* (vl/25)) + 750;

pwm\_r = ((1024 - 750) \* (vr/25)) + 750;

//apply voltages to wheels

**if** (pwm\_l > 0){

//must switch off mototrs before switching direction

**if** (flag\_p\_l != 1){

analogWrite(EN1, 0);

flag\_p\_l = 1;

delay(100);

}

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

}**else**{

//must switch off mototrs before switching direction

**if** (flag\_p\_r == 1){

analogWrite(EN1, 0);

flag\_p\_r = 0;

delay(100);

}

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

pwm\_l = abs(pwm\_l);

}

**if** (pwm\_r > 0){

//must switch off mototrs before switching direction

**if** (flag\_p\_r != 1){

analogWrite(EN2, 0);

flag\_p\_r = 1;

delay(100);

}

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

}**else**{

//must switch off mototrs before switching direction

**if** (flag\_p\_r == 1){

analogWrite(EN2, 0);

flag\_p\_r = 0;

delay(100);

}

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

pwm\_r = abs(pwm\_r);

}

//apply required speed for each motor

//note: the 0.9 factor is just used for calibration since the motors aren't identical

analogWrite(EN1, pwm\_l\*0.9);

analogWrite(EN2, pwm\_r);

SendData(x, y, vl, vr, pwm\_l, pwm\_r, ts, count1, count2, rev1, rev2, v1, v2, v\_actual, Ax, Ay, Az, T, Gx, Gy, Gz);

delay(300);

}

// -------------------- ACCELEROMETER FUNCTIONS --------------------

**void** I2C\_Write(uint8\_t deviceAddress, uint8\_t regAddress, uint8\_t data){

Wire.beginTransmission(deviceAddress);

Wire.write(regAddress);

Wire.write(data);

Wire.endTransmission();

}

// read all 14 register

**void** Read\_RawValue(uint8\_t deviceAddress, uint8\_t regAddress){

Wire.beginTransmission(deviceAddress);

Wire.write(regAddress);

Wire.endTransmission();

Wire.requestFrom(deviceAddress, (uint8\_t)14);

AccelX = (((int16\_t)Wire.read()<<8) | Wire.read());

AccelY = (((int16\_t)Wire.read()<<8) | Wire.read());

AccelZ = (((int16\_t)Wire.read()<<8) | Wire.read());

Temperature = (((int16\_t)Wire.read()<<8) | Wire.read());

GyroX = (((int16\_t)Wire.read()<<8) | Wire.read());

GyroY = (((int16\_t)Wire.read()<<8) | Wire.read());

GyroZ = (((int16\_t)Wire.read()<<8) | Wire.read());

}

//configure MPU6050

**void** MPU6050\_Init(){

delay(150);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_SMPLRT\_DIV, 0x07);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_PWR\_MGMT\_1, 0x01);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_PWR\_MGMT\_2, 0x00);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_CONFIG, 0x00);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_GYRO\_CONFIG, 0x00);//set +/-250 degree/second full scale

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_ACCEL\_CONFIG, 0x00);// set +/- 2g full scale

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_FIFO\_EN, 0x00);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_INT\_ENABLE, 0x01);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_SIGNAL\_PATH\_RESET, 0x00);

I2C\_Write(MPU6050SlaveAddress, MPU6050\_REGISTER\_USER\_CTRL, 0x00);

}

// --------------------------------------------------------------------------------

**void** Connect2Wifi(){

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

WiFi.begin(WIFI\_NAME, WIFI\_PASS);

delay(4000);

}

**void** SendData(**double** X,**double** Y, **double** VL,**double** VR,**double** PWM\_L,**double** PWM\_R,**double** TS, **double** E1, **double** E2, **double** REV1, **double** REV2, **double** V1, **double** V2, **double** V\_ACTUAL, **double** AX, **double** AY, **double** AZ, **double** T, **double** OX, **double** OY, **double** OZ){

HTTPClient http;

http.begin("http://" + String(host) + ":5000/data");

http.addHeader("Content-Type", "application/x-www-form-urlencoded");

http.POST(

"x=" + String(X) + "&"

"y=" + String(Y) + "&"

"vl=" + String(VL) + "&"

"vr=" + String(VR) + "&"

"pwm\_l=" + String(PWM\_L) + "&"

"pwm\_r=" + String(PWM\_R) + "&"

"TS=" + String(TS) + "&"

"E1=" + String(E1) + "&"

"E2=" + String(E2) + "&"

"REV1=" + String(REV1) + "&"

"REV2=" + String(REV2) + "&"

"V1=" + String(V1) + "&"

"V2=" + String(V2) + "&"

"V Actual=" + String(V\_ACTUAL) + "&"

"AX=" + String(AX) + "&"

"AY=" + String(AY) + "&"

"AZ=" + String(AZ) + "&"

"T=" + String(T) + "&"

"OX=" + String(OX) + "&"

"OY=" + String(OY) + "&"

"OZ=" + String(OZ)

);

http.writeToStream(&Serial);

http.end();

}

**Server Code**