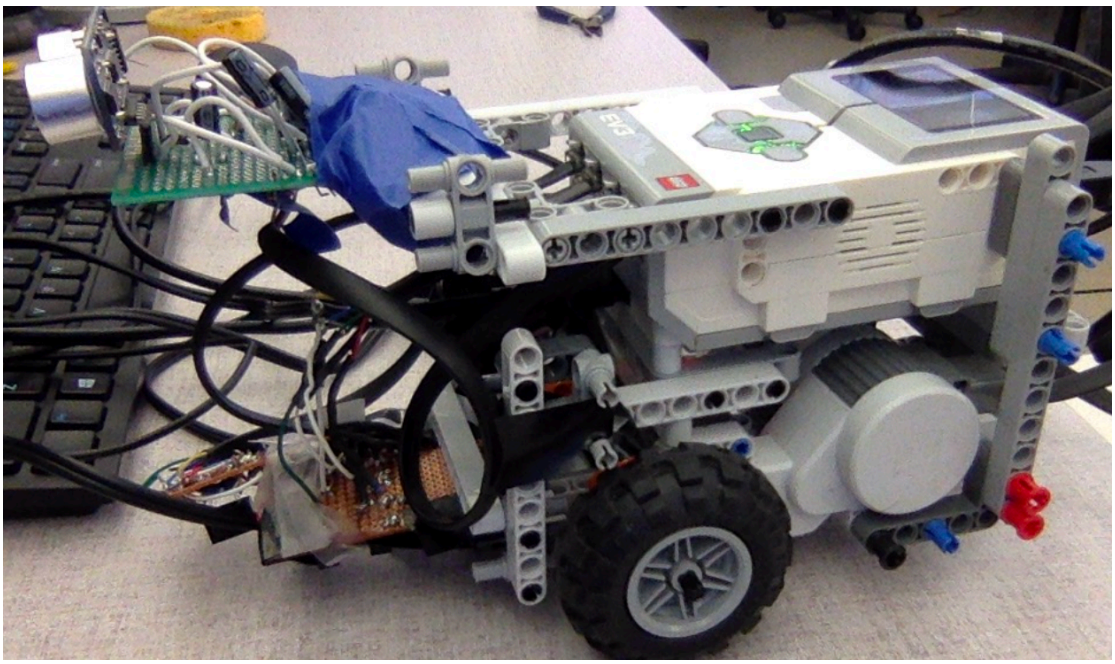
ECE 556 Project report

Himasai Polimetla [ID #200265383, Unity ID hpolime]
Binoy Thomas [ID # 200260021, Unity ID bthomas7]

Introduction

The objective of the project is to design an Unmanned vehicle for tracking a line using the PID control algorithm. The project consisted of 3 tasks. Each of the tasks used the sensors and motors in a way that all the requirements are met. The first task was to track a path using the PID controller. The second task was collision avoidance where we were asked to stop the bot as close to the block as possible. The third task was platooning which requires us to follow the TA's bot at a constant distance of 20 cm. The UV was designed using the LEGO EV3 brick and motors and the light sensors used in this project were EESF5-B and the ultrasonic sensor was the SRF05.



Figure(1): Design of the Unmanned vehicle

Project setup

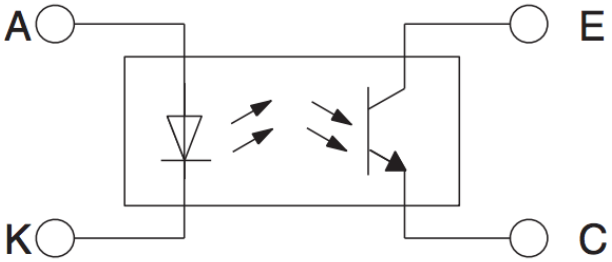
The Unmanned vehicle was designed using the Lego EV3. The brain of the robot is the EV3 brick which serves as the control center and the power station of the robot. The code for controlling the EV3 brick was written on MATLAB. The brick is powered by the EV3 rechargeable batteries. It can be recharged while inbuilt in a model, saving us the trouble of disassembling and reassembling a robot to replace batteries.

Figure(2): Design of Swivel wheel

The robot is a two-wheeled mobile robot. A small swivel provides third point contact to ensure stability. The robot is driven by two high speed EV3 Large Servo Motor coupled to wheels. The robot is battery powered, with the motors supplied with regulated power supply to ensure consistent motor power at all operating scenarios. The Large Motor is a powerful "smart" motor. It has a built-in Rotation Sensor with 1-degree resolution for precise control. The Large Motor is optimized to be the driving base on your robots.

Figure(3): EV3 Large motor

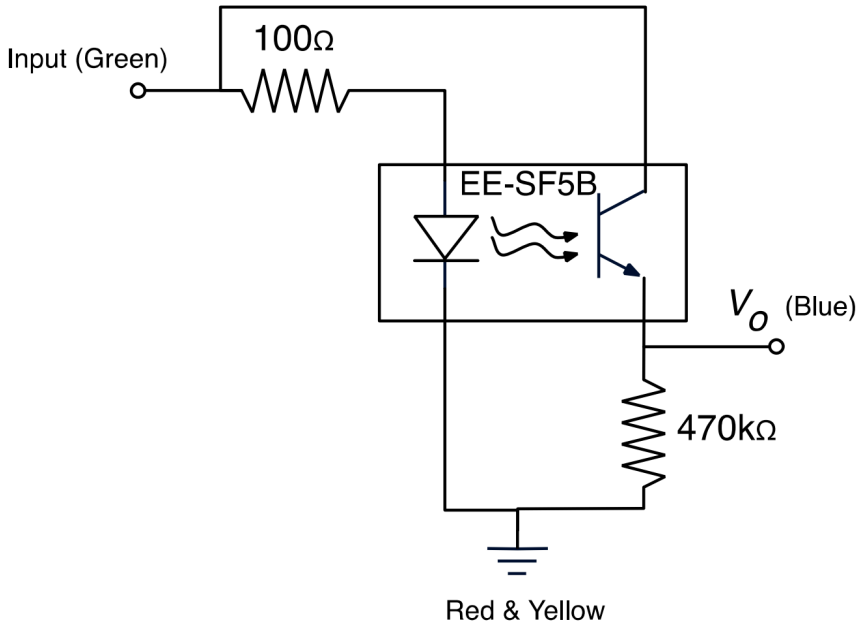
The line-tracking was done using the EE-SF5B photo-microsensor which is a reflective light sensor. The Color Sensor is a digital sensor that can detect the intensity of light that enters the small window on the face of the sensor. It measures the intensity of light reflected back from a light-emitting transmitter. The sensor uses a scale that ranges from 2850 (very dark) to 3450 (very light). The robot was programmed to move around on a white surface until a black line is detected. The light sensors were placed at right angle to the surface at a distance of about 1.2 cm. In this orientation most of the light transmitted from the transmitter is received by the receiver. The internal connections for the EESF5B are shown in figure(4). The circuit for the line tracking is given in figure(5).



Figure(4): EESF5-B

In the Figure(4):

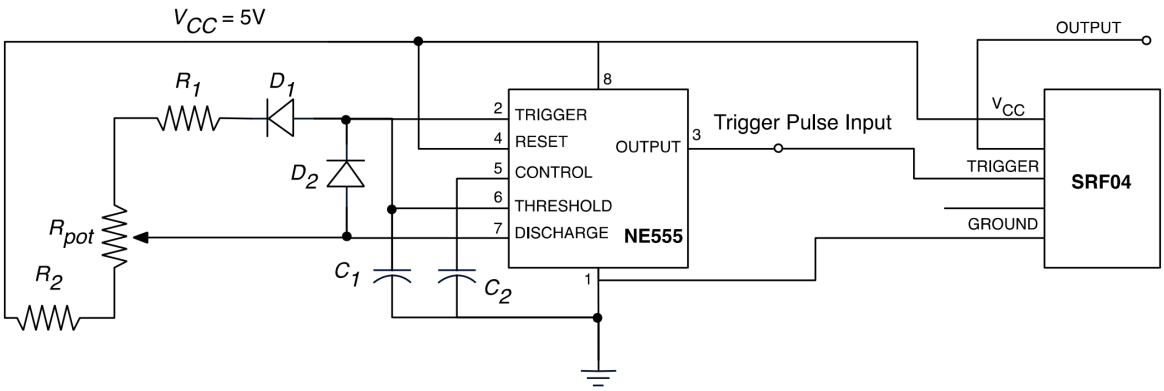
Terminal No.	Name
A	Anode
K	Cathode
C	Collector
E	Emitter



Figure(5): Circuit diagram for light sensor

The robot has an array of two EESF5 light sensors to detect a non-reflective (Black) line on a reflective (White) surface. The sensors are placed at a separation of 1 cm and the total sensor coverage is 2 cm. The sensor array is placed in front of the robot. The rays emitted by the light sensor fall on the surface. If the surface is reflective (White), the rays are reflected back to the receiver which conducts current. So, the potential difference across the device is reduced. If the surface is an absorbing plane (Black), the rays will be absorbed and current won't flow. The sensors are connected to the EV3 via the NXT connector.

The Ultrasonic Sensor is a digital sensor that can measure the distance from an object in front of it. It does this by sending out high-frequency sound waves and measuring how long it takes the sound to reflect back to the sensor. The sound frequency is too high for us to hear. Distance from an object was measured in centimeters. This allowed us to program our robot to stop at a certain distance from the block for Task2 and follow the TA bot in Task 3. The Ultrasonic sensor's circuit included an unstable multi-vibrator which was achieved using a 555 timer and the SRF05 was used as the ultrasonic sensor. The 555 triggers the SRF05 sensor. The circuit diagram for the ultrasonic sensor is given in figure(6).



Figure(6): Circuit diagram for ultrasonic sensor

Algorithm design

The output of the light sensor array is fed to the EV3 which calculates the error term from sensor values. The most convenient and reliable way to compute the error term is by dividing the weighted average of the sensor readings by the sum and normalizing the values. The formula used is given as follows:

$$Error = Normalized\ Light\ Sensor\ reading - Setpoint$$

The normalized sensor readings were obtained from the light sensor array and then they were normalized to a value between 0 and 100. The setpoint was taken to be 50 and these values were subtracted which gave us the error reading. We took the average of the sensor values and this was fed to the control algorithm. The main advantage of this approach is that the two sensor readings are replaced with a single error term which can be fed to the control algorithm to compute the motor speeds such that the error term becomes zero. Further, the error value becomes independent of the line width and so the robot can tackle lines of different thicknesses without needing any change in code.

Similarly, for the Ultrasonic sensor, the readings were obtained and used for calculating the distance between the bot and the block.

Task 1: Line tracking using PID control

The system has been tested initially with the classical PID controller implemented in parallel form. The well-known PID controller equation is shown here in continuous form.

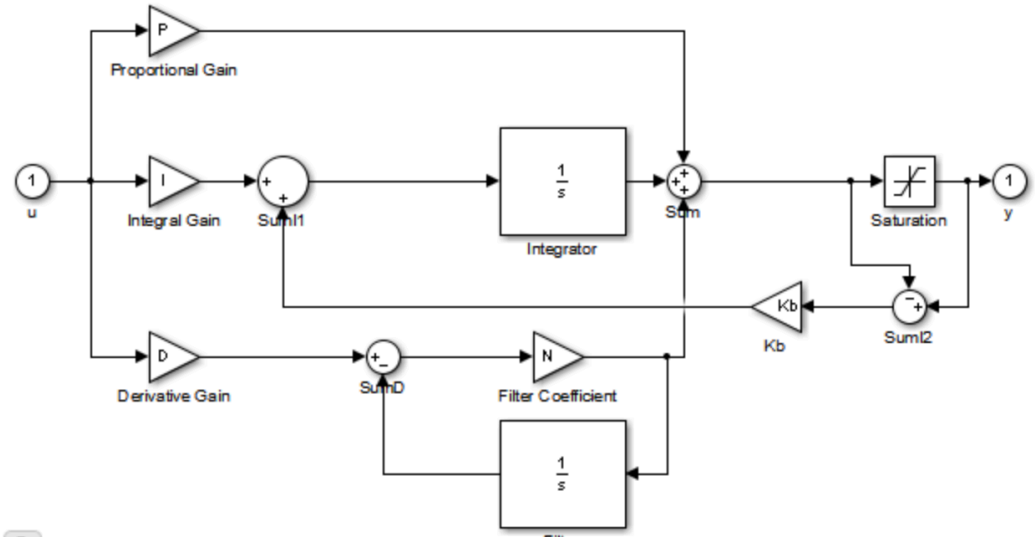
$$e(t) = k_p + k_i \int e(t)\,dt + \frac{d\,e(t)}{dt}$$

Where kP, kI and kD refer to proportional, integral and derivative gain constants respectively.

For implementation in discrete form, the controller equation is modified by using the backward Euler method for numerical integration.

$$e(n) = K_p * e(n) + K_i * \sum e(n) + K_d(e(n) - e(n-1))$$

Where KP, KI and KD refer to proportional, integral and derivative gain constants respectively



Figure(7): PID controller

The motor speeds are set depending on two parameters; the PID controller output and the maximum motor power, which determines the speed at which the robot is running. The motor speed equations are given as

If the sensor error parameter is negative,

Left motor = set_speed + e(n); Right motor = set_speed;

If the sensor error parameter is positive,

Right motor = set_speed + e(n); Left motor = set_speed;

From the above equations, it can be clearly seen that when the robot deviates to the right (negative error parameter), the left motor speed is reduced to make the robot move to the left. Similarly, when the robot deviates to the left (positive error parameter), the right motor speed is reduced to make the robot move to the right. Another notable point is that, when the sensor error parameter is negative, the controller output usually will also have the same polarity.

Task 2: Parking

The ultrasonic sensors were used for the parking task. The error readings of the ultrasonic sensor were calculated as

$$Error = Normalized\ Ultrasonic\ Sensor\ reading - Setpoint$$

The ultrasonic sensor readings were in the range of 200-400. These values were normalized to the centimeter range such that a reading of 20 is equal to 20 cm. When the distance reached 8 cm, the control stops the motors and breaks out of the loop. This ensures that the bot parks and stops as close to the block as possible.

Task 3: Platooning

The aim of this task to follow the TA's UV at a constant distance of 20 cm and to follow the bot properly. The ultrasonic sensors as well as the light sensors were used for this task. 2 light sensors were used for tracking the path and one light sensor was used to detect the red tape. The ultrasonic sensors were used to measure the distance from the TAs bot. The task only asks us to move in a straight line. So we used only a proportional controller for line tracking. The distance from the TA's UV were constantly measured and supplied to the control algorithm. The change in the distance was also measured which constitutes the speed of the TA's UV.

Algorithm development

Tasks 1 and 2: Line tracking using PID control and Collision Avoidance

The PID control was used for line tracking. The step by step implementation of the PID control is as follows.

Initially, a simple proportional controller was tested with a straight line. It was seen that the robot oscillates greatly to the left and to the right while tracking the line and the transient response was considerably slow.

Increase of the Proportional Gain KP improved the speed of the transient response but the oscillations became worse. Decreasing the gain reduced the oscillations but at the cost of more sluggish transient response. The performance became even worse when the straight line was replaced by a curved path because of the addition of steady state error. The simple P-controller was not suitable for the system. So, the derivative and integral terms were added to improve the performance.

In PID control, the derivative term is used to reduce the overshoot and to improve the transient response. But the derivative term is extremely sensitive to noise in the sensor readings.

Since the sensor error term is affected by random errors due to non-linearity, the derivative gain had to be kept low in order to prevent rapid motor speed changes.

With this PD controller, the line tracking system performance is fairly satisfactory for straight lines. But for curved paths, the performance was not very satisfactory due to steady state error. As a result, the overall motor speed had to be reduced for the robot to be stable when negotiating turns and curves. This reduced overall robot speed.

Integral term is usually used to reduce steady state error in a control system. Addition of the integral term reduces steady state error when the path is a simple curve. But, when the path is made up of various curves, straight lines and turns, integral control action becomes problematic.

Normally, integral control action is used to reduce steady state error due to a constant disturbance (like friction in a motor speed control system). But in this system, the disturbance due to line curvature is not constant because the line characteristics changes continuously. So, direct implementation of integral control is not effective.

The implementation of the Integral Term in Classical PID controller was unsuitable for line tracking robot because of the problems when the curvature of the line changes. In order to solve this problem, only the last ten error values are summed up instead of adding all the previous values.

This creates an integral control effort that strives to counteract steady state errors in curved paths but is also adaptive to changes in curvature of the track. This technique provides satisfactory performance for line tracking.

The integral term only considered the sum of the last 5 values and not the sum of all error values. The is the modified version of the PID controller which helps because it only considers a few values of errors and not all the values of errors.

The Ultrasonic sensor was used to stop the Vehicle before the block at the end of the track.

Task 3: Platooning

For this task, we believe that the difference between the current ultrasonic sensor reading the previous reading is proportional to the speed of the Moving Vehicle and used this parameter to estimate its speed. We decided to control the speed based on criteria like the speed of the Moving Vehicle (KMV), the distance between the Unmanned Vehicle and the Moving Vehicle (K_{sep})and the error derived for line tracking (KP).
In the initial case, when the separation between the two vehicles is greater than 25cm, K_{sep} is used mainly to control the speed of the vehicle such that the Unmanned Vehicle initially speeds up to reach the optimal separation that is required between the UV and the Moving Vehicle.

Once the UV is within the required range from the Moving Vehicle, the K_{MV} parameter dominates the control part of the UV's speed. This is to ensure that the UV adjusts to the changing speed of the Moving Vehicle.

The K_P parameter was used to ensure that the UV tracked the given path based on the error derived from the light sensor.

Using these cases, we successfully developed an algorithm to follow the UV_{TA} constructed by TAs moving at varying speeds in the front and stop on the red line at the end of the track.

Results and Discussions

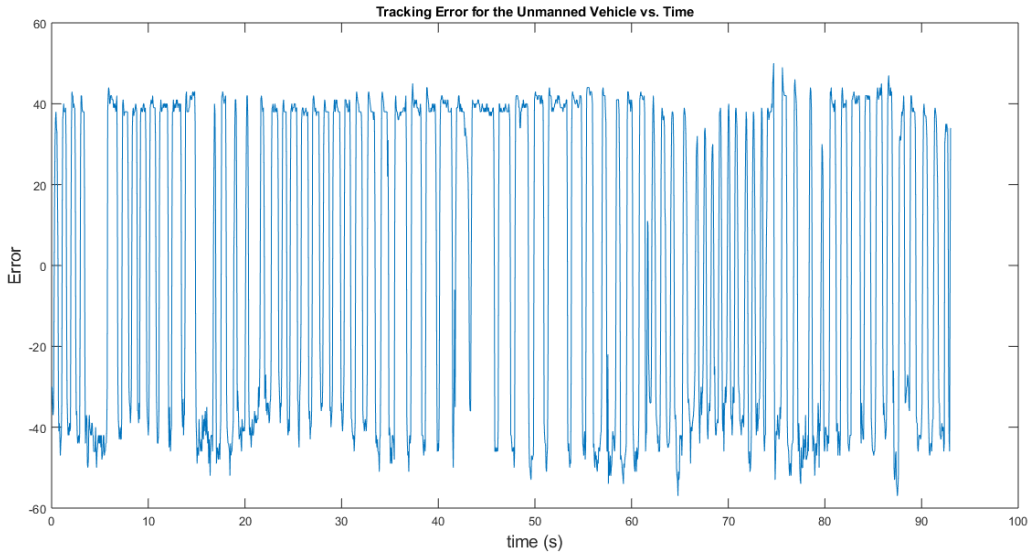
Task 1

For this task, we chose our PID control parameters to be:

$K_P = 1.8$, $K_D = 0.08$, and $K_I = 5$

We tested our Unmanned Vehicle on a path laid using seven of the given modules and found that the total time taken to travel was 93 seconds.

Plot:



Figure(8): Plot of Task 1

Discussions

We believe that the constant change in the errors in the plot is in a small range and that this oscillations in the value of errors is to properly track the given path. To better understand these errors, we calculated mean square error for different PID parameters. One such case was to tune K_D parameter. To increase smoothness, we decided to increase K_D to 8. Though this improved the smoothness to an extent, the time taken to cover the path also increased. Thus, we decided to use the initial K_D i.e 4 for our experiment.

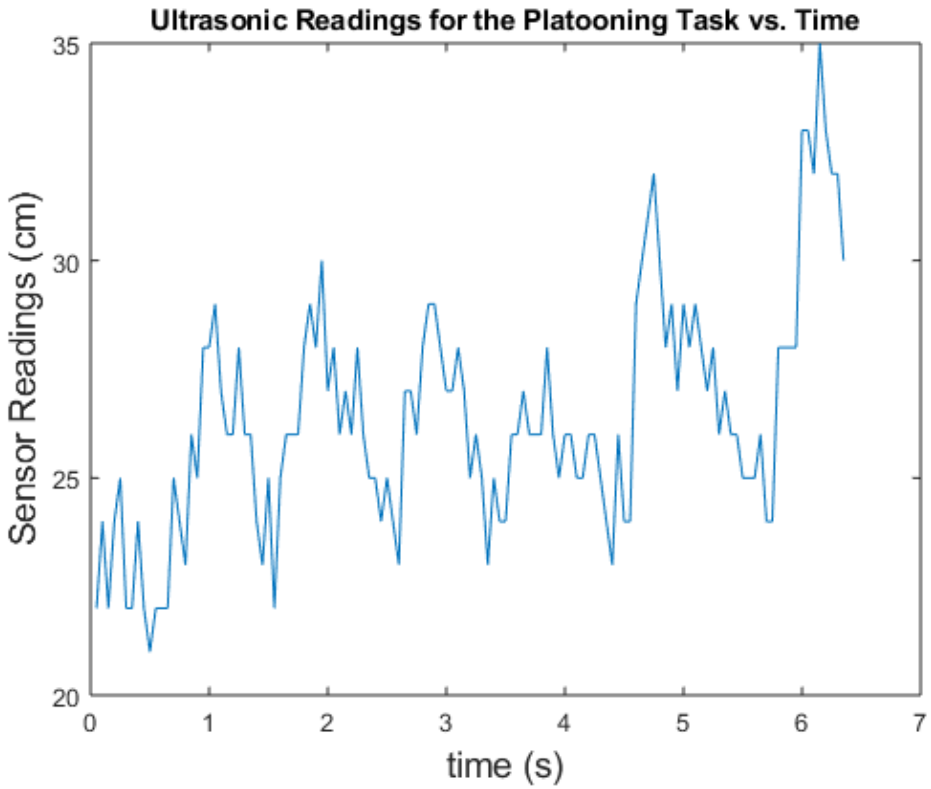
Task 2

We found that the Unmanned Vehicle stopped 2cm before the block while tracking the path laid.

Task 3

For this task, we chose $K_P = 0.1$
We found that the Unmanned Vehicle stopped on the red line at the end of the track.

Plot:



Figure(9): Plot of Task 3

Discussions

From the plot, we can see that the Unmanned Vehicle stayed within 20-25cm range from the moving vehicle for most of the time. The UV goes beyond 25cm only when the moving vehicle speeds up as the UV adjusts to the rising speed.

Code:

Task 1 and 2: Line Tracking and Collision Avoidance

```
clear
clc
myev3=legoev3('usb');
set_speed=30;
mdiff=5;
mymotor1=motor(myev3,'B');
mymotor2=motor(myev3,'C');
mymotor1.Speed=set_speed+mdiff;
mymotor2.Speed=set_speed;
kp=1.8;
ki=0.08;
kd= 5;%8;
prev_error=0;
start(mymotor1);
start(mymotor2);
i=1;
fid = fopen('ECE556-group13_task1_trial1.txt','w');
start_t = clock;
Start_conv=start_t(6)+60*start_t(5)+60*60*start_t(4);
k=1;
j=1;
while (true)
    ls_read=readInputDeviceREADY_RAW(myev3,2,0,1);%2988 3424
    us_read=readInputDeviceREADY_RAW(myev3,4,0,1);
    us = us_read/10;
    read_values_normal=(ls_read-2950)/5;
    error=(read_values_normal-50);
    error_list(i)=error;
    sensor_readings(k)=read_values_normal;
    sum_e=sum(error_list);
    diff=error-prev_error;
    if(us<8)
        mymotor2.Speed=0;
        mymotor1.Speed=0;
        break
    end
    if(error>0)

         mymotor2.Speed=set_speed;
        mymotor1.Speed=set_speed+kp*(error)+kd*(diff)+ki*(sum_e)+mdiff;
```

```
    else
        mymotor2.Speed=set_speed-kp*(error)-kd*(diff)-ki*(sum_e);
        mymotor1.Speed=set_speed+mdiff;
    end
    Time_st = clock;
    Time_conv=Time_st(6)+60*Time_st(5)+60*60*Time_st(4);
    T_stamp(k)=Time_conv-Start_conv;
    prev_error=error;
    if(i==5)
        i=0;
    end
    i=i+1;
    k=k+1;
end
tq_t1=0.05:0.05:0.05*length(T_stamp)+0.05;
T_stamp(length(T_stamp)+1)=T_stamp(length(T_stamp))+0.05;
vq=zoh(T_stamp',sensor_readings',tq_t1',false);
for l=1:length(vq)
    fprintf(fid,'@ %f @ %f\r\n',sensor_readings(l),tq_t1(l));
end

fclose(fid);
```

Task 3: Platooning

```
clear
myev3=legoev3('usb');
mymotor1=motor(myev3,'B');
mymotor2=motor(myev3,'C');
mdiff=3;
mymotor1.Speed=mdiff;
mymotor2.Speed=0;
start(mymotor1);
start(mymotor2);

j=1;
kp=0.1;
fid = fopen('ECE556-group13_task3_trial1.txt','w');
start_t = clock;
Start_conv=start_t(6)+60*start_t(5)+60*60*start_t(4);
us_prev=0;
while (true)
mdiff=3;
us_read=readInputDeviceREADY_RAW(myev3,4,0,1);
us = us_read/10;
us_array(j)=us;
us_diff = us-us_prev;
if us-21>5
    K_sep=13;
    ctrl=5;
    K_TA=0.3;
elseif us-21<5
    K_sep=0;
    ctrl=0;
```

```
    K_TA=8;
    mdiff = 0;

else
    K_sep = 10;
    ctrl=us-20;
    K_TA = 1;
end
Speed_ctrl = (K_sep*ctrl) + (K_TA* us_diff);
ls_read=readInputDeviceREADY_RAW(myev3,2,0,1);%2988 3424
read_values_normal=(ls_read-2950)/5;
error=(read_values_normal-50);
ls_read_l=readInputDeviceREADY_RAW(myev3,1,0,1)%2988 3424
if ls_read_l>3300
    mymotor1.Speed=0;
    mymotor2.Speed=0;
    break
end
if(error>0)

     mymotor1.Speed=Speed_ctrl+mdiff;
    mymotor2.Speed=Speed_ctrl+kp*ctrl*(error);
else
    mymotor1.Speed=Speed_ctrl-kp*ctrl*(error)+mdiff;
    mymotor2.Speed=Speed_ctrl;
end
us_prev=us;
Time_st = clock;
Time_conv=Time_st(6)+60*Time_st(5)+60*60*Time_st(4);
T_stamp(j)=Time_conv-Start_conv;
j=j+1;
end

tq_t3=0.05:0.05:0.05*length(T_stamp)+0.05;
T_stamp(length(T_stamp)+1)=T_stamp(length(T_stamp))+0.05;
vq=zoh(T_stamp',us_array',tq_t3',false);
for l=1:length(vq)
        fprintf(fid,'@ %f @ %f\r\n',vq(l),tq_t3(l));
end

fclose(fid);
```

References

[1] K. A. Unyelioglu, C. Hatipoglu, and U. Ozguner, “Design and Analysis of a Line Following Controller”, IEEE Trans. Contr. Syst. Technol., vol. 5, no. 1, pp. 127-134, 1997.

[2] Balaji, Vikram , and M.Chandrasekaran. “Optimization of PID Control for High Speed Line Tracking Robots.” 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS 2015) n. pag. Print.