North Carolina State University

PROJECT REPORT

Intelligent Space

Authors:
Davis HARRISON
David JUAN
Binoy THOMAS
Juan TORRECILLA

Supervisors:
Dr. Mo-Yuen CHOW
Zheyuan CHENG

Advanced Diagnosis, Automation, and Controls (ADAC) Laboratory
Department of Electrical and Computer Engineering

March 29, 2019

Contents

Introduction

The objective of this project was to develop MATLAB/Simulink code to control four Lego Mindstorms EV3s. The EV3s were designed and built for maximum efficiency with ease of building to ensure consistent, quality performances. After the EV3s were built, Simulink code was developed after numerous iterations to control, manipulate, and monitor the EV3s. A Graphical User Interface (GUI) was written and coded with MATLAB's AppDesigner tool to allow users to wirelessly change what the Simulink program is doing via WIFI communication. OptiTrack was utilized in the Simulink and GUI programs to allow for EV3 location monitoring, parking, and finding the shortest path. Multiple forms of analyses were conducted on the project to given reasoning behind decisions as well as open discussions for future improvements to the systems used. This report contains all the information necessary for future students and/or hobbyists to recreate the project as well as use analytical insight and discussions to improve upon the current setup.

Design of Unmanned Vehicle

The UV, shown in Figure 1, was designed for maximum maneuverability and optimum sensor output. The Lego EV3 is the brain of the UV and all the sensors and actuators are attached to its ports. The Lego building kit was used to build the entire structure of the Unmanned vehicle. Figure 1 below shows the completed EV3 robot design.

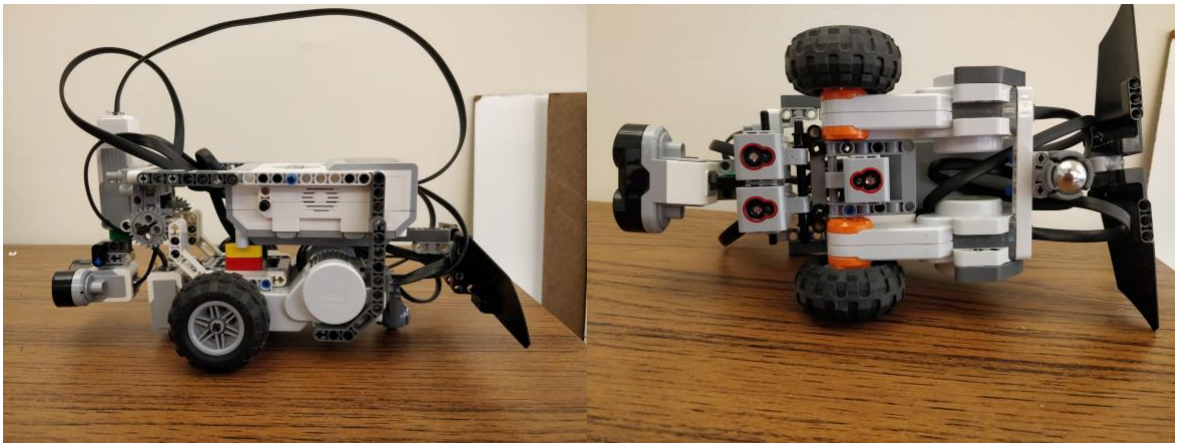


Figure 1. Unmanned Vehicle Design

Two color sensors were connected in a line, perpendicular to the forward direction of the vehicle, facing the ground. They were placed with a little gap in between such that both are perfectly aligned with the edges of the line. One sensor was placed a little behind to detect the red and blue tape. This sensor was set to the Color mode whereas the other two sensors were set to Reflected light intensity mode. They were placed 1 cm off the ground for optimum sensor output. The ultrasonic sensor was placed at the front of the vehicle, facing forward, so that it could detect objects ahead. Two large motors with wheels attached were placed on either side of the UV near the front. A small caster wheel was placed in the center of the rear to allow the

UV to turn freely. This wheel configuration placed the UV's axis of rotation to be at the front of the vehicle, near the color sensors. Since the axis of rotation is near the color sensors, any change in direction would be instantly picked up by the color sensors. This allowed for tight control of the UV. The medium motor was attached to the front of the UV with the ultrasonic sensor attached to it. The medium motor is used such that the ultrasonic sensor can face the direction of movement of the unmanned vehicle. A fin was attached at the back of the UV so that the ultrasonic sensor can detect the distance to the leader properly.
Factors taken into consideration while designing the UV

1. Alignment of the light sensors
   The 2 light sensors were perfectly aligned such that they detect the edges of the line. This ensures that the UV does not oscillate to a great extent and the path is detected faithfully.
2. Position of light sensors and motors
   The light sensors must be close to the axis of rotation of the motors and the motors should be as close to each other as possible. If the motors are close to each other, then the turning radius if the UV is small and a slight change in the sensor readings will only cause a small change in the motor velocity and the motors will not give a jerky movement. This, in turn, will not cause drastic changes in consecutive sensor readings and we will obtain smooth path tracking.
3. Position of the ultrasonic sensor
   The ultrasonic sensor must not be too low to the ground such that it detects imperfections in the ground and it should be in perfect alignment with the back fins attached at the back of the UV
4. The UV should be as compact as possible
   We tried to build the UV to be as compact as possible. The dimensions of the UV were only a summation of the dimensions on the components like the brick, motors and sensors. We tried to leave the least space between these components. The wires were also arranged tightly.

Program Design

Much of the Simulink program was updated and rewritten from the Phase One product. Two new blocks were introduced: Parking and Send Data EV3. The former will be discussed in a later section and the latter will be used and discussed in Phase Three but was added in this phase. The program blocks and tags were color coded and renamed for ease of program understanding. Program comments were also added in some areas. The program blocks have the colors: blue, purple, orange, gray, and red. The tags have the colors: white, yellow, and green. The color blue indicates input blocks. Purple indicates EV3 function blocks such as Path Tracking. Orange blocks are output blocks such as displaying information on the EV3. The grey block indicates the motor block. The red block is extra credit or extraneous 'for fun' blocks. White tags carry commands such as mode of the EV3. Yellow tags carry sensor data and green tags carry motor data. Figure's 2 and 3 below show the overall schematic of the program in high level flowchart form and Simulink form respectively. For the flowchart, the blue input blocks have parentheses that contain all the outputs for that block. The remaining block parentheses

contain all the inputs used for that block. In addition, a MATLAB script was written to make selecting which EV3 had code being uploaded to it and insertion of an IP address easier. The MATLAB script should be run before each EV3 code upload to ensure the correct UDP ports are being used as well as the correct IP address is being used with the GUI. Upon the first time running the code, the program will ask the user to input their iSpace IP address. Each run will ask the user which EV3 they are using then update the correct port values. This code can be found in Appendix C.

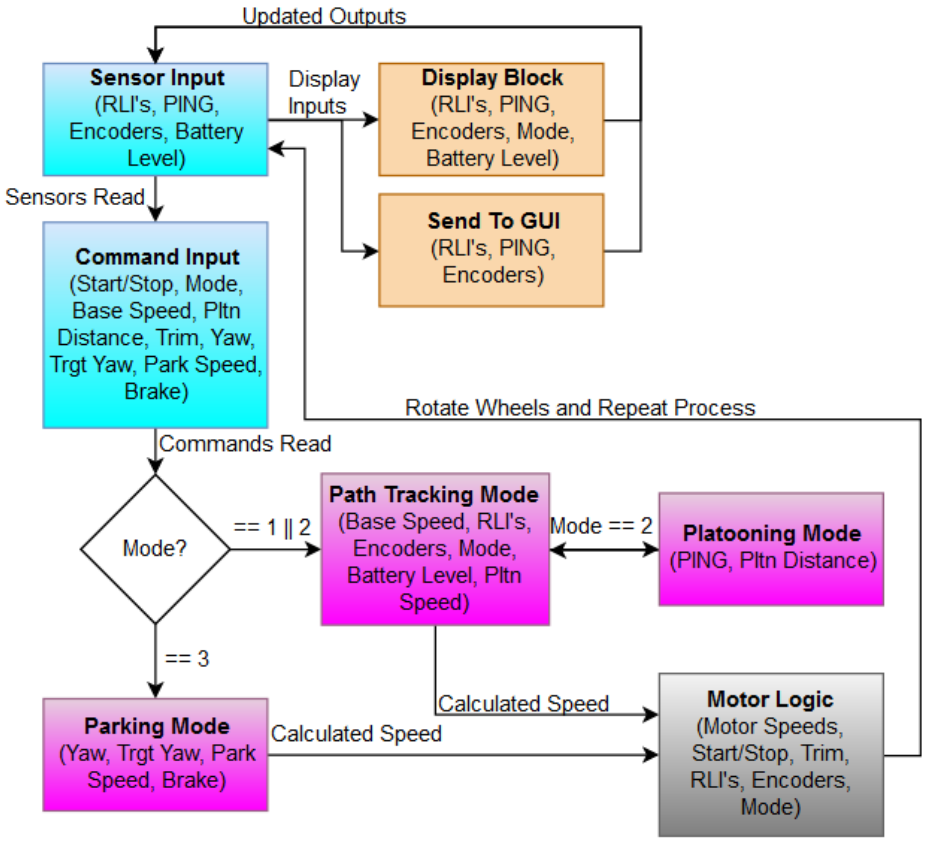


Figure 2. Simulink Program High Level Flowchart

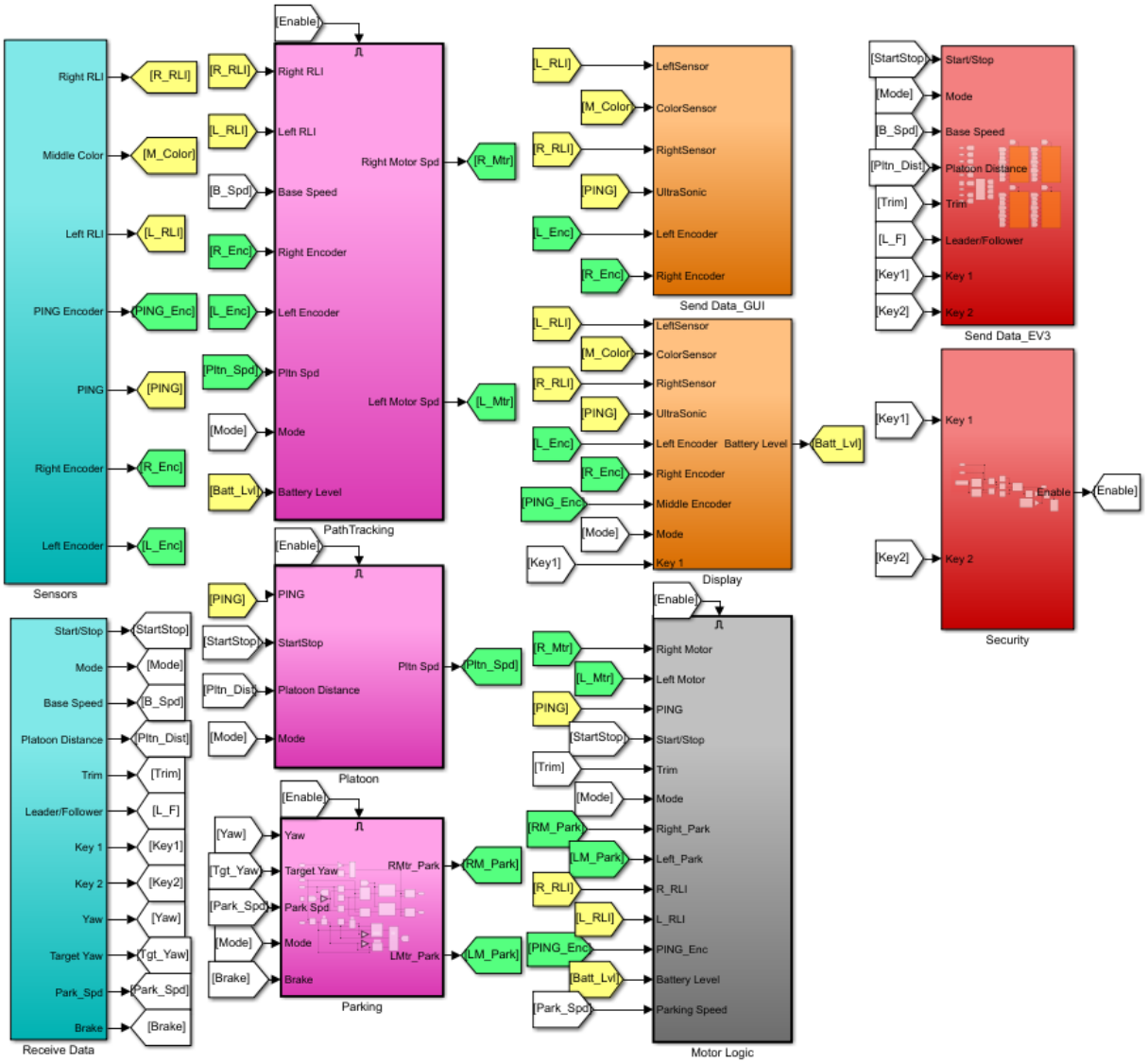


Figure 3. Overall Program Schematic

Sensors Block

The sensors block is responsible for all the EV3 internal inputs. These inputs include: right reflected light intensity (RLI) sensor, middle color sensor, left RLI sensor, ultrasonic (PING) servo encoder, PING sensor, right motor encoder, and left motor encoder. These inputs provide data to all the actuators and data processing units of the program. These sensor blocks are then type casted to double to allow the other program blocks to use the data. The right and left RLI sensors are used to detect the path of the track. The middle color sensor is currently unused but is available for use for necessary applications. The PING servo encoder is used for platooning to know the position of the PING sensor. The PING sensor is used during platooning to know the distance from the vehicle that is being followed in centimeters. The right and left encoders are used with path tracking and platooning in order to estimate the EV's current speed. Figure 4 below shows the Sensors block schematic.

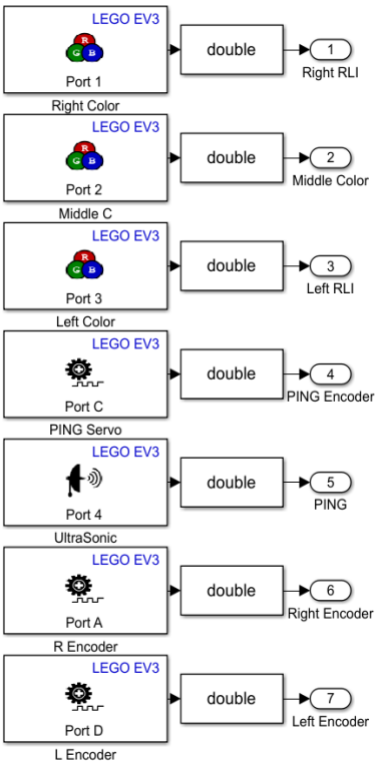


Figure 4. Sensor Block Schematic

Receive Data Block

The Receive Data block is the block that receives commands from the user via the Graphical User Interface (GUI). The commands that are received are: Start/Stop, Mode, Base Speed, Platoon Distance, Trim, Leader/Follower, Key 1, and Key 2. Start/Stop commands the EV3 to start or stop moving. The Mode command turns sets the mode to either Path Tracking, Platooning, or Parking. The Base Speed command sets the base speed of the EV3 in the range of 0 to 100 percent of the max speed of the motors. The Platoon Distance command sets the platooning distance in centimeters from 0 to 100. The second UDP receive block is only used when in the parking mode. The team felt more reliability would come with using a quickly sampled port separate from the other commands. This yields faster updates when parking and smaller packet sizes. Faster command updating when parking is crucial to ensure the EV3 remains controllable throughout the various stages of parking and de-parking. The commands for parking are sent and updated 100 times per second. The four commands sent on the parking UDP receive block are Yaw, Target Yaw, Parking Speed, and Brake. The EV3 is sent its current yaw and target yaw in order to correct its orientation when moving. The parking speed command determines the speed at which the EV3 travels when parking and de-parking. The brake command is used to stop the EV3 when necessary. An example of this is to avoid collision with other parking EV3's. Figure 5 below shows the updated Receive Data Block.

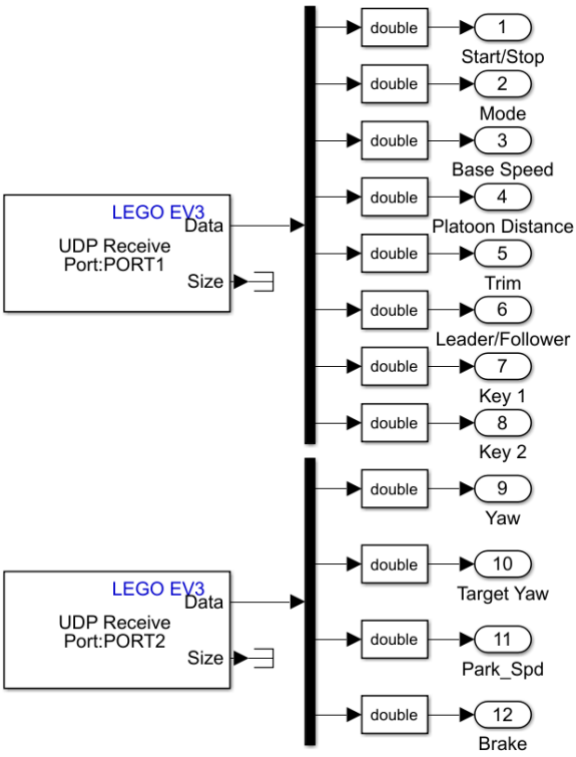


Figure 5. Receive Data Block Schematic

Path Tracking Block

The Path Tracking block is composed of four sub-blocks and the Fuzzy logic. These sub-blocks are PID Bias, Speed Bias, Real Time Speed Estimation, and Battery Bias. They allow for greater control during path tracking as well as create a more robust system. Juan and I determined through brain storming combined with testing that these blocks are essential to having a high-quality Path Tracking block. The sub-blocks will be explained in greater detail in subsections below. Figure 6 below is a high-level flowchart of the Path Tracking block. Equation 1 below is the mathematical model for the overall block. The sum of the Fuzzy controllers is plus/minus because the summed value is added to one wheel and subtracted from the other. Figure 7 and 8 below illustrates the motor speeds and the Path Tracking block output gains under a set of conditions. The purpose of the graph is to show what the block is doing. Many other plots with different sets of conditions can be found in Appendix D. Only a left curved is illustrated because a right curve would just have the opposite motor values. The entire Path Tracking block was simulated in MATLAB under real track conditions in order to collect the data shown. Figure 9 below shows the updated Path Tracking block.

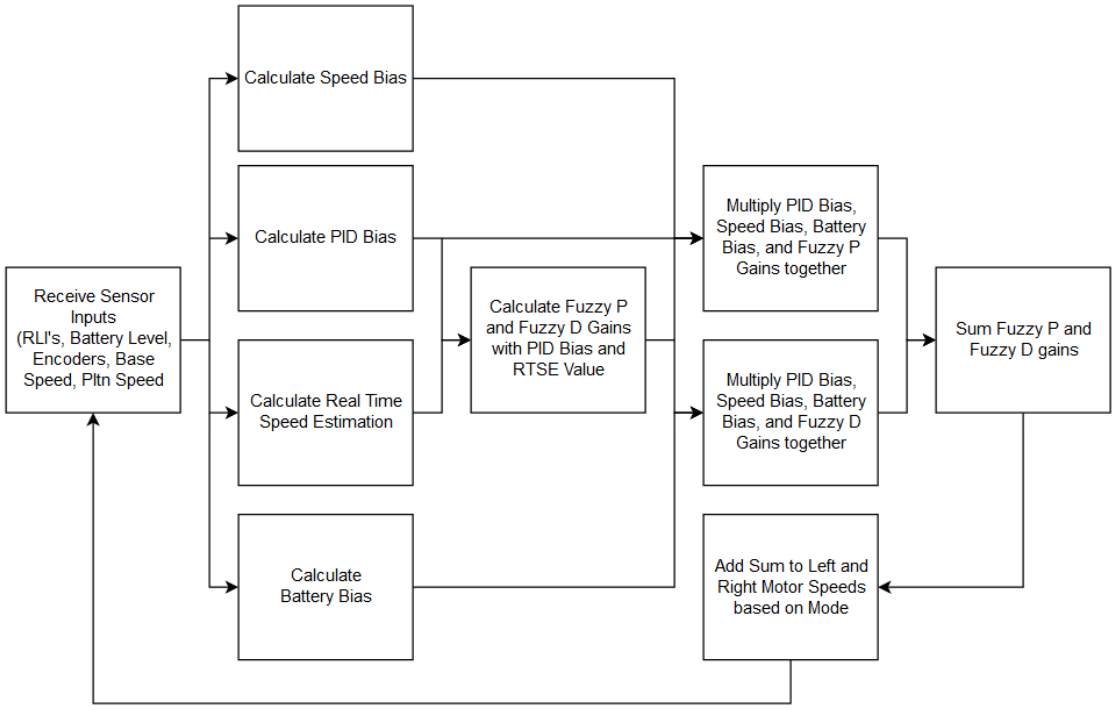


Figure 6. Path Tracking Block High Level Flowchart

$$Speed_{PathTrack} = Speed_{Base} \pm \sum_{Fuzzy=P}^{D} \left[\prod \left[(Bias_{PID})(Bias_{Battery})(Gain_{Fuzzy})(Bias_{Speed})\right]\right] \quad (1)$$

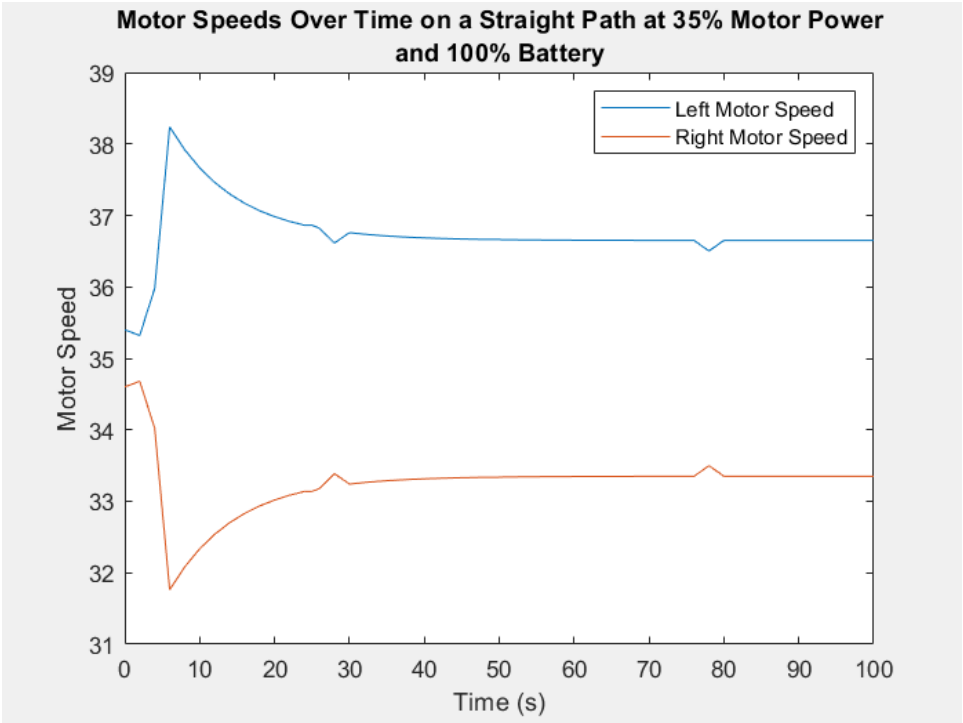


Figure 7. Motor Speeds Graph 1

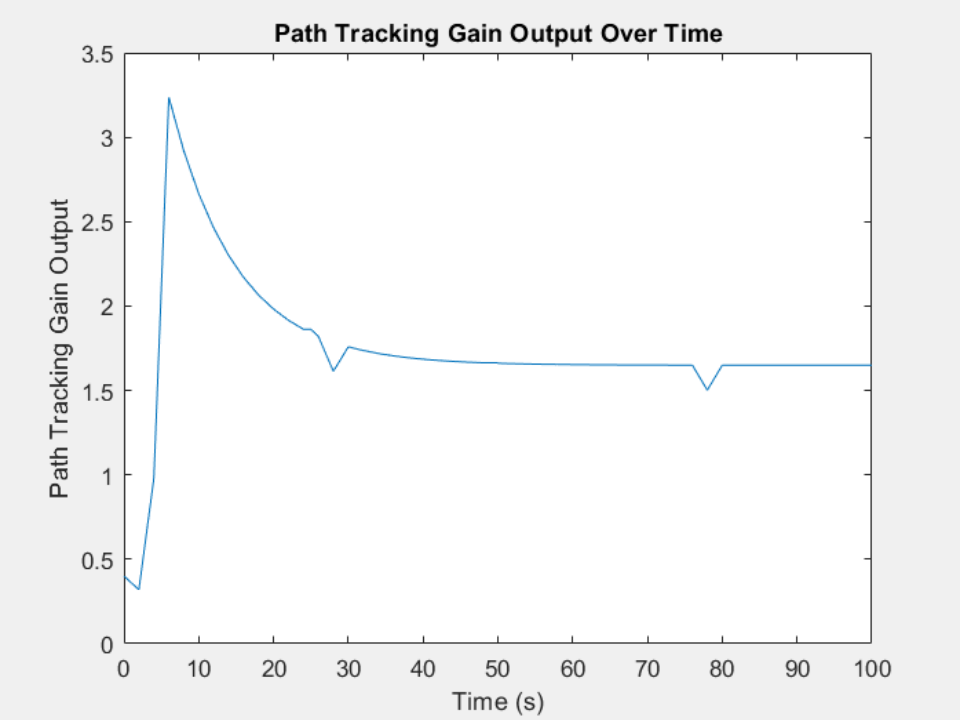


Figure 8. Path Tracking Gain Output 1

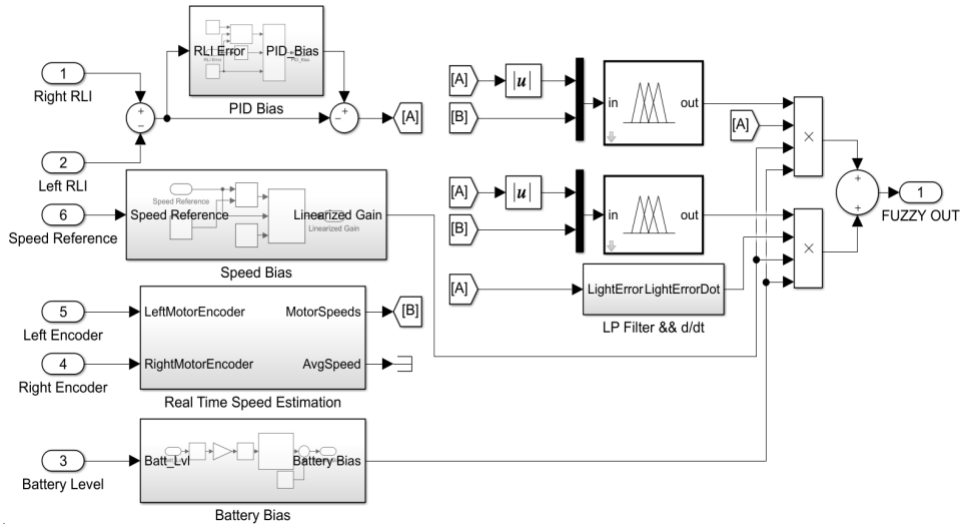


Figure 9. Path Tracking Block Schematic

PID Bias (Juan)

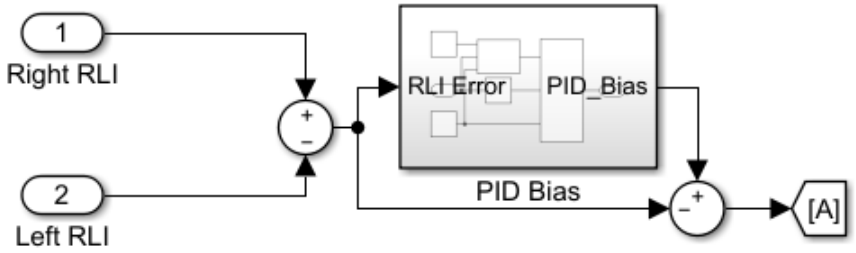


Figure 10. Error computation

In order to be controlling our system all the time, the error is increased or reduced (RLI Error) by a small number. This number is the output of the PID_Bias block in Figure 10.

Mathematical equations for Figure 10:

RLI: Reflected Light Intensity (Raw values obtained from light sensor).
R./L. RLI: Reflected Light Intensity for Right / Left sensor.

$$RLI\ E. = R.RLI - L.RLI$$
$$A = -(RLI\ E.) + PID\_Bias$$
$$R.RLI\ \epsilon\ [3\ ,94]$$
$$L.RLI\ \epsilon\ [3\ ,94]$$
$$RLI\ E.\epsilon\ [-91\ ,91]$$

A is the error that will be the input in the Fuzzy logic. Maximum and minimum value of light sensors depend on distance to the floor. From experience, usually maximum ≈ 94 and minimum ≈ 3.

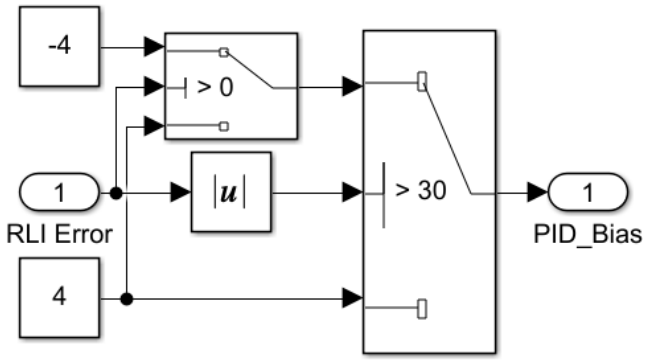


Figure 11. PID_Bias

Displaying the |RLI Error| on the EV3 screen we realized that when it was in a straight line it never went bigger than ≈ 30.

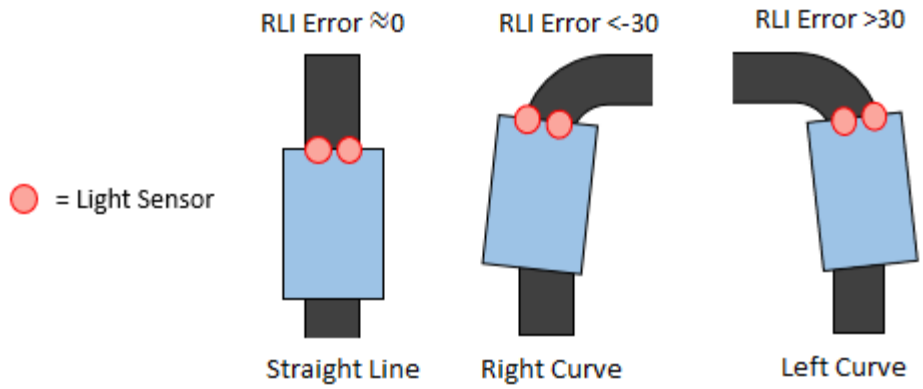


Figure 12. Relation between Light error and robot movement

In the PID_Bias figure, the switch on the right is used to do what mentioned above in case the error is smaller than 30 (when the robot is on a straight line), but when the error is bigger than 30 (curve case) we should change this small number to a negative or positive value depending on the turn the EV3 is doing. If we do not use the left switch the robot will turn more to one side than to the other.

Mathematical equations for Figure 11:

$If\ |RLI\ E.| > 30$ *Curve case

$\{$

$If\ RLI\ E. > 0$ *Left curve

$\{PID\_Bias = -4\}$

$Else$ *Right curve

$\{PID\_Bias = +4\}$

$\}$

$If\ |RLI\ E.| \leq 30$ *Straight Line

$\{PID\_Bias = +4\}$

$RLI\ E. \epsilon\ [-91\ , 91]$

$PID\_Bias\ \epsilon\ [-4\ , 4]$

$A\ \epsilon\ [-95\ , 95]$

*If we did not differentiate between curve or straight line: PID_Bias = 4 and our A ϵ [-87, 95] having more error when the robot turns to one side than to the other.

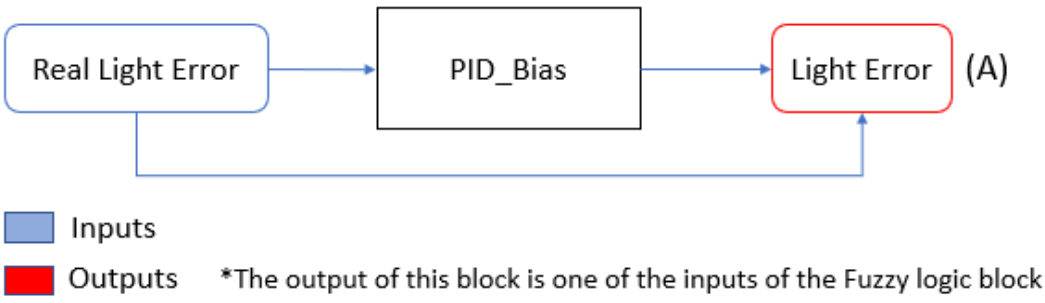


Figure 13. PID_Bias Block Diagram

Speed Bias

The Speed Bias sub-block linearizes the Fuzzy output gains in relation to the speed when the speed is higher than 55 percent. The maximum output gain of the fuzzy is for 100 percent speed. We needed to linearize the Fuzzy output in relation to the speed because we found it to be difficult to configure the Fuzzy logic with when the speeds were between 50 and 100 percent. If we adjusted the Fuzzy to work for 75 percent, it would stop working for 100 percent and vice versa. Through our testing and manipulation of the Fuzzy, we noticed that there is a linear relationship between the base speed and gains. Based on our findings we implemented this Speed Bias block in order to only require us to configure the Fuzzy logic for 100 percent speed. After implementation we saw that the EV3 handled the track very similarly at all speeds. Equation 2 below models the output of the block at speeds greater than 55 percent. Speeds less than 55 percent result in a block output of 1. Figure 14 below shows the output of the Speed Bias block with speeds ranging from 0 to 100 percent. Figure 15 below shows the Speed Bias block.

$$Bias_{Speed} = \frac{Speed_{Base}}{100} \quad (2)$$

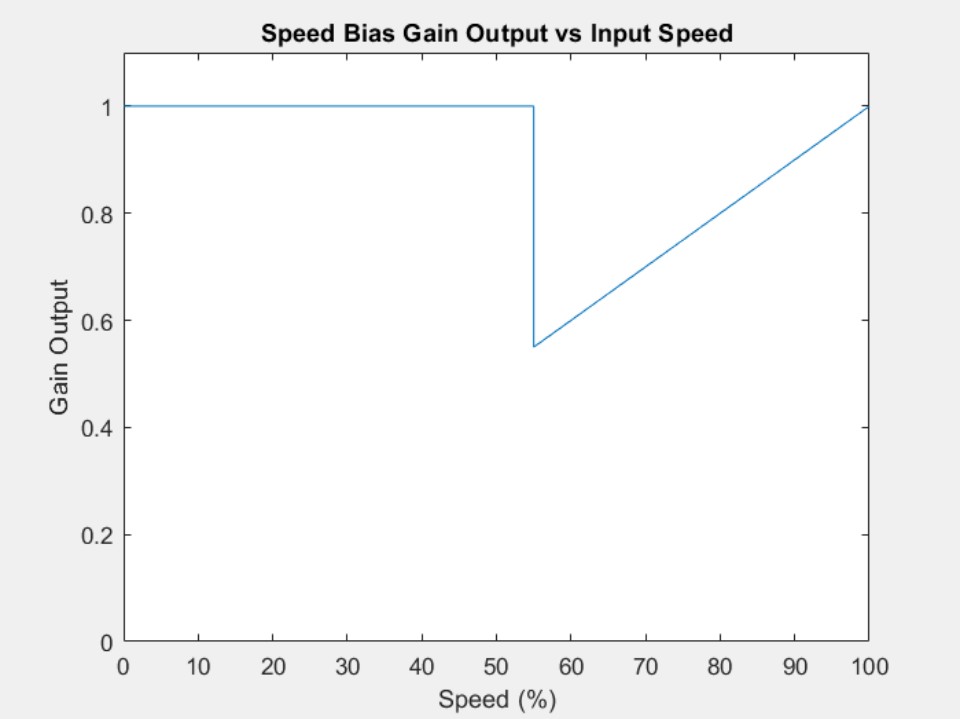


Figure 14. Speed Bias Gain Output

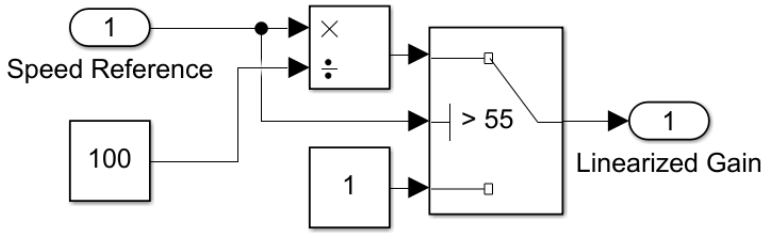


Figure 15. Speed Bias Block Schematic

Real Time Speed Estimation

The Real Time Speed Estimation sub-block uses the derivative of the encoders to determine the velocity of the motors. The encoder values are the positions of the wheels in relation to time. Taking the derivative of the positions yields the velocities of the wheels. First, we take a 10-window moving average of the encoder positions. Next, we take the derivative of the moving averages and take the highest value of the two wheels to determine the motor speed. The maximum speed of the two wheels is used because one wheel rotates faster in turns, so we want that speed to be used as an input to the Fuzzy controller in order to have a lower error in turns. Equation 3 below is the equation for determining the maximum speed of the wheels. Figure 16 below shows the output of the block. The encoder values change by 10 radians every second in this example data set. As a result, it can be seen in the figure that the estimated speed calculated from the block is 35 percent speed. Figure 17 below shows the Real Time Speed Estimation block.

$$Speed_{RTSEMax} = max\left[\frac{d}{dt}\left(\frac{\sum_{i=1}^{10} \theta_{rightencoder}}{10}\right), \frac{d}{dt}\left(\frac{\sum_{i=1}^{10} \theta_{leftencoder}}{10}\right)\right] \quad (3)$$

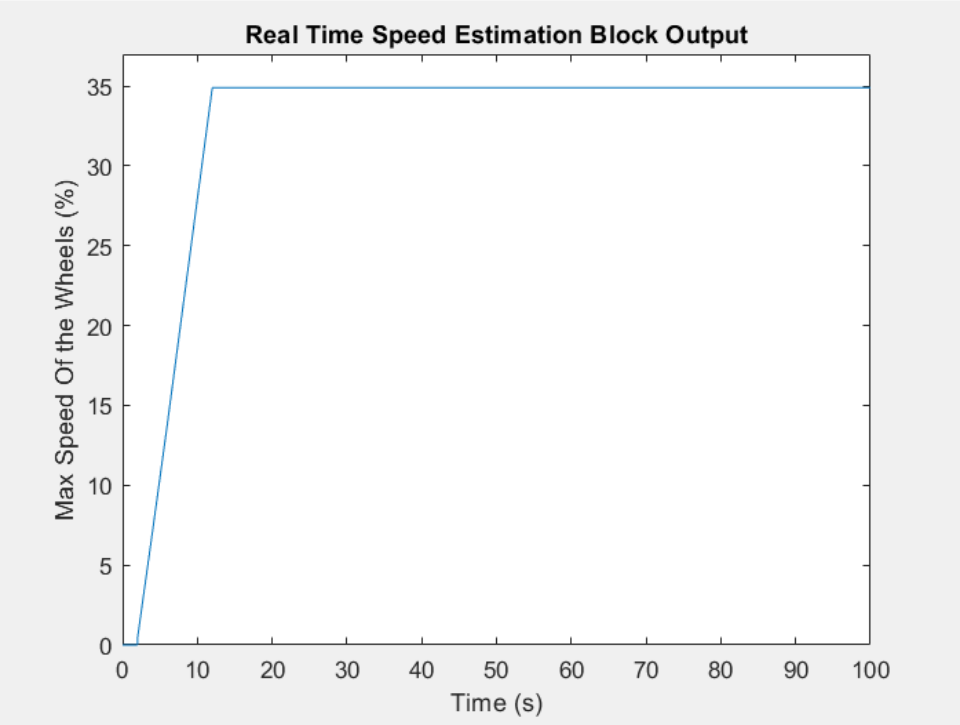


Figure 16. Real Time Speed Estimation Block Output

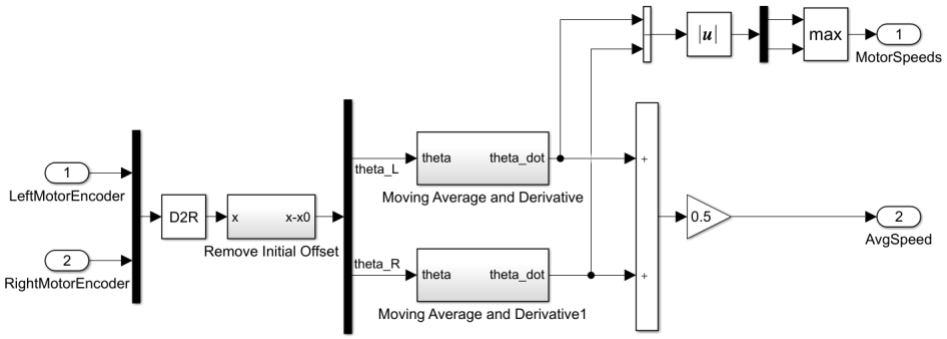


Figure 17. Real Time Speed Estimation Block Schematic

Since the block also outputs the average speed, I decided to perform some analysis on the C1 and C2 values using the max speed and average speed. I ran ten trials using max speed as the Fuzzy input and ten trials with average speed as the Fuzzy input. The data was collected with an EV3 running at 75% power clockwise around the outer track at 100% battery. Table 1 below shows the recorded data.

Table 1. Real Time Speed Estimation C1/C2 Trials

RTSE Comparison

MAX SPEED		AVG SPEED	
C1	C2	C1	C2
502	774	414	625
382	719	455	777
408	746	410	740
461	810	428	733
440	782	418	741
459	850	656	966
500	851	388	641
435	743	381	625
441	799	395	627
384	663	446	719

With this data, I decided to use ANOVA (Analysis of Variance) to analyze the data. ANOVA analyzes the differences among group means in a sample. With ANOVA, if the calculated F-Value is above the $F_{critical}$-Value then we will reject the null hypothesis. The null hypothesis of this data is that the means of the two outputs are the same. This means that there is no difference between the two output's effects on the C1 and C2 values. The ANOVA analysis was performed using Excel. Table 2 below is the ANOVA result with the C1 value.

Table 2. C1 ANOVA Summary

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
MAX C1	10	4412	441.2	1753.511111		
AVG C1	10	4391	439.1	6369.211111		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	22.05	1	22.05	0.005429214	0.942075117	4.413873419
Within Groups	73104.5	18	4061.361111			
Total	73126.55	19				

The F-Value is less than the $F_{critical}$-Value. This means we fail to reject the null hypothesis resulting in no statistically significant difference on the C1 value between using the max speed and the average speed. Table 3 below is the ANOVA result with the C2 value.

Table 3. C2 ANOVA Summary

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
MAX C2	10	7737	773.7	3420.011111		
AVG C2	10	7194	719.4	10856.93333		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	14742.45	1	14742.45	2.06521081	0.167850076	4.413873419
Within Groups	128492.5	18	7138.472222			
Total	143235	19				

The F-Value is less than the $F_{critical}$-Value. This means we fail to reject the null hypothesis resulting in no statistically significant difference on the C2 value between using the max speed and the average speed. Despite the ANOVA tests yielding no statistical difference, I felt that one output had to have at least a small advantage over the other. Table 4 below shows the averages and standard deviations of each output for both the C1 and C2 values.

Table 4. Averages and Standard Deviations of C1/C2 Values

	MAX SPEED		**AVG SPEED**	
	C1	C2	C1	C2
AVERAGE	441.2	773.7	439.1	719.4
STD DEV	41.8749461	58.48086107	79.80733745	104.1966009

The justification of using the max speed as the final output over average speed is that of the ten trials, the standard deviation was lower than that of the average speed for both C1 and C2 values. This yields more consistent C1 and C2 values.

Battery Bias

The Battery Bias sub-block uses the linearized battery level to adjust the gains when the battery starts to deplete. The dynamics of the battery became a real interest to me. As the battery died, I noticed the path tracking capability degraded severely. I wanted to build a Simulink block to compensate for the battery dynamics such that the EV3 runs smoothly with low errors consistently regardless of battery levels. In order to build the block, I conducted a battery level test. I ran the EV3 around the track at 100 percent motor speed and recorded the battery levels at 5-minute intervals until it died. Table 5 below displays the data I collected from my test. The last column in the table is the output of the final Battery Bias block I developed. I will discuss my process for implementation below.

Table 5. Battery Level Test Results

Time (min)	Battery Level (%)	delta Batt Lvl	Block Output
0	100	0	1
5	100	0	1
10	100	0	1
15	100	0	1
20	100	0	1
25	100	0	1
30	100	0	1
35	100	0	1
40	100	0	1
45	100	0	1
50	100	0	1
55	100	0	1
60	100	0	1
65	100	0	1
70	100	0	1
75	90	10	1
80	75	15	1.002
85	63	12	1.004
90	53	10	1.008
95	43	10	1.016
100	36	7	1.032
105	26	10	1.064

110	17	9	1.128
115	10	7	1.128
120	9	1	1.256
125	2	7	1.256
130	0	2	1.256

Table Five's results are summarized in Figure 18 below. The figure's trendline is a 2-window moving average. Figure 19 below is the snapshot of the data when the battery starts to deplete after 70 minutes. Figure 19 contains a third order polynomial trendline for the data with an R-squared value to show how well the trendline fits the data model. Figure 20 below shows the change in battery levels at depletion. The figure contains a 2-window moving average trendline.

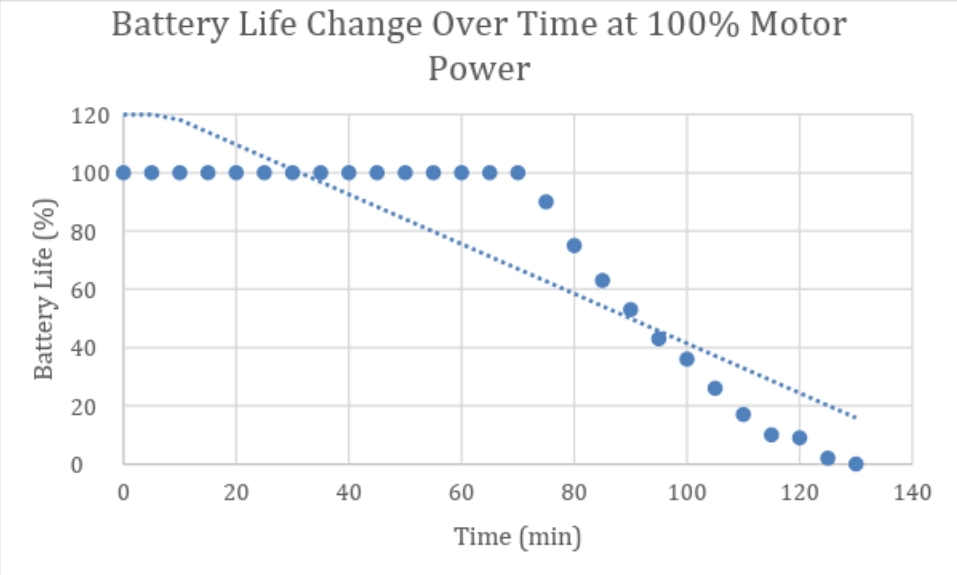


Figure 18. Battery Test Results Graph

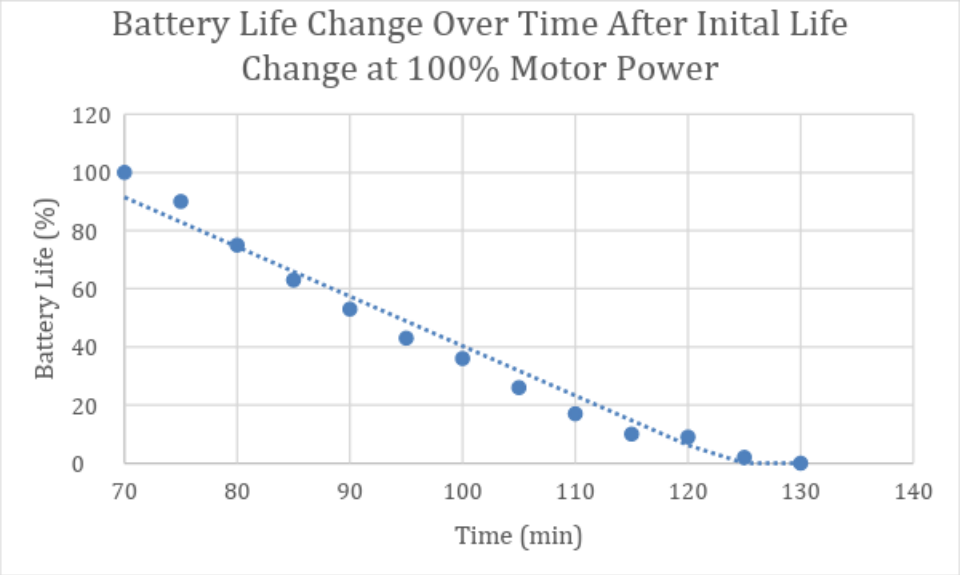


Figure 19. Battery Test Results Graph at Depletion

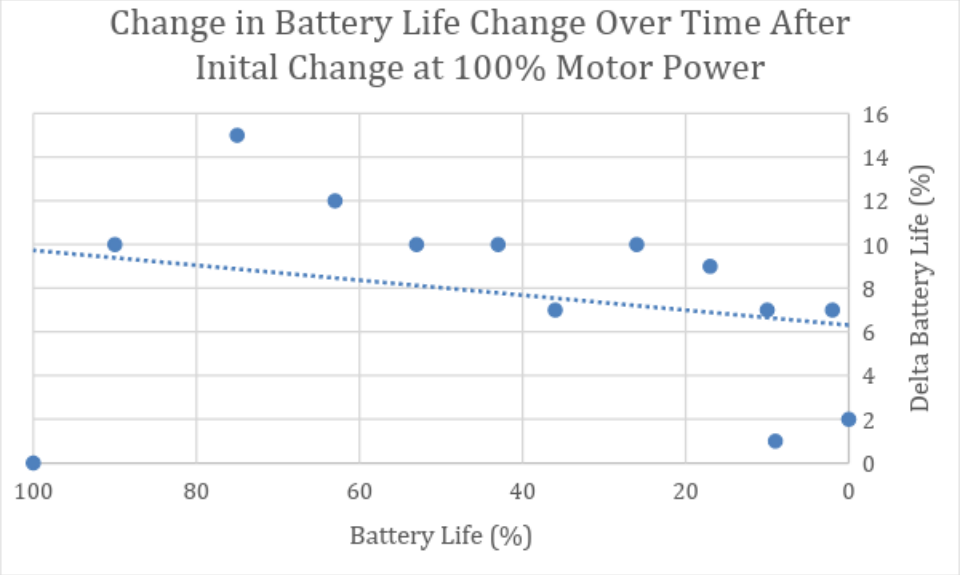


Figure 20. Delta Battery Levels Test Results at Depletion

The inverse equation of the trendline in Figure 19 was originally used to create the Battery Bias block. However, in actual implementation I found that the equation did not help much in keeping the error low when the battery started to deplete. Interestingly, the equation made the performance worse at battery levels greater than 90 percent as the block caused oscillations in the straight paths. My assumption was that in actuality, the gains needed to be much lower at a higher battery life and much higher at a lower battery life. Based on the difference in data versus the changing physical dynamics of the model, I decided to use a simple exponential equation to increase the gains more and more as the battery depleted. The simplest equation to use as a starting point was a 2^n power equation. Equation 4 below shows the equation I implemented for block. The equation results in a gain output of 1 at a battery level of between 90 and 100

percent and a gain output of 1.256 at a battery level between 0 and 9 percent. The equation is only applied when the battery level drops below 90 percent because above that threshold no additional gain is needed. The result is that the EV3 performs very well at all battery levels. Figure 21 below is the output of the Battery Bias block with battery values ranging from 100 percent to 0 percent from the test above with a 2-window moving average trendline. Figure 22 below is the block's output at all values from 100 percent to 0 percent. Figure 23 below is the Battery Bias block. A lookup table was used to implement the equation due to it requiring less processing power, ease of use, and only applying the equation when the battery dropped below the 90 percent threshold.

$$Gain_{BatteryBias} = 1 + \frac{2^{8-floor\left(\frac{BatteryLevel}{10}\right)}}{1000} \quad (4)$$

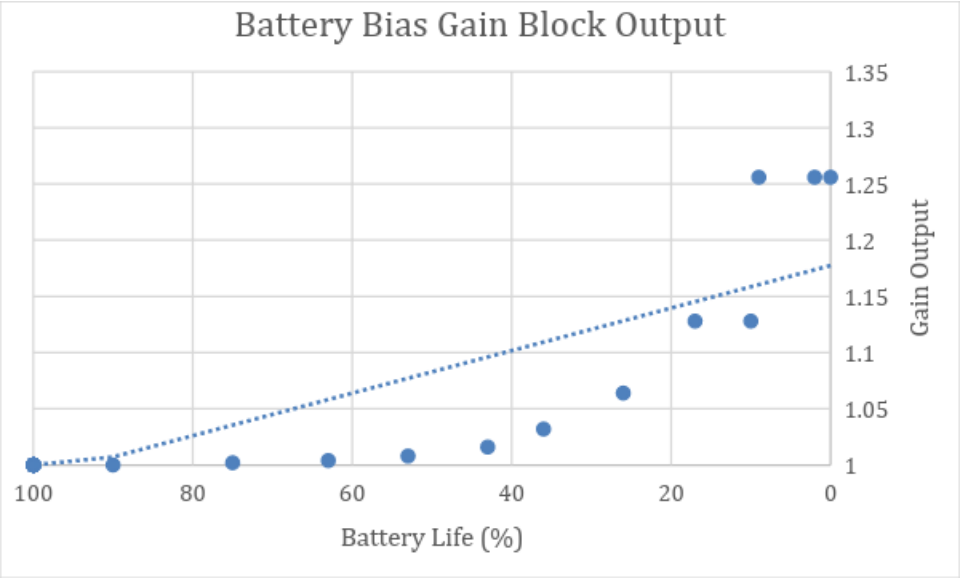


Figure 21. Battery Bias Block Test Output

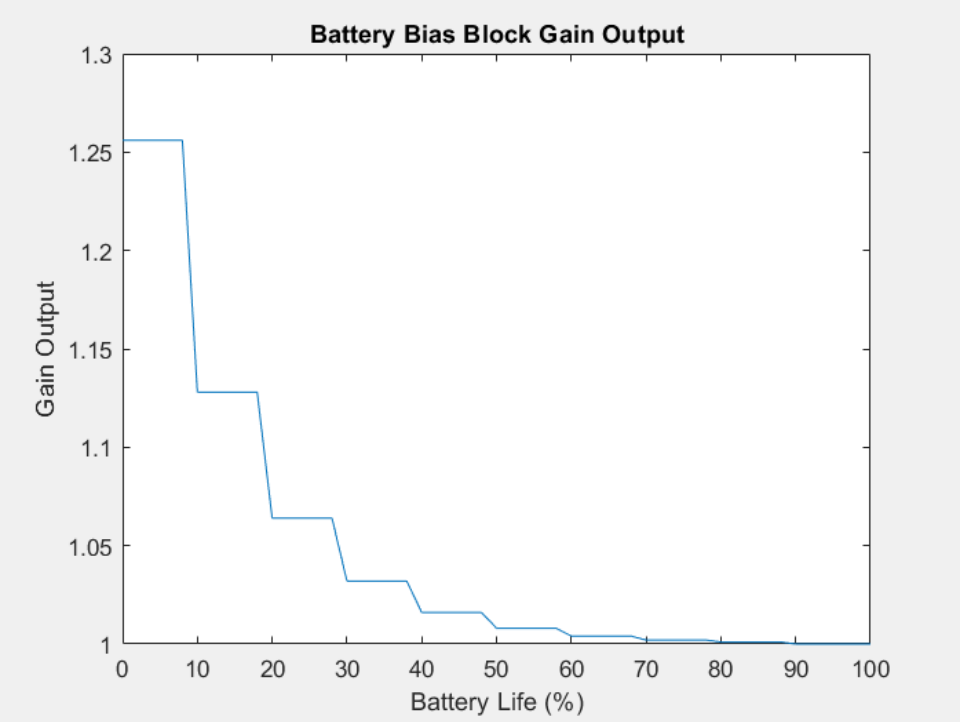


Figure 22. Battery Bias Block Output

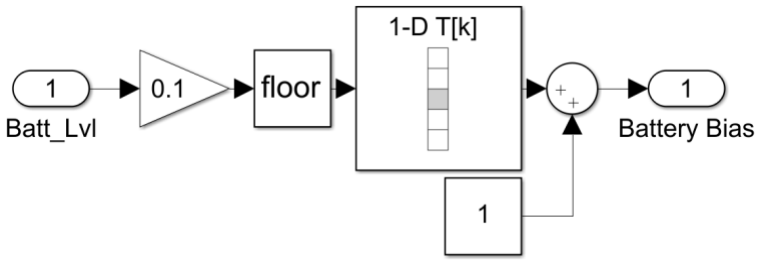


Figure 23. Battery Bias Block Schematic

Fuzzy Logic

The problem with using just a PID controller to do the path tracking is that the gains that make the EV3 run smoothly in the straight line are not the same ones that make the EV3 run smoothly in the curve. With fuzzy logic we can solve this problem by increasing gradually these gains when the error becomes higher and taking different factors into account.

As it can be seen from the following figure two inputs are given to the fuzzy block: the first one (A) is the error and the second one (B) is the real time speed of the car obtained through the encoders.

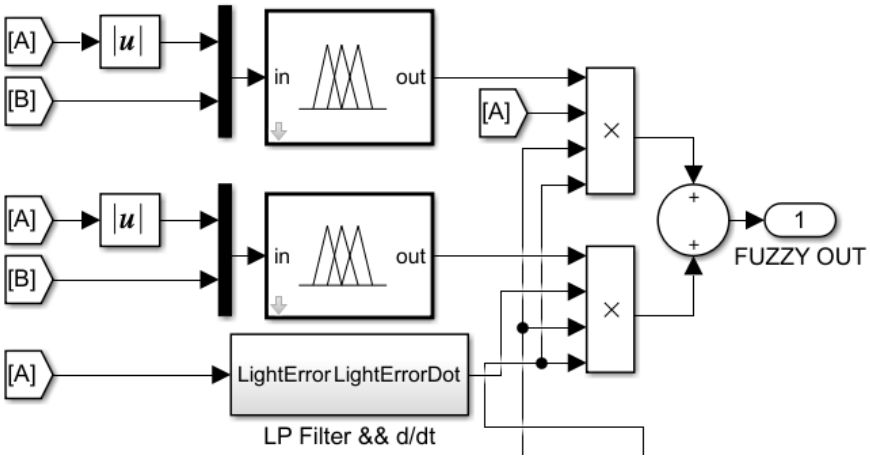


Figure 24. Fuzzy Logic

The output of each fuzzy block will be the proportional and derivative gains respectively of the PD controller used to track the line. These gains will be obtained considering the membership functions for the inputs and the rules stablished. The two membership functions (for each input) used are the following:

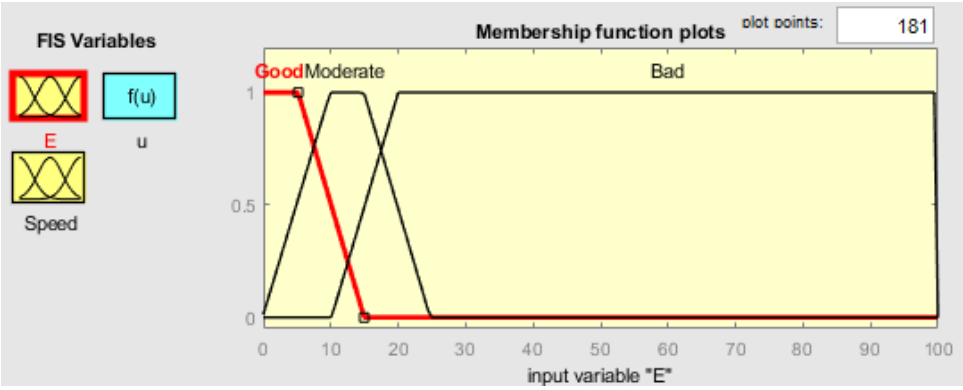


Figure 25. Membership function of error

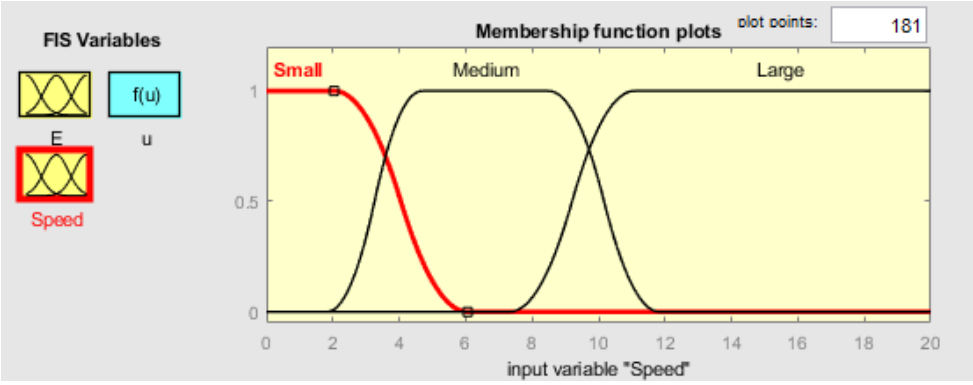


Figure 26. Membership function of speed

The error has been divided in three ranges and we are using the encoders information to have a closer estimation of the real time speed. Also, for both membership functions the variable curves

shape have a gradual transition between them. For example, the light error goes gradually from good to medium and from medium to bad thanks to the trapezoidal form.

1. If (E is Good) and (Speed is Small) then (u is Min) (1)
2. If (E is Good) and (Speed is Medium) then (u is Min) (1)
3. If (E is Good) and (Speed is Large) then (u is Mid) (1)
4. If (E is Moderate) and (Speed is Small) then (u is Min) (1)
5. If (E is Moderate) and (Speed is Medium) then (u is Mid) (1)
6. If (E is Moderate) and (Speed is Large) then (u is Max) (1)
7. If (E is Bad) and (Speed is Small) then (u is Mid) (1)
8. If (E is Bad) and (Speed is Medium) then (u is Mid) (1)
9. If (E is Bad) and (Speed is Large) then (u is Max) (1)

Figure 27. Fuzzy rules

In the figure, “u” means the proportional or derivative gain used between a minimum, medium and maximum gain that are chosen after trying with different numbers. Using the information of the encoders has really improved the performance of our EV3 robots making the system much more robust and being able to work at maximum speed. This is because we want the robot to gradually increase its speed to follow the line and we do not want high changes that can cause the system to oscillate too much. For example, in rule 3 if the error is good and the actual speed is large, we do not want the control to go suddenly to the minimum possible, we want first to go to a medium control and then when the speed is medium through rule 2 go to a lower speed. Before deciding to use the encoders, we tried to reduce the speed of the robot when the error increased what also gave a reliable performance but not as consistent as the one that we finally achieved using the encoders data. A Block Diagram is shown in the following figure to have a better understanding of how the path tracking logic works.

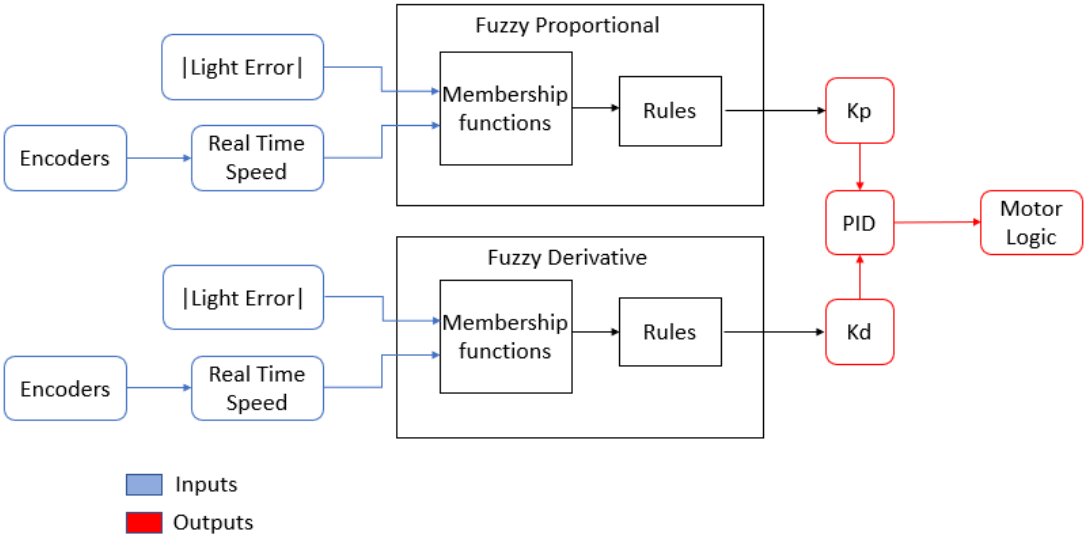


Figure 28. Fuzzy Logic Block Diagram

Platoon Block (Juan)

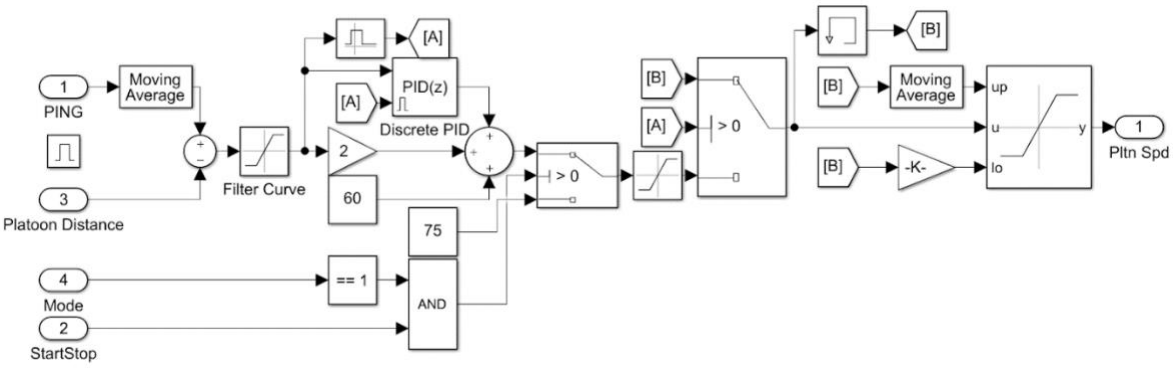


Figure 29. Platooning

Mathematical equations for Platooning:

PING: Raw values obtained from the ultrasonic sensor.
P.D. = Platoon Distance: Constant value we want to maintain with the leader robot.
n: Window length (Moving average parameter of Simulink=number of samples considered in the average)
i: last sample
i-1: previous sample to last sample
PING (After Moving Average) = PING (A.M.A.)
At the beginning when there is no data of previous PING values:

$$PING(i-n) = 0 \; for \; n \geq 1$$

$$PING\ (A.M.A.) = \frac{PING(i) + PING(i-1) + PING(i-2) + \cdots + PING(i-n)}{n}$$

$$D.E. = Distance\ Error = PING\ (A.M.A) - P.D.$$

$If\ D.E > 30$ *filter for the Distance Error

$$\{ D.E. = 30 \}$$

$$Else\ If\ D.E. < -30$$

$$\{ D.E. = -30 \}$$

$$PID = Discrete\ PID + 2 * D.E. + 60$$

$$PING\ \epsilon\ [0\ ,255]$$

$$P.D. = K$$

$$D.E. \epsilon\ [-K\ , 255 - K]$$

$$D.E. \epsilon\ [-30\ ,30]$$

*We decided to use a filter for the Distance Error (input of the PID) in order to avoid very high or very low values.

Platooning logic:

1. We want the robot to autonomously decide what is the base speed (that then will enter in the fuzzy logic for the path tracking) it needs to maintain a constant distance with the leader.

2. An integral part in the controller is used because just using a proportional controller we would be able to maintain a constant distance with the leader but not the distance we want. The integral controller resets to 0 when the error gets very small between 2 and -2 Inside the Platooning (previous page figure) an interval block of Simulink has been used to say to A (binary value) when the error is between 2 and -2.

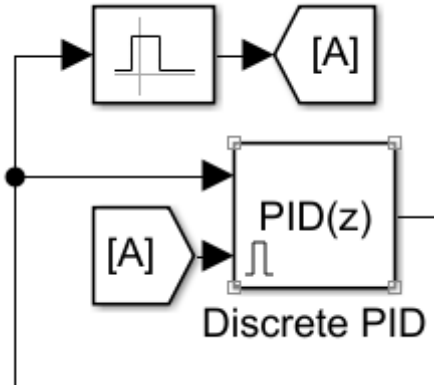


Figure 30 Integral part for platooning

Mathematical equations for Figure 30:

$$K_i: constant\ integral\ gain$$
$$Discrete\ PID = K_i * D.E.$$
$$If\ D.E. \leq 2\ \&\&\ \geq -2$$
$$\{A = 1\}$$
$$Else$$
$$\{A = 0\}$$
$$If\ A = 1$$
$$\{integral(D.E.) = 0\}$$
$$Else$$
$$\{integral(D.E.) = D.E.(i) + D.E.(i+1) + \cdots\}$$
$$integral(D.E.)\ \epsilon\ (-\infty, +\infty)$$
$$Discrete\ PID\ \epsilon\ (-\infty, +\infty)$$

3. Once the error is very small, we also want to maintain the speed achieved. For this purpose, in case A=1 we give the motors the previous speed through a memory block:

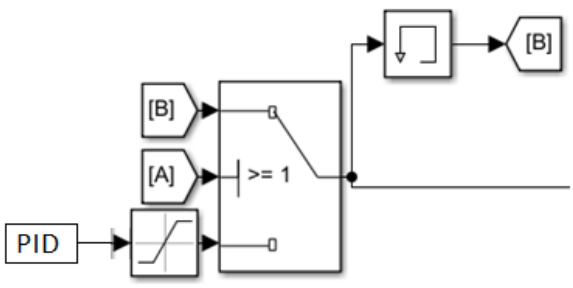


Figure 31 Maintain speed in platooning

Mathematical equations for Figure 31:

Platooning Speed = P.S. = B (in the Simulink)

From (1): $PID = Discrete\ PID + 2 * D.E. + 60$

$$If\ PID > 100$$
$$\{PID = 100\}$$
$$If\ PID < -100$$
$$\{PID = -100\}$$

$If\ A = 0$

$$\{P.S. = PID\}$$
$$Else$$
$$\{P.S. = P.S.(i-1)\}$$
$$P.S. \epsilon\ [-100\ , 100]$$

4. The last change we made was to consider the average speed for a large window length of point B (Platooning Speed) in order to saturate the maximum speed of the platooning and avoid the robot increase the speed very fast in the curve when it loses the contact of the leader robot.

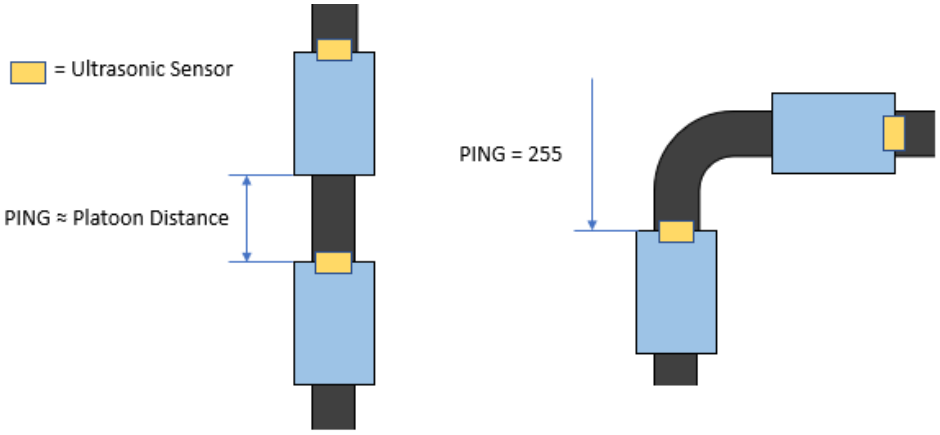


Figure 32 Platooning problem in the curve

In the above figure we can see that in the curve PING = 255 and then Platooning Speed goes suddenly to 100. At this speed, the follower robot will collide with the leader robot if the distance set between them is very small. In order to avoid this problem, the average speed of the follower robot in previous samples (P.S.(i-n)) has been considered and the Final Platooning Speed increases very slowly once the PING = 255.

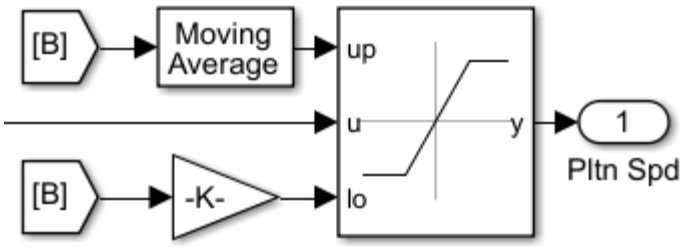


Figure 33 Platooning Saturation

Mathematical equations for Figure 33:

Platooning Speed = P.S. = B (in the Simulink)
After Moving Average = A.M.A.

$$P.S.(A.M.A) = \frac{P.S(i) + P.S.(i-1) + P.S.(i-2) + \cdots + P.S.(i-n)}{n}$$

$$If\ P.S. > P.S.(A.M.A)$$
$$\{Final\ P.S. = P.S.(A.M.A.)\}$$
$$If\ P.S. < -P.S.(A.M.A.)$$
$$\{Final\ P.S. = -P.S.(A.M.A.)\}$$
$$Final\ P.S. \epsilon\ [-P.S.(A.M.A., P.S.(A.M.A.)]$$

This Final Platooning Speed is the one that goes into the Motor Logic Block for path tracking.
Other option we tried to avoid the robot increase the speed very fast in the curve was to obtain the derivative error of the Distance Error and that when it arrived at a very high value, delay the platooning action by some time. In Figure x.10 when the follower loses contact with the leader robot:

$$When\ PING\ goes\ to\ 255 : \frac{d(Distance\ Error)}{dt} \to \infty$$

$$When\ PING\ \approx Platoon\ Distance : \frac{d(Distance\ Error)}{dt} \to 0$$

However, sometimes the ultrasonic values were not stable in the straight line (giving high derivative values when PING ≈ Platoon Distance) and we had to try other methods to solve this problem.

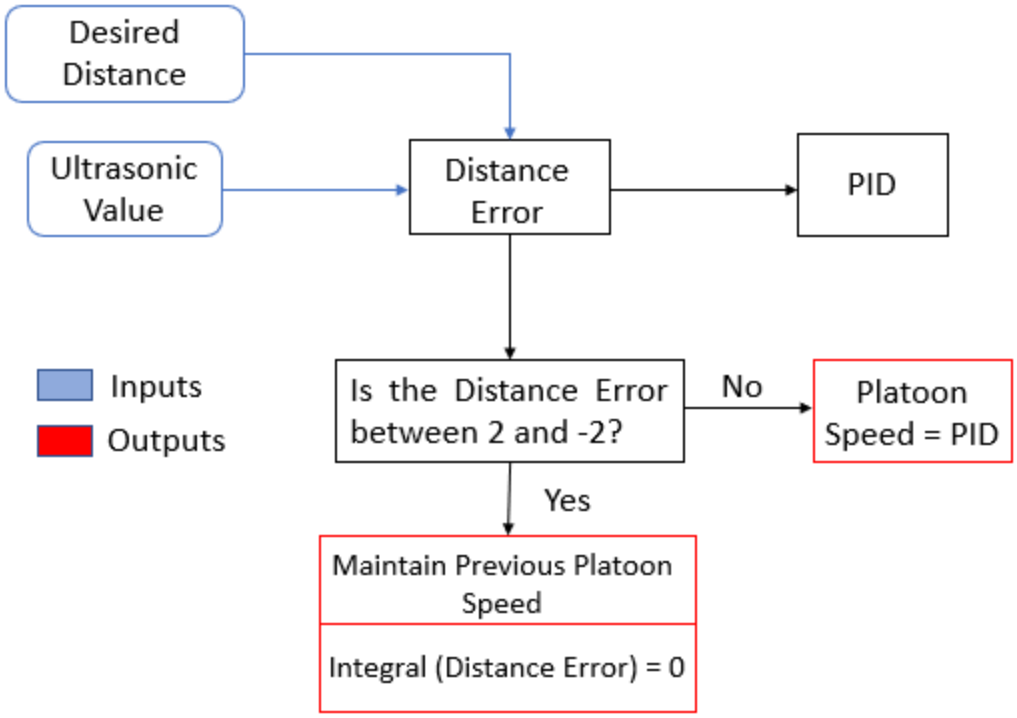


Figure 34. Platooning Block Diagram

As it can be understood from the mathematical equations and the platooning block diagram, once an acceptable error (between 2 and -2) has been achieved, we do not continue modifying the speed of the follower robot.

Send Data to GUI Block

The Send Data to GUI block sends the sensor values to the user to be used by the GUI. The values that are sent are the left RLI, the color sensor, the right RLI, the PING distance, the left encoder, the right encoder, and the battery. The battery values are changed to a percentage of the total amount of battery remaining before being sent. These values aid the user in knowing various aspects of the EV3 and its health. Figure 35 below shows the Send Data to GUI block.

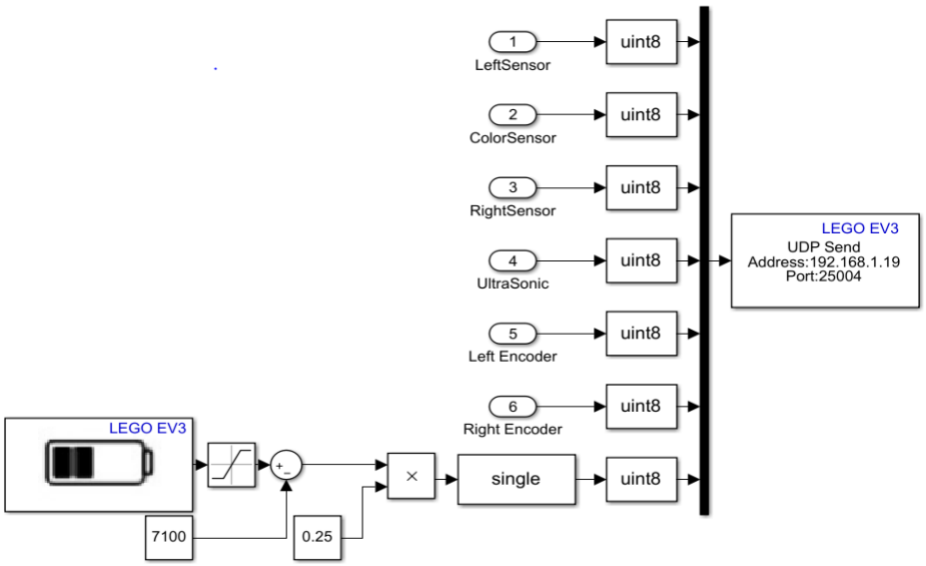


Figure 35. Send Data to GUI Block Schematic

Display Block

The Display block contains two sub-blocks which are a Battery Linearizer block and a Status Indicator block. The Battery Linearizer block transforms the raw battery data into a linearized battery percent from 0 to 100. This battery percentage is used in the Path Tracking main block as well as the Status Indicator sub-block. The Status Indicator block lights up the LED on the EV3 brick to different colors based on the mode the EV3 is in as well as battery life. In path tracking, the LED is green. In platooning, the battery is flashing green. In parking the LED is flashing orange. If the battery drops below 20 percent, the LED is flashing red. If the battery drops below 10 percent the LED is red. Also, if the battery drops below 10 percent, an auditory tone will play from the EV3 brick to also alert the user to its low battery level. The Status Indicator block was implemented so that it is easy for the user to get visual feedback from the EV3 on its current status. This is especially helpful when debugging and knowing the health of the EV3 in a new way. Figure 36 below shows the Display block.

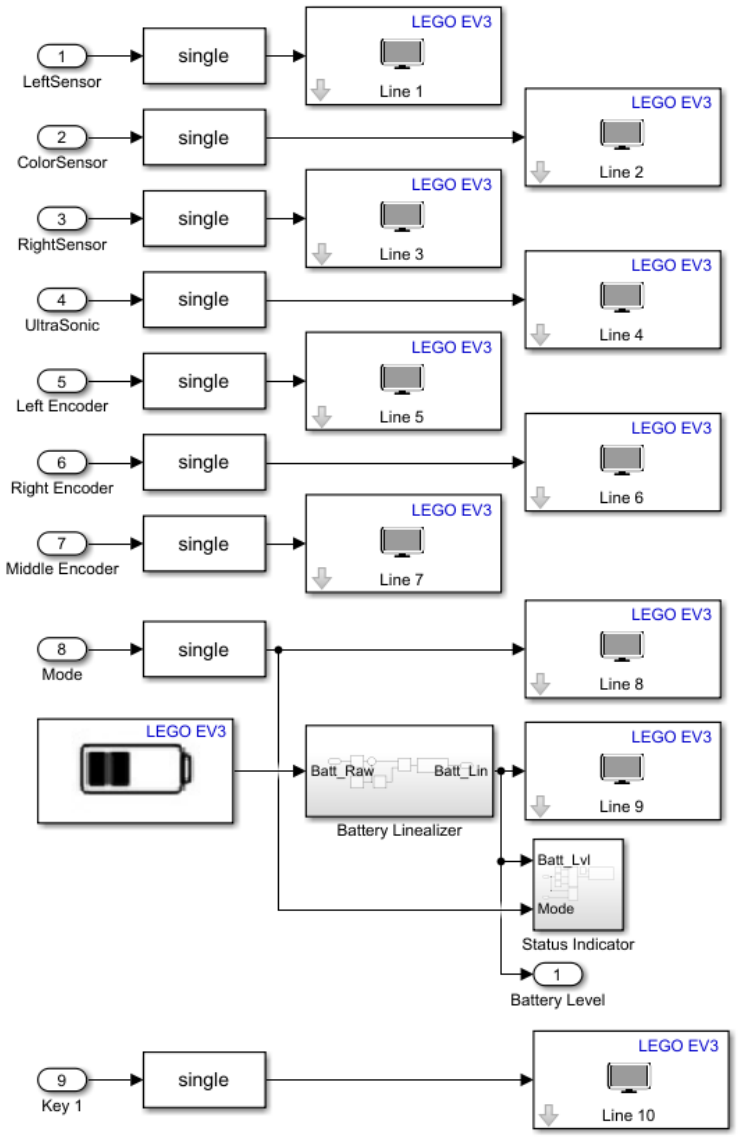


Figure 36. Display Block Schematic

Motor Logic Block (Davis)

The Motor Logic block handles all the logic for the two DC motors as well as the PING servo. The Motor Logic block uses the left and right RLI (Reflected Light Intensity) as well as the PING (ultrasonic) sensor for several rules. These rules are to prevent the EV3 from driving if it runs off the track, collision avoidance, and preventing crashing when de-parking. Figure 37 below shows the high-level block diagram for the Motor Logic as well as the Servo Logic blocks. Figure 38 below shows the updated Motor Logic block.

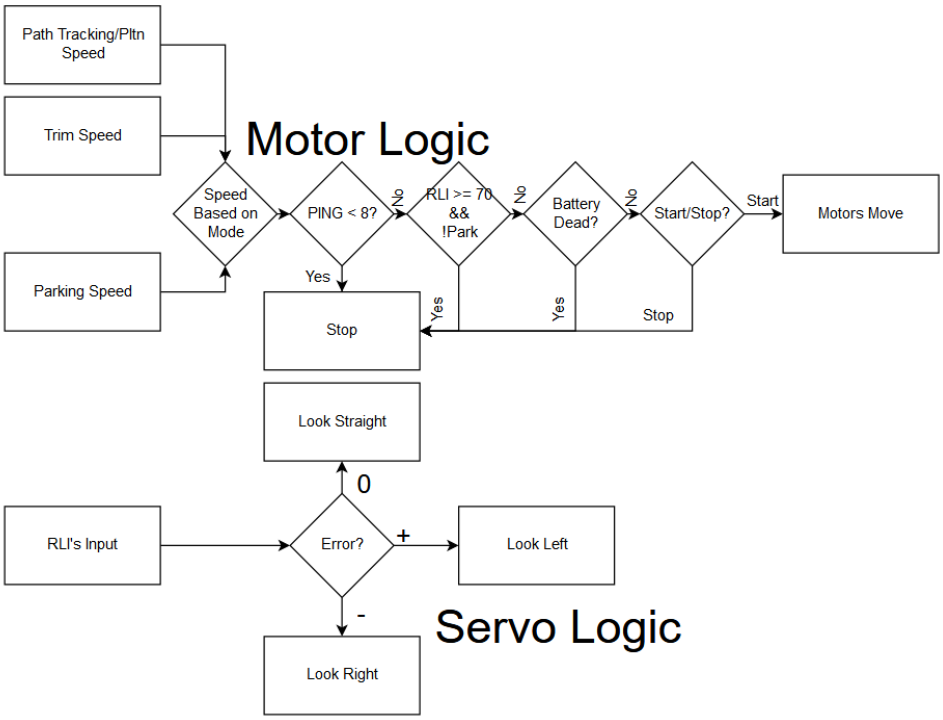


Figure 37. Motor Logic and Servo Logic Block High Level Flowchart

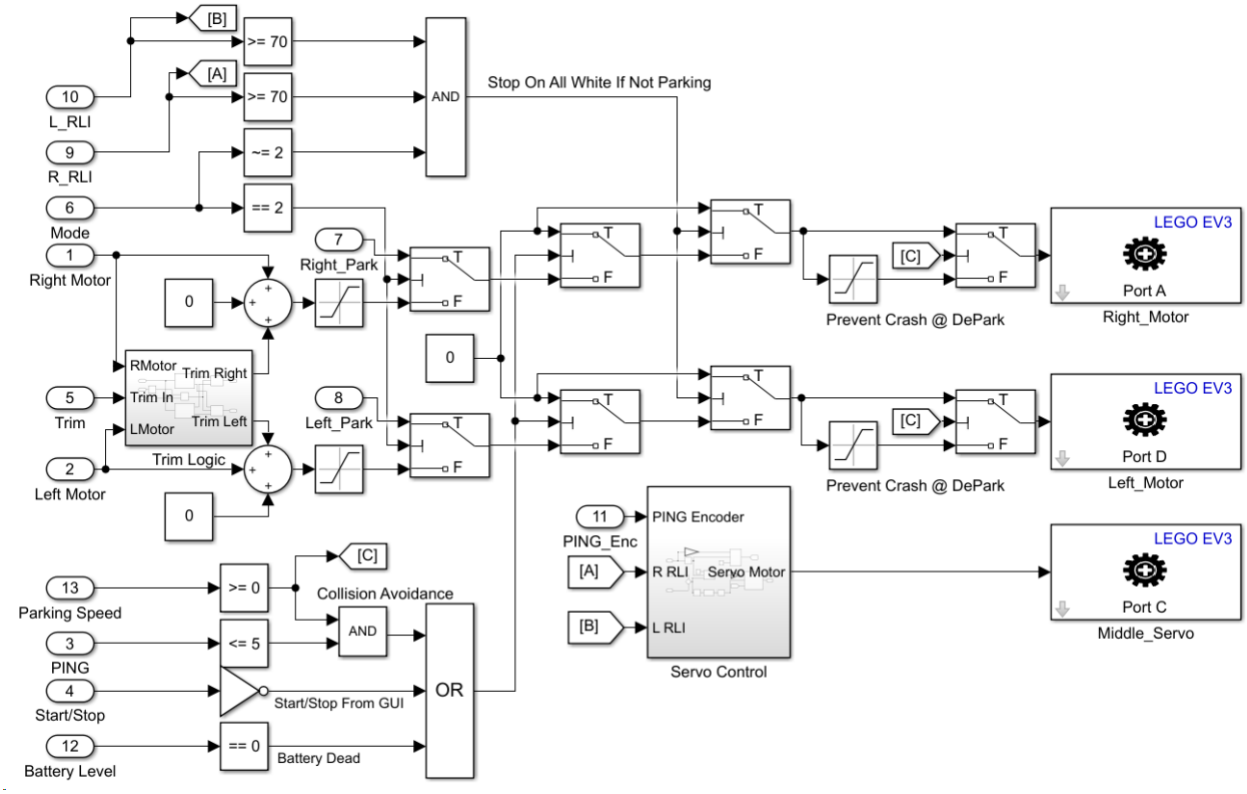


Figure 38. Motor Logic Block Schematic

I noticed that sometimes even on a straight line an EV may drift towards one side. To compensate for this, I developed the Trim block to correct any drifting that may occur. The block adds very small amounts to the input speed to offset the drift. I have also found you can use the Trim block to also make turns to drive on the inner sub-tracks. At lower speeds less than or equal to 50 percent, the trim value is added to the speed of the specific wheel. The speeds greater than 50 percent, the trim value is subtracted from the wheel. The GUI sends a value which is used in the trim equation. A 0 trim value results from the GUI sending an input of 10. The GUI sends values between 1 and 19. Ten is subtracted from the input value and if the result is positive then the result is added to the right wheel. Conversely, if the result is negative then the value is added to the left wheel. Equation 5 below shows the trim input-output relationship for speeds less than or equal to 50 percent. Equation 6 below shows the relationship for speeds greater than 50 percent. Figures 39 and 40 below show the equations in use with a given input range of 0 to 20. The changing trim values can be seen in these figures. Figure 41 is the Trim block diagram.

$$Trim_{low} = \{x - 10,\ 1 \leq x \leq 19\ 0,\ else \tag{5}$$

$$Trim_{high} = \{-(x - 10),\ 1 \leq x \leq 19\ 0,\ else \tag{6}$$

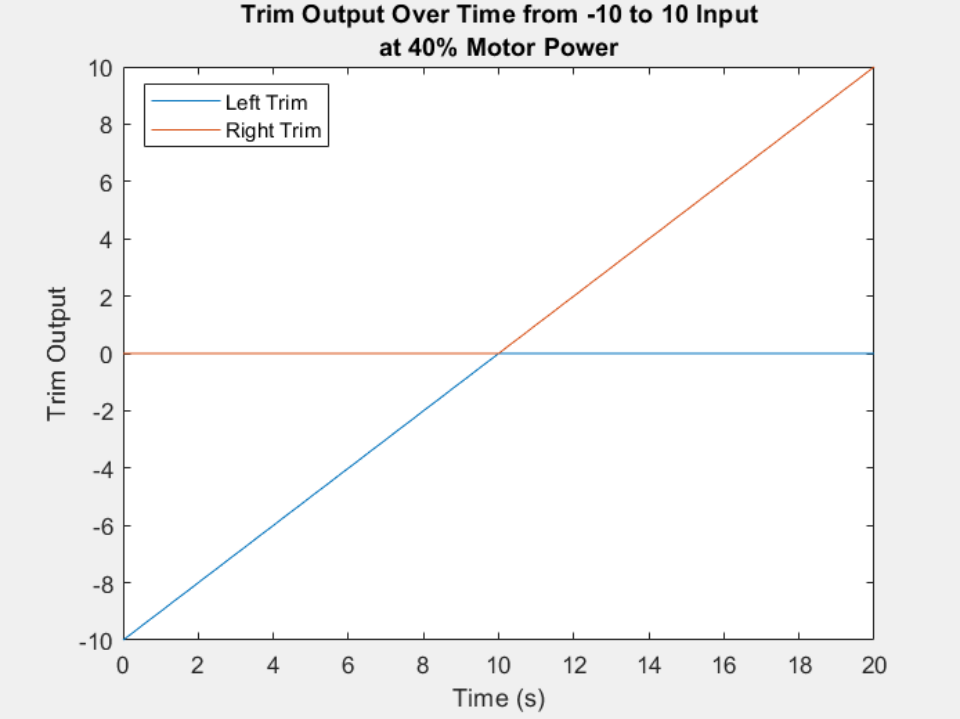


Figure 39. Low Trim Output

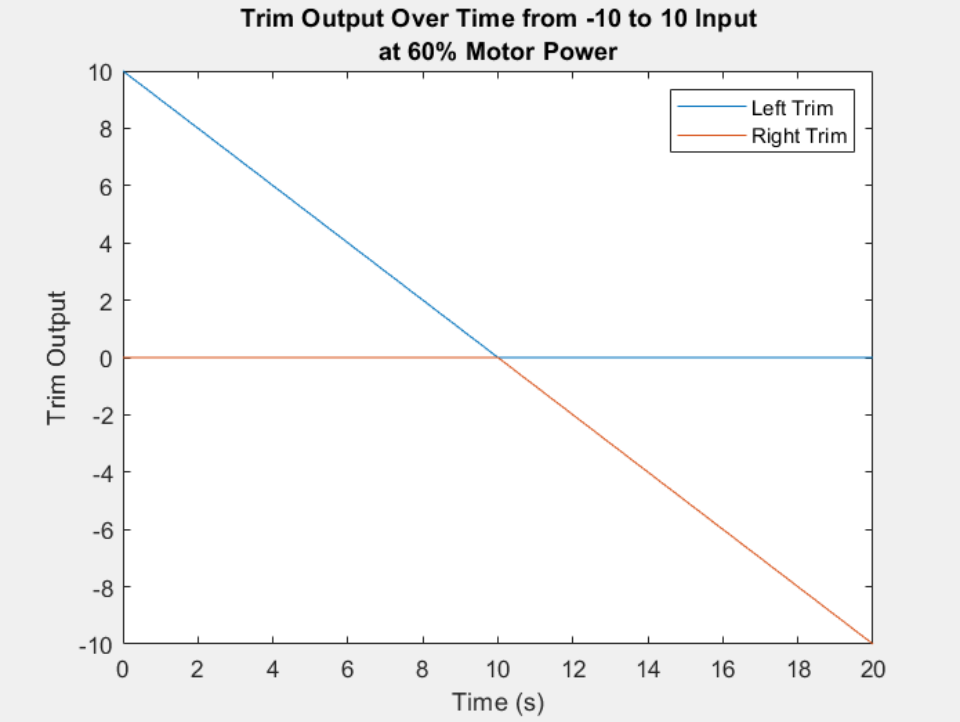


Figure 40. High Trim Output

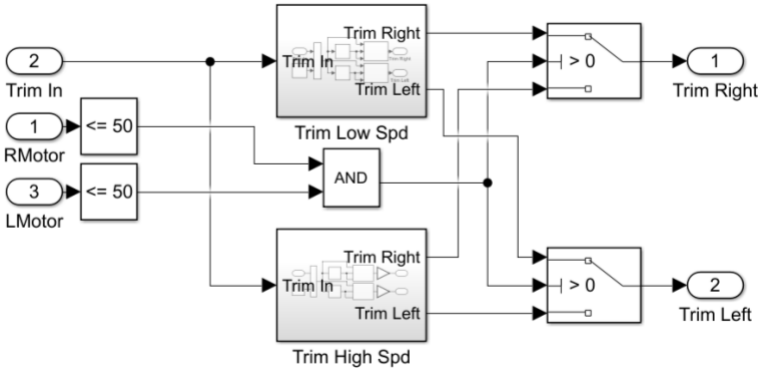


Figure 41. Trim Block Schematic

Servo Control Block (Juan)

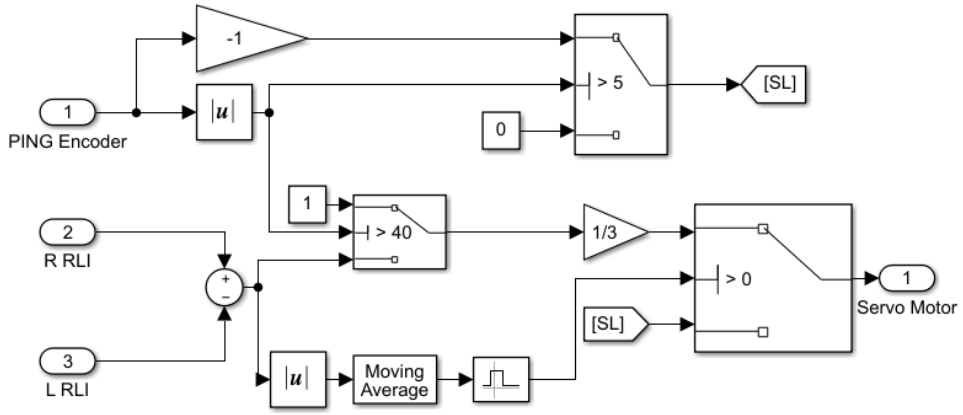


Figure 42. Servo Motor

We can see three inputs are given to the servo motor block. The first one is the servo motor encoder that gives data related with the servo position and the other inputs are used to obtain the light error. We want the servo to face where the robot is going, so if the robot is on a curve the servo should turn and when it is on a straight line the servo should come back to its original position. As explained in the PID_Bias section, the light error is close to 0 when the robot is on a straight line and higher than $|30|$ when it is on a curve.

Mathematical equations Servo Motor (Figure 42):

$$RLI: Reflected\ Light\ Intensity\ (Raw\ values\ obtained\ from\ light\ sensor).$$

$$RLI\ E. = R.RLI - L.RLI$$

P.E. = PING Encoder: gives a positive value when the ultrasonic is facing right, negative when it is facing left and 0 when the ultrasonic is in its original position.
S.M. = Servo Motor: Moves the motor connected to the ultrasonic changing its position and for this reason, it changes the PING Encoder values.

$If\ |RLI\ E.| > 30$ *Curve case

$$\left\{S.M. = \frac{1}{3} * RLI\ E.\right\}$$

$* If\ we\ are\ in\ a\ curve\ we\ move\ the\ ultrasonic\ to\ one\ side\ or\ to\ the\ other\ depending\ on\ the\ curve$
$Else$ *Straight Line case

$$\{$$

$If\ P.E. > |5|$

$$\{S.M. = -P.E.\}$$

$$If\ P.E. < |5|\ or\ P.E. > |40|$$

$$\{S.M. = 0\}$$

$$\} *$$

*If we are in a straight line, we want the ultrasonic to come back to its original position.
Also, when P.E.>|40| the ultrasonic will not move (S.M. = 0), this way we do not move the ultrasonic staying in a fixed position (40, -40 or 0)

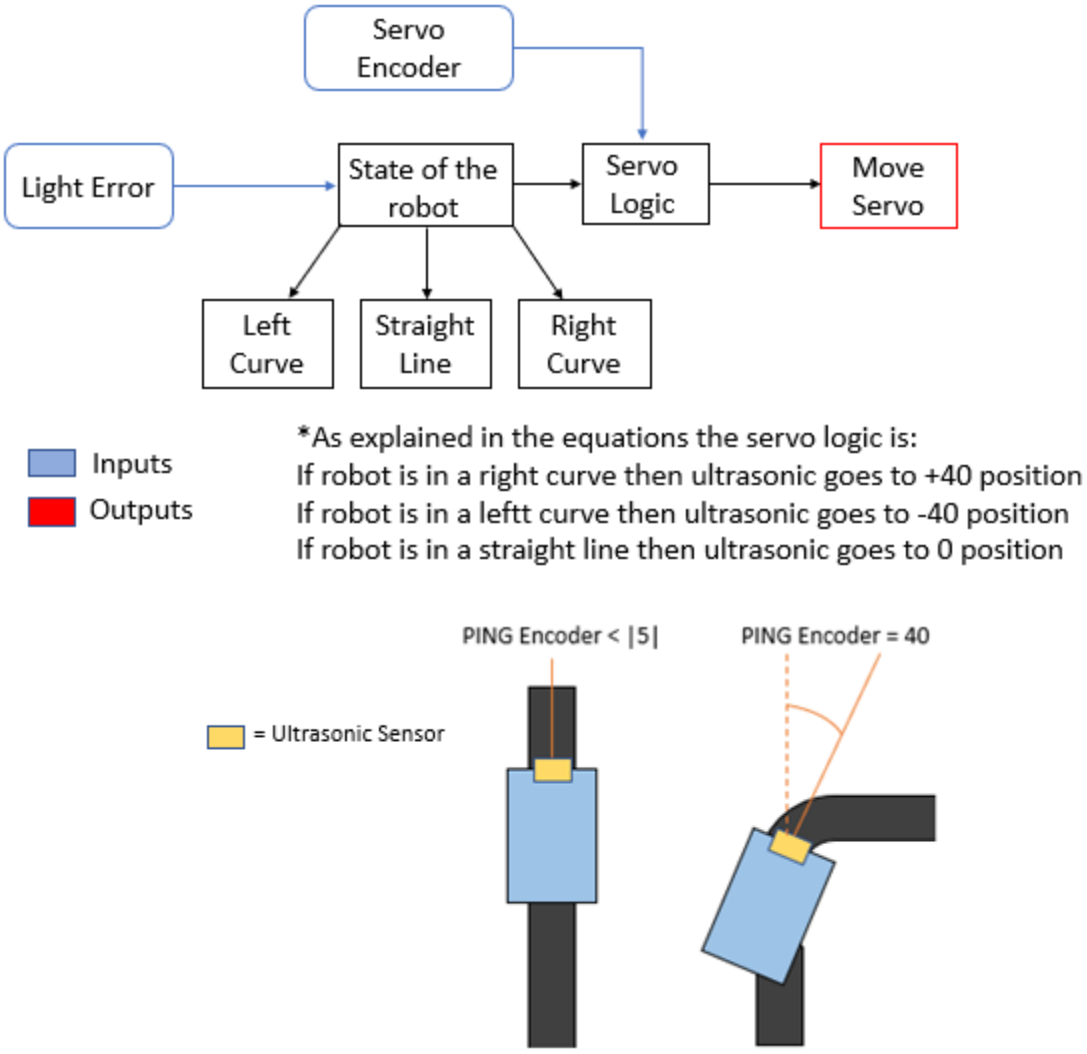


Figure 43. Right curve makes servo motor turn to a fixed 40 position.

Send Data to EV3 Block (Davis)

The Send Data to EV3 block sends the current EV3 commands to the other active EV3s. The commands that are sent overwrite the commands sent from the GUI. This block is activated by the Leader/Follower command from the GUI. If an EV3 is set to leader then it will send its received commands from the GUI to all the active EV3s. Ideally, future revisions would have all the commands for all four EV3s sent to all four EV3s. This means that EV1 would receive the commands for EVs 1-4. So, should an EV3 detect that it has lost connection to the GUI, it would be able to ask other EV3s for its commands to continue normal operation. Figure 44 below shows the Send Data to EV3 block.

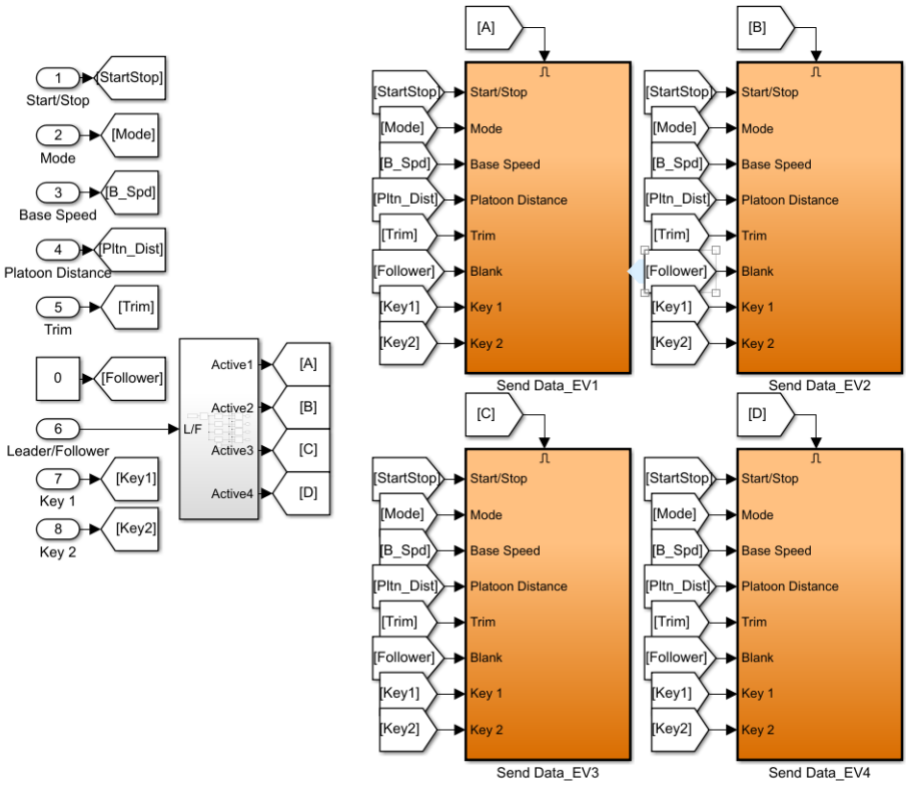


Figure 44. Send Data to EV3 Block Schematic

Security Block (Davis)

To relate the project to future technologies such as Vehicle-to-Vehicle or Infrastructure-to-Vehicle communications, I wanted to create a block that would act as a security wall to protect the EV3 from hacking attempts. The security in the block is a very low security measure with using only two security keys although the security measures could be expanded upon. In actual application, the vehicle would ignore hacking attempts. With this block, it expresses the hacking attempt to show users in an educational setting that the EV3 is trying to be hacked. This block takes the two loaded security keys and compares them to the security keys that are sent. If they match, then the EV3 performs the commands received. If the security keys are mismatched then the security block deactivates the Path-Tracking, Platooning, Parking, and Motor Logic blocks. This prevents the EV3 from performing any actuation tasks. Also, the block causes the EV3 to swivel about its center of movement. An alarm sounds as well to alert the user. Figure 45 below shows the Security block.

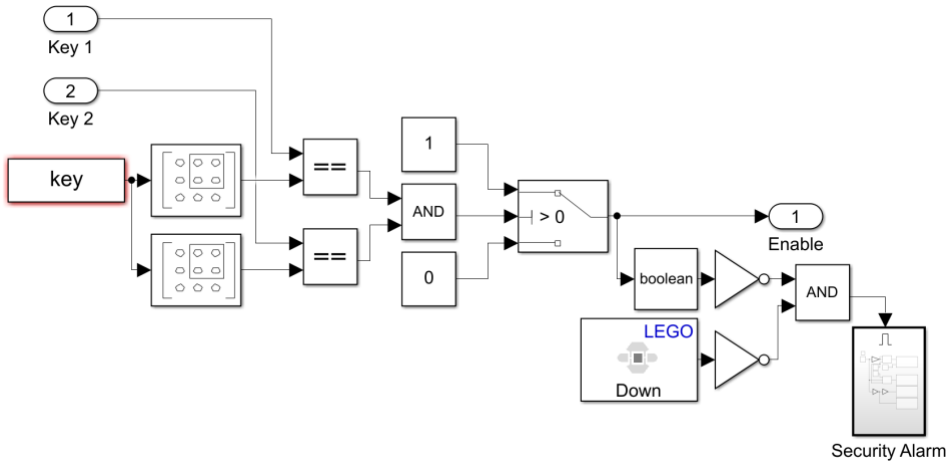


Figure 45. Security Block Schematic

Four Sensors in a Line (Juan)

Following the advice of Dr. Chow and as an extra credit part of the project, a robot for path tracking that uses four sensors in a line was built, this way the robot will be able to follow the edge of the line by taking into account what it has in front of him. With this sensor displacement the robot should behave much better than how we have it for the other EV3s because it will consider the error in advance. Also, a gear system has been implemented in this model to increase the maximum speed of the robot.

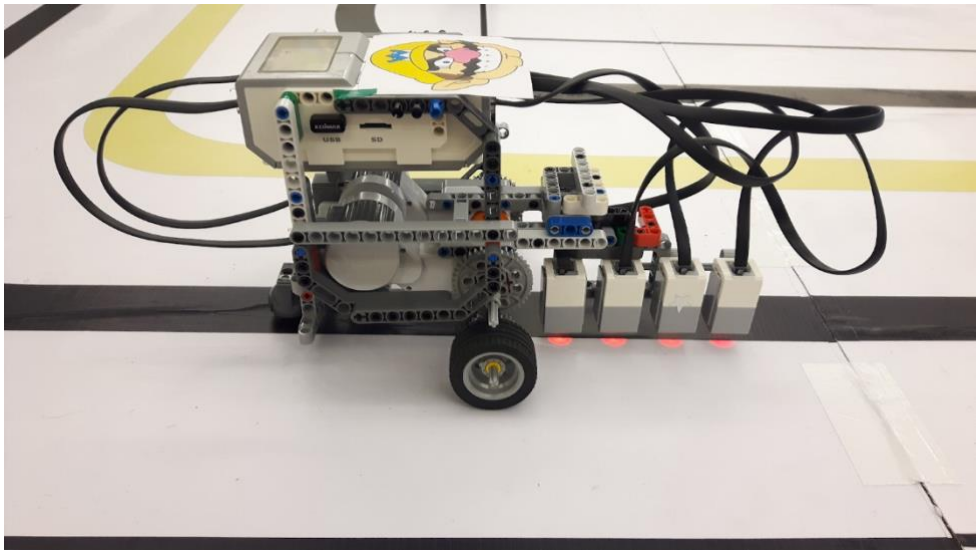


Figure 46. EV3 Robot with 4 Light sensors

Figure 47. Gear system

In this new case we thought about controlling three errors as it can be seen in the following image.

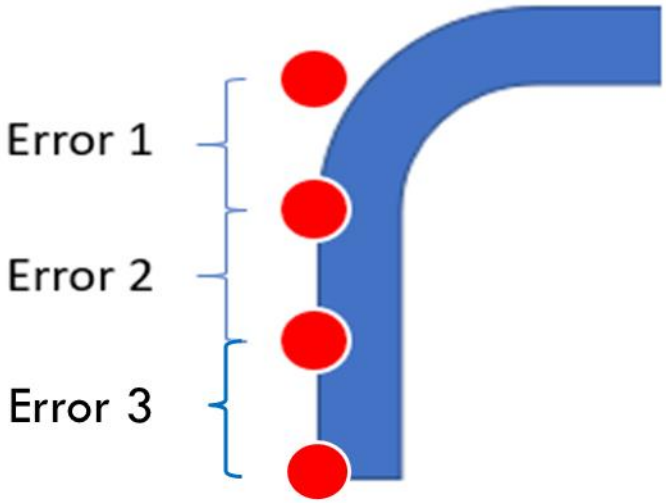


Figure 48. Errors considered for the Fuzzy Logic

We decided to use the same fuzzy (same membership functions, different outputs) that we were using for the two sensors inside the line, the fuzzy block mission in this case will be to tune the proportional gain of a proportional controller for the error 1, error 2 and error 3.
When error 2 or error 3 become higher means the EV3 needs more turning power to make the turn. For this reason: $Kp_3 > Kp_2 > Kp_1$

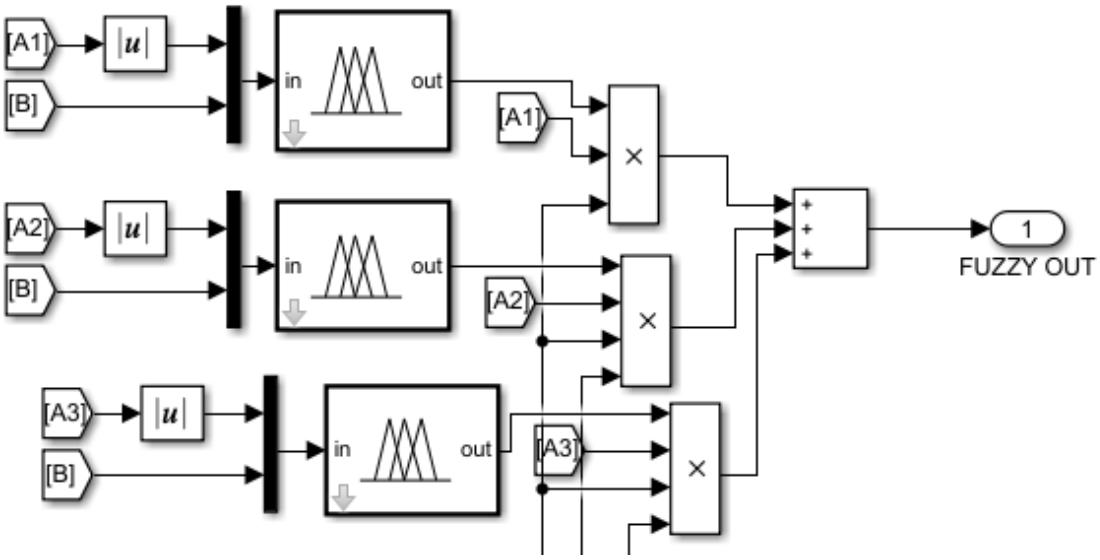


Figure 49. Fuzzy Logic for 4 sensors in a line

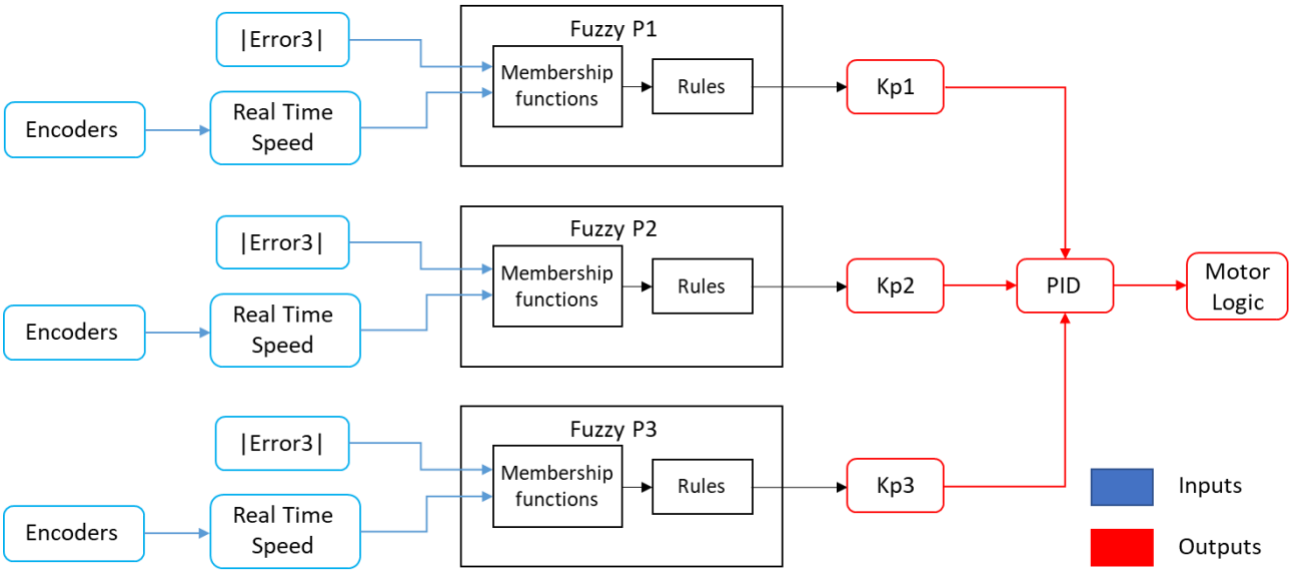


Figure 50. 4 sensors Fuzzy Diagram

The motor logic is based on changing the speed of each wheel according to the error the EV3 is reading from the sensors:

$$Wheels\ Speed = Base\ Speed + Kp_1 * Error1 + Kp_2 * Error2 + Kp_3 * error3$$

To see how the sensor positioning affects the path tracking smoothness, an analysis has been done in the “Path tracking and Platooning evaluation” section of this document.

Path Tracking and Platooning Evaluation

Following the grading criteria established in the ECE 756 course:

1. Accumulated tracking error index C1:

$$C_1 = \sum_{k=1}^{N} |\mathrm{RLI}(k) - \mathrm{RLI}_{\mathrm{ref}}| \cdot \Delta t$$

The N signifies total number of data points; RLI is the normalized (0-100) reflected light intensity reading from the light sensor; Δt is the sampling time (1 second).

2. Accumulated tracking smoothness index C2:

$$C_2 = \sum_{k=1}^{N-1} \left| \frac{(\mathrm{RLI}(k+1) - \mathrm{RLI}_{\mathrm{ref}}) - (\mathrm{RLI}(k) - \mathrm{RLI}_{\mathrm{ref}})}{\Delta t} \right| \cdot \Delta t$$

3. Platooning task performance

$$C_3 = \sum_{k=1}^{N} |\mathrm{Distance}(k) - \mathrm{Distance}_{\mathrm{ref}}| \cdot \Delta t$$

The Distance(k) is the raw ultrasonic sensor reading; $\mathrm{Distance}_{\mathrm{ref}}$ is set as 20 cm. The A+ is achieved when $C1 < 700, C2 < 800 \; and \; C3 < 1300$

C1, C2 and C3 have been calculated for all the robots built to check their reliability.
For the main system (2 sensors PD controller), after 20 trials these are the C1, C2 and C3 obtained:

Trials consisted on one EV3 platooning other EV3 for one lap inside the inner loop of the track at 65% of speed. The C1, C2 and C3values are calculated for the follower robot.
For the system with 4 sensors in a line, after 20 trials these are the C1 and C2 obtained:

Trials consisted on the 4 light sensors' robot path tracking for one lap inside the inner loop of the track at 90% of speed. Comparing the average results for both systems after the trials, we can see the C1 and C2 values are smaller using 4 sensors in a line than for the main system. This makes sense because putting the sensors in a line makes the EV3 able to see the curve before than if the sensors are not in a line. However, the 4 sensors' EV3 is more unreliable, sometimes it gets out of the track because once the 4 sensors for any reason get out of the edge and when all of them receive white the EV3 loses its path. For this reason, maybe the best solution would be to mix both configurations to have the reliability of two sensors at the same height and the smoothness of the sensor in a line displacement.

Communication (Binoy)

Introduction

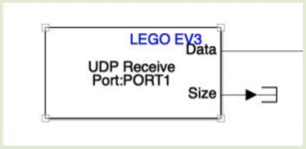
Communication between the various components in the project was done using the Wi-Fi network and data was transferred using UDP communication. UDP communication was used over TCP/IP. TCP/IP communication has a lot of overhead which ensures that packets are not lost during the communication. After a packet is transmitted, it waits for an acknowledgement before transmitting the next packet. UDP communication doesn't really have a lot of overhead and it is possible that data packets are lost during transmission. A few lost packets don't affect our system in any way. Hence, UDP communication was used. In order to send and receive data from the EV3, we used the UDPSend/ UDPReceive block of the LEGOEv3 toolbox in

Simulink. The UDPReceive block receives UDP packets from an IP network and saves it in a buffer. The block outputs data which send a UDP packet in a one-dimensional vector of the specified data type. We set the data type to be int8 and the packet size to 7 because we are receiving a 7 valued vector. For the parking we used another packet with 4 values as shown in Table 6. We used the UDPSend block in Simulink to send data from the EV3 to the GUI. We set the Remote IP Address as the IP address of the PC on which the GUI is running and the port address as any open port on the PC. On MATLAB, we used the DSP System Toolbox to send and receive data through the GUI. The UDPReceiver and UDPSend System objects were used for receiving and sending data through the GUI.

The dsp.udpreceiver receives UDP packets from a remote IP Address which we can set using the Remote IP Address property. This then saves the data on a buffer. We can set the maximum buffer size by using the ReceiveBuffersize property. dsp.udpsender sends the data from the GUI to the LegoEV3. This function sends a packet of data to the specific port which is given in the Remote IP Port field. The data has to be sent in the int8 format. We specify a port for each of the UVs and each UV sends and receives data from the same Remote IP Port. The data sent and received from the ports are specified in Table 6 given below.

Table 6. UDP communications

GUI control			Simulink control
Dsp sender		Dsp receiver	
Start/stop	Present yaw	Left sensor	
Mode	Target yaw	Right sensor	
Base speed	Parking speed	Ultrasonic	
Platooning distance	Brake	Left encoder	
Trim value		Right encoder	
Leader/follow		Battery	
Key(security)			



The buffer size was set to one so that any data that enters the buffer is immediately forwarded to the system. This ensure that the latest information was forwarded to the EV3 block

Analysis of Network Traffic

We noticed that the designed systems worked fine for one EV, but they start to misbehave when more than one EVs are connected. One of the reasons was the network traffic on the system. Communication between the client PC, the host Optitrack PC and the EV3s were done using the Wi-Fi network.

For the purpose of analysis, we considered rate of the packets transferred, latency and did an endpoint analysis for the endpoints on the network.
Source IP: 192.168.0.25
Client PC: 192.168.0.132

EV3 IP: 192.168.0.103
EV2 IP: 192.168.0.102
EV1 IP: 192.168.0.101

When no EVs are connected						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	60	6137	86	527k	0.152	10

When 1 EV is connected without parking mode						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	67	6786	274	1771k	0.671	26
PC-EV1	67	6515	49	319k		4.76

When 1 EV is connected with parking mode						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	62	6334	274	1720k	0.671	27.7
PC-EV1	62	6078	54	301k		4.85

When 2 EVs are connected without parking mode						
	Time	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	40	4139	369	1703k	1.508	42.5
PC-EV1	40	3975	54	301k		4.85
PC-EV2	40	4139	49	202k		5.05

When 2 EVs are connected with parking mode						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	65	6615	369	2741k	1.919	42.16
PC-EV1	65	6360	54	311k		4.78
PC-EV2	65	6616	49	324k		4.98

When 3 EVs are connected with parking mode						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	43	4357	510	2390k	2.553	55.5
PC-EV1	43	4181	54	204k		4.74
PC-EV2	43	4212	49	206k		4.79
PC-EV3	43	4353	49	213k		4.95

When 3 EVs are connected with parking mode						
	Time(sec)	Packets	Packet Size	Bytes transferred	Latency(msec)	Throughput(kbps)
Host Client	55	5477	510	3036k	2.885	55.69
PC-EV1	55	5255	49	257k		4.67
PC-EV2	55	5315	49	260k		4.72
PC-EV2	55	5473	49	268k		4.87

Latency is the amount of time it takes for the packets to get transmitted. We noticed that the latency kept increasing as the number of EV3s started increasing. Latency also increased in the parking mode of operation compared to normal operation.
The throughput kept increasing as the number of EV3s went up. This is expected and the rate of transfer of data must increase as the data increases. This made us believe that the issues were with the bandwidth available for the Client PC and the EV3s.

OptiTrack Implementation (Binoy)

Introduction

OptiTrack is the creator of precision motion capture and 3D capture systems used for a large number of applications. The system present in the iSpace lab consists of 8 cameras fitted on the ceiling which can capture the exact position and orientation of trackable using reflective markers present on the objects. The OptiTrack systems use an application called Tracking tools which streams the data to a PC which can be set on the application. The data is sent as UDP packets. The packets consist of the position of the robots along with the orientation information. The EV3 is fitted with reflective markers. They have to be fixed in a different orientation to ensure that each is a separate trackable. The OptiTrack system present in ISpace works like a Positioning system which can be used to display the position and orientation information and use it for our application.

Hardware Setup

Optical motion capture systems utilize multiple 2D images from each camera to compute, or reconstruct, corresponding 3D coordinates. To ensure good capture of trackable:

- Place the cameras circumnavigating around the capture volume, so that markers in the volume will be visible by at least two cameras at all times.
- Minimize ambient lights, especially sunlight and other infrared light sources.
- Clean capture volume. Remove unnecessary obstacles within the area.
- Tape, or Cover, remaining reflective objects in the area.
- If there are any reflective points that cannot be removed, then they must be masked
- In order to obtain accurate and stable tracking data, it is very important that all of the cameras are correctly focused to the target volume. If the markers are blurry, focus the cameras so as to make the object less blurry
- The trackable attached on the EV3 must be kept high such that a shadow is not seen on the balls and they are clear to the infrared cameras of the Optitrack

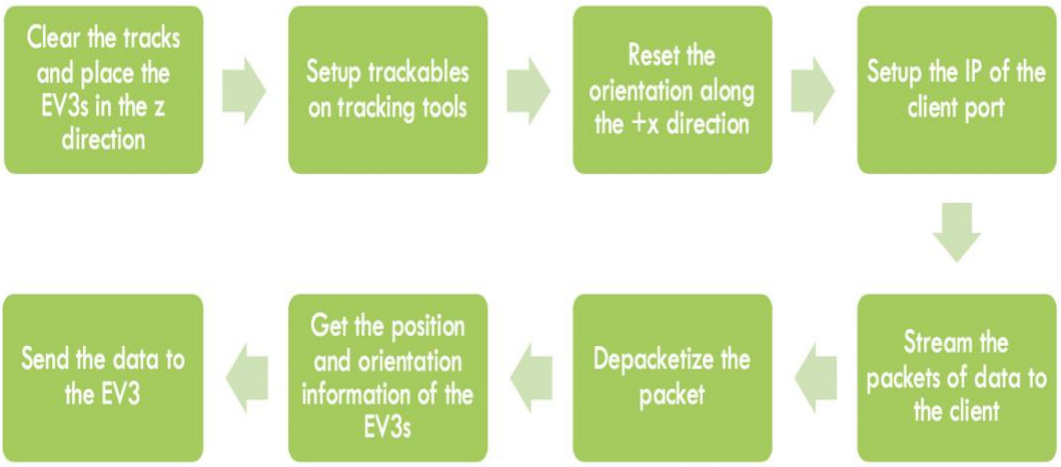


Figure 51. Optitrack Setup

Calibration

During camera calibration, the system computes position and orientation of each camera and amounts of distortions in captured images. Using calibration data, Optitrack constructs a 3D capture volume. Even if setups are not altered, calibration accuracy will naturally deteriorate over time due to ambient factors, such as fluctuation in temperature and other environmental conditions. The calibration steps were followed as given in the user manual. The 3- trackable wanding was completed to generate sampling data and the ground plane was also set. After we calibrated the system, we observed that

- The x and z axes are in line with the map

- The dead zones have reduced

Tracking Tools

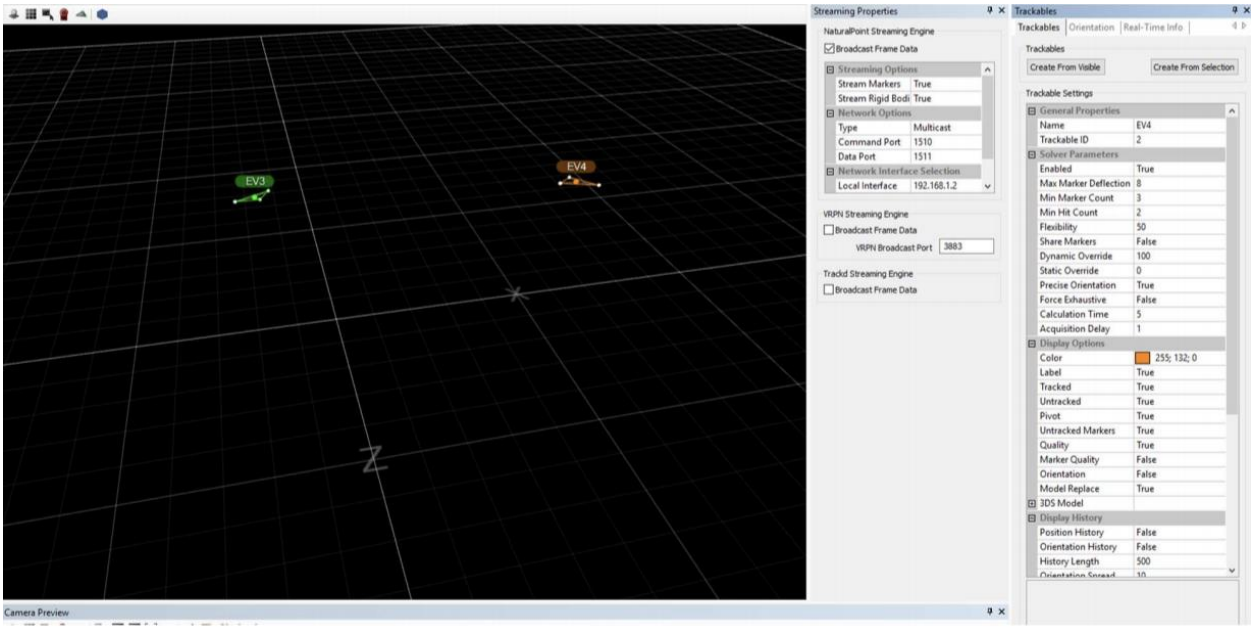


Figure 52. Tracking tools

The trackable were observed on an application known as TrackingTools. The EV3s are placed on the area and then the trackable are set on the application. When we set the trackable the application creates a rigid body

Rigid Bodies

A Rigid Body is a collection of three or more markers on an object that are interconnected to each other with an assumption that the tracked object is not deformable. A minimum of three markers must be used to define a rigid body. The data was received on the Client PC by means of UDP packets. These UDP packets contained information of all the reflective dots and the position and orientation of the rigid bodies. All the rigid bodies must have a unique set of trackable orientation.

For each of the rigid bodies to be unique, the must have

- A unique marker arrangement

- A unique marker to marker distance

- A unique marker count

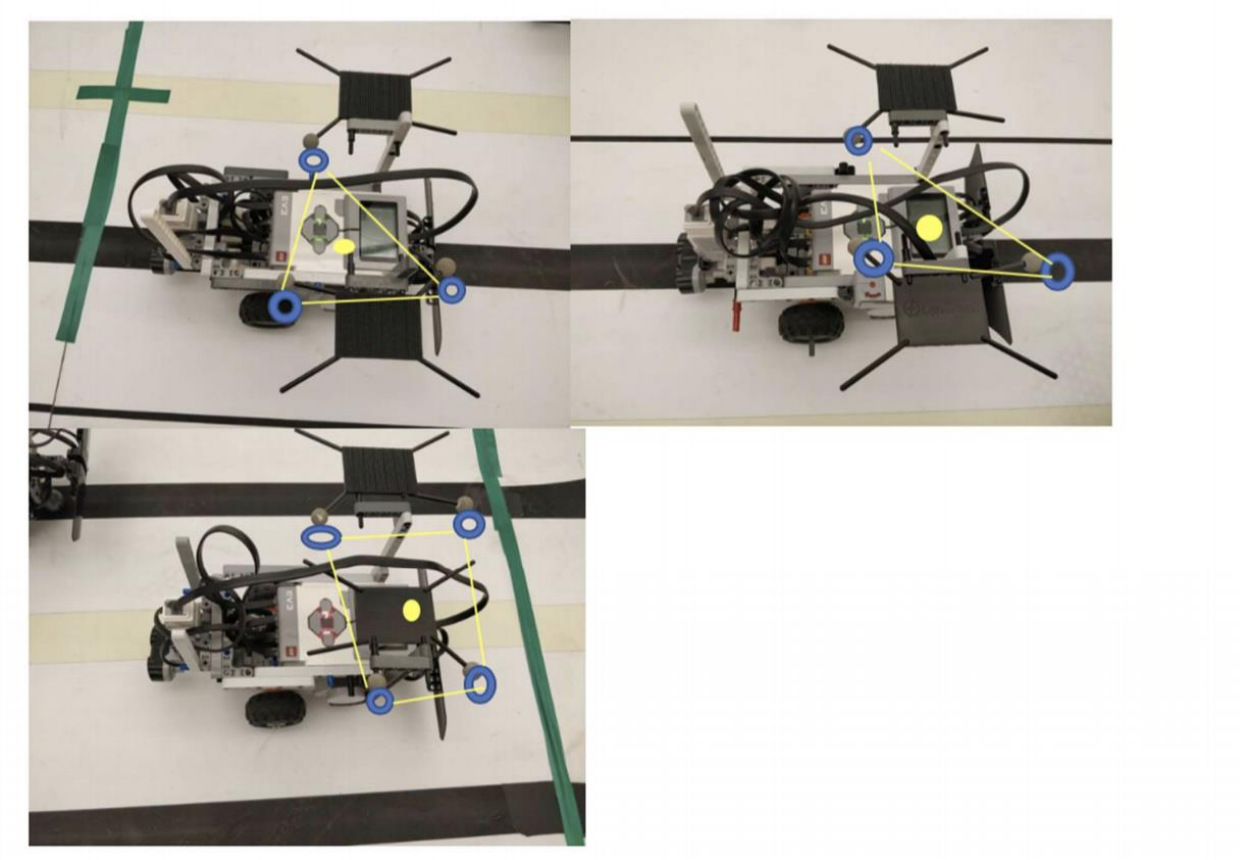


Figure 53. Rigid body placement of reflective balls

Approach for Data Streaming

There are primarily two approaches to get data from the TrackingTools application. NatNet is a client/server networking protocol which allows sending and receiving data across a network in real-time. It utilizes UDP along with either Unicast or Multicast communication for integrating and streaming reconstructed 3D data, rigid body data, and skeleton data from OptiTrack systems to client applications. The approach called direct depacketization was used to obtain the tracking information in MATLAB. This approach is mostly used when we are trying to stream data to an application that is not compatible with tracking tools. MATLAB is such an application and hence this method was used for obtaining the position and orientation information in MATLAB.

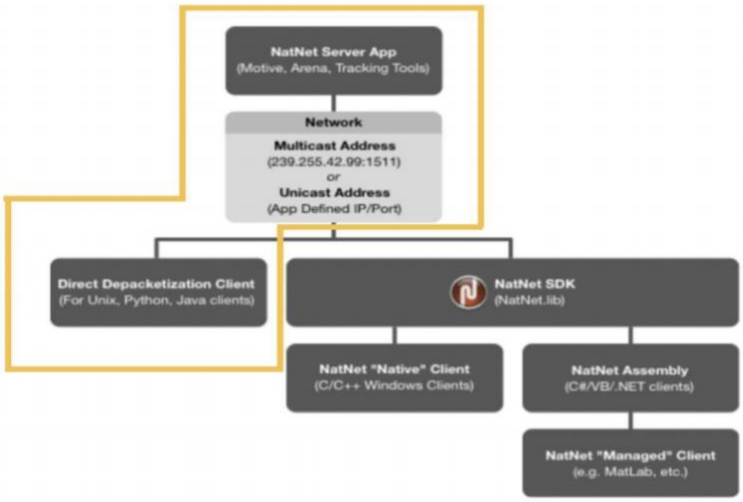


Figure 54. Data streaming approaches

Depacketization

The depacket function goes through the packet of data obtained in MATLAB and picks out only the data that was useful. The number of trackables and the number of rigid bodies can be variable. It was necessary to create a robust function which could allow any number of trackables and rigid bodies. The depacket function outputs the position and orientation information in the form of quaternions of all the rigid bodies.

```
-------------
Begin Packet
-------
Message ID : 7
Byte count : 357
Frame # : 0
Marker Set Count : 2
Model Name: Trackable 2
Marker Count : 3
        Marker 0 : [x=2.01,y=0.13,z=-2.96]
        Marker 1 : [x=1.91,y=0.13,z=-2.93]
        Marker 2 : [x=2.01,y=0.13,z=-2.86]
Model Name:
Marker Count : 0
Unidentified Marker Count : 6
        Marker 0 : pos = [1.97,0.10,-2.60]
        Marker 1 : pos = [1.91,0.13,-2.93]
        Marker 2 : pos = [2.01,0.13,-2.96]
        Marker 3 : pos = [2.09,0.10,-2.55]
        Marker 4 : pos = [2.01,0.13,-2.86]
        Marker 5 : pos = [2.08,0.10,-2.61]
Rigid Body Count : 2
ID : 1
pos: [2.05,0.10,-2.58]
ori: [0.01,0.81,-0.01,0.59]
Marker Count: 3
        Marker 0: id=1  size=0.0        pos=[2.09,0.10,-2.55]
        Marker 1: id=2  size=0.0        pos=[1.97,0.10,-2.60]
        Marker 2: id=3  size=0.0        pos=[2.08,0.10,-2.61]
Mean marker error: 0.00
ID : 2
pos: [1.98,0.13,-2.92]
ori: [-0.01,-0.66,0.00,0.75]
Marker Count: 3
        Marker 0: id=11 size=0.0        pos=[2.01,0.13,-2.96]
        Marker 1: id=12 size=0.0        pos=[1.91,0.13,-2.93]
        Marker 2: id=13 size=0.0        pos=[2.01,0.13,-2.86]
Mean marker error: 0.00
Skeleton Count : 0
latency : 2512.523
End Packet
-------------
```

```
-------------
Begin Packet
-------
Message ID : 7
Byte count : 216
Frame # : 0
Marker Set Count : 1
Model Name: Trackable 1
Marker Count : 3
        Marker 0 : [x=1.95,y=0.10,z=-2.59]
        Marker 1 : [x=1.88,y=0.10,z=-2.67]
        Marker 2 : [x=1.92,y=0.10,z=-2.72]
Unidentified Marker Count : 3
        Marker 0 : pos = [1.95,0.10,-2.59]
        Marker 1 : pos = [1.88,0.10,-2.67]
        Marker 2 : pos = [1.92,0.10,-2.72]
Rigid Body Count : 1
ID : 1
pos: [1.92,0.10,-2.66]
ori: [0.01,0.14,-0.00,0.99]
Marker Count: 3
        Marker 0: id=1  size=0.0        pos=[1.95,0.10,-2.59]
        Marker 1: id=2  size=0.0        pos=[1.88,0.10,-2.67]
        Marker 2: id=3  size=0.0        pos=[1.92,0.10,-2.72]
Mean marker error: 0.00
Skeleton Count : 0
latency : 521.129
End Packet
```

Figure 55. Screenshot of the packet received on the client

Analysis of the packet obtained and explanation of the depacketization

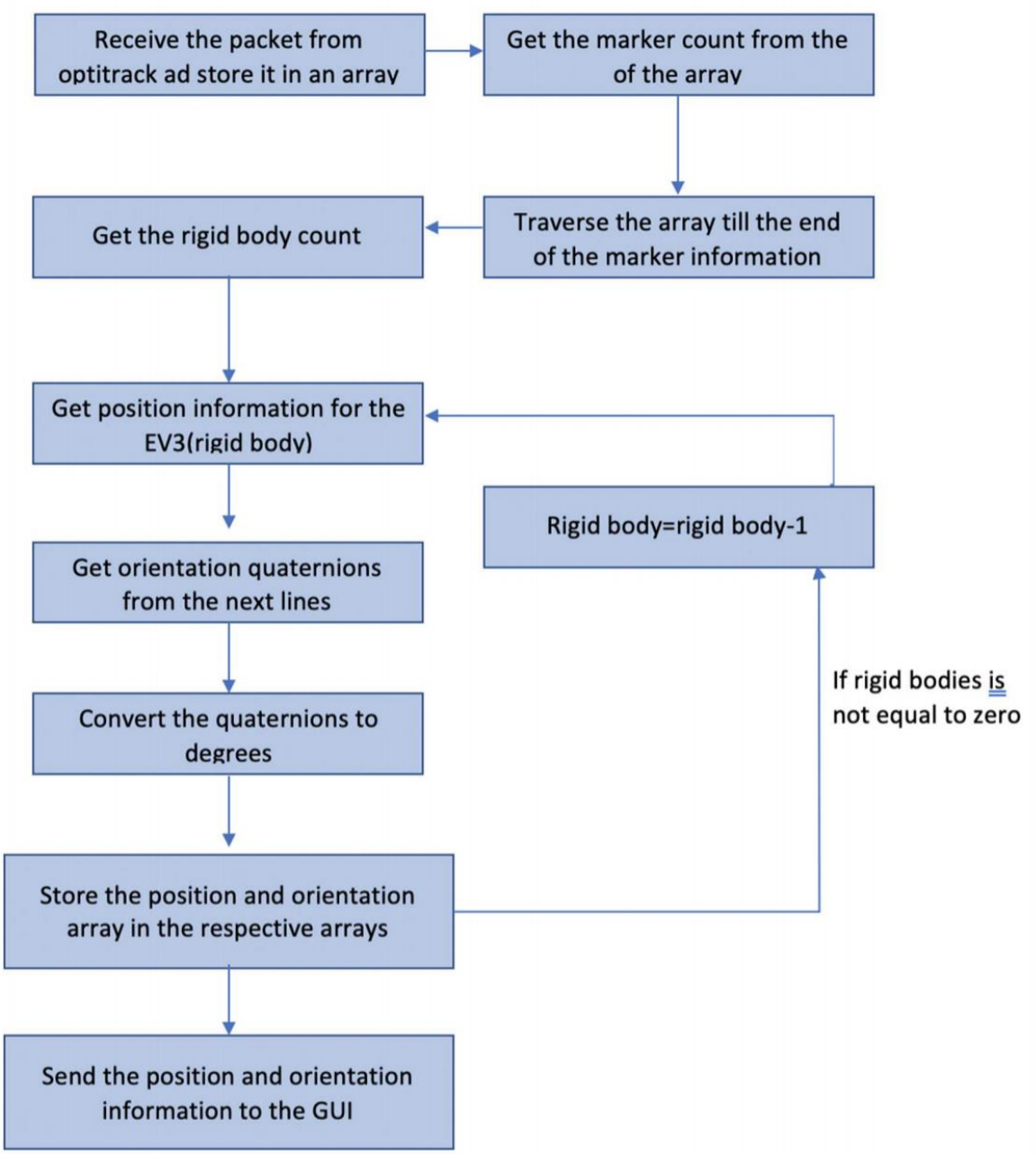


Figure 56. Depacketization

Analysis of Optitrack

During the course of the project, we faced some problems with the Optitrack. Below is an analysis of the possible reasons for this

1. Reflections off the track

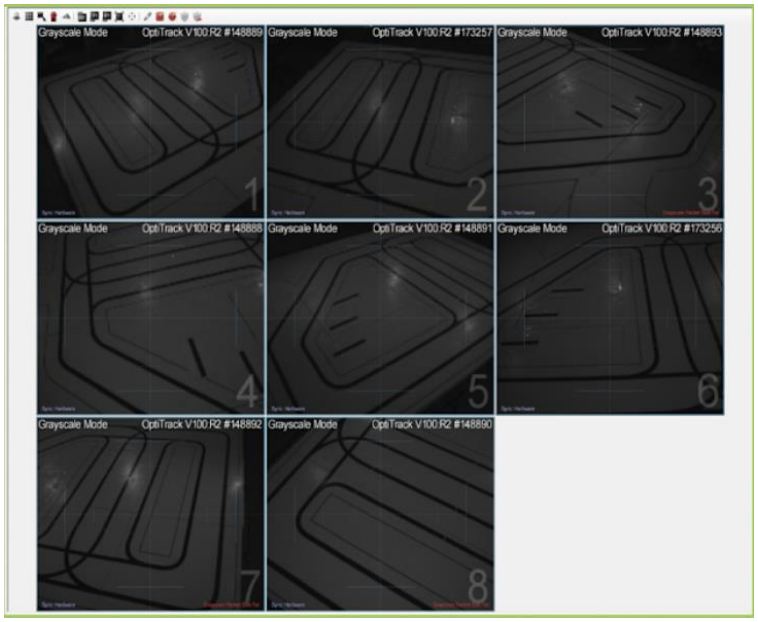


Figure 57. Reflections

During calibration we noticed that the infrared cameras on the ceiling would pick up reflections from light sources. These reflections would cause the trackables to be masked whenever they reached these positions.

2. Ghost balls appearing in the parking area

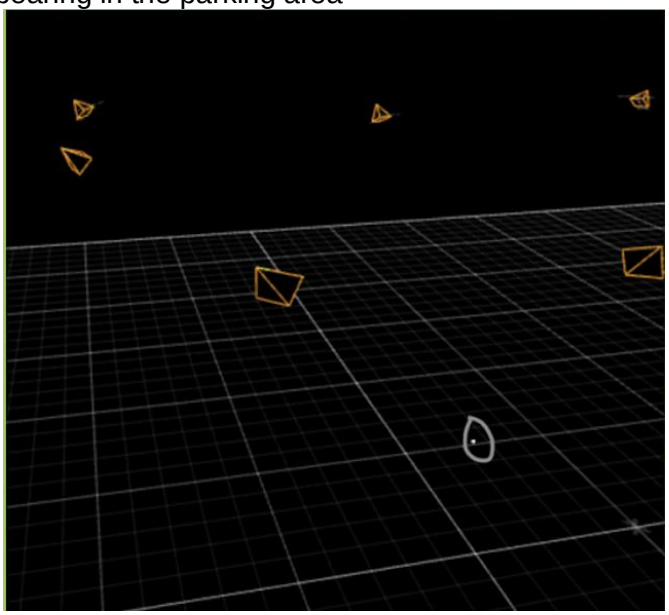


Figure 58. Ghost balls

Ghost balls would appear on the track sometimes when the EV3s are moving. This would cause the trackable balls to interchange and the EV3s would go untracked.

3. Interchangeable trackables
   Another problem was that even though the trackables were distinct from each other they would interchange which would drastically affect our system. We assumed that this is a flaw with TrackingTools

4. Balls not reflecting properly

After a lot of handling, the reflective balls had lost their reflective property and hence they do not reflect properly.

5. Communication overload

Since there are 4 EV3s and each of them would continuously communicate with the client PC and the host PC, the throughput would decrease as explained in the previous section

Quaternion

The quaternion is an abstract means of representing attitude. It is a four-dimensional vector used to describe a three-dimensional attitude representation. the 3-2-1 rotation sequence most commonly known as yaw (or heading), pitch, and roll. These three angles work great for most applications.

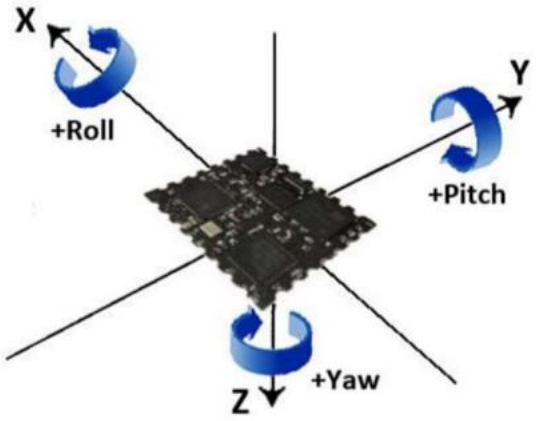


Figure 59. 3-2-1 Euler orientations

There is one problem with the Euler angle attitude representation; there exists two attitudes where you have a singularity in the solution. Take a look at Figure 59 and imagine that the pitch angle is equal to 90 degrees. In this case the yaw and roll perform the same operation. This may not be a problem. One way to get around this problem is to add an additional degree so that we have an over defined attitude representation. This is what the quaternion does in a very simplistic sense. So, quaternions basically represent four values. The first three are the values of the principal axes and the fourth is the principal angle. For our use, we need to extract the yaw value from the quaternions. The yaw is given by the formula

$$yaw = \tan^{-1}\left(\frac{2(q_0q_1 + q_3q_2)}{{q_3}^2 - {q_2}^2 - {q_1}^2 + {q_0}^2}\right)$$

Accuracy of the Optitrack

The Optitrack can be used to get the accurate position and orientation information of the EV3s. We conducted experiments to test the accuracy of the position and orientation values

Stopping box Tolerance	Observation
0 cm	The EV3 failed to stop in any test
3 cm	The EV3 failed to stop in any test
5 cm	The EV3 stopped on 30% of the trials
8 cm	The EV3 stopped on 60% of the trails
10 cm	The EV3 stopped on every try

Angle Tolerance	Observation
0 degrees	The EV3 kept oscillating around the point
2 degrees	The EV3 would not get the exact stopping angle in every try
4 degrees	The EV3 stopped exactly on the point
6 degrees	The EV3 overshot on 60 % of the tries
8 degrees	The EV3 overshot in every try

Parking Simulink

In vehicle dynamic control of the Ev3, controlling the lateral dynamic motion is very important where it will determine the stability of the vehicle. One of the approaches that are available for lateral dynamics control is a yaw stability control system. The yaw value gives the angle value with respect to the yaw axis of rotation. This value is used to control the EV3 when there is no line for it to follow. The position and the orientation information are used to control the UV to go from its present position to the desired position

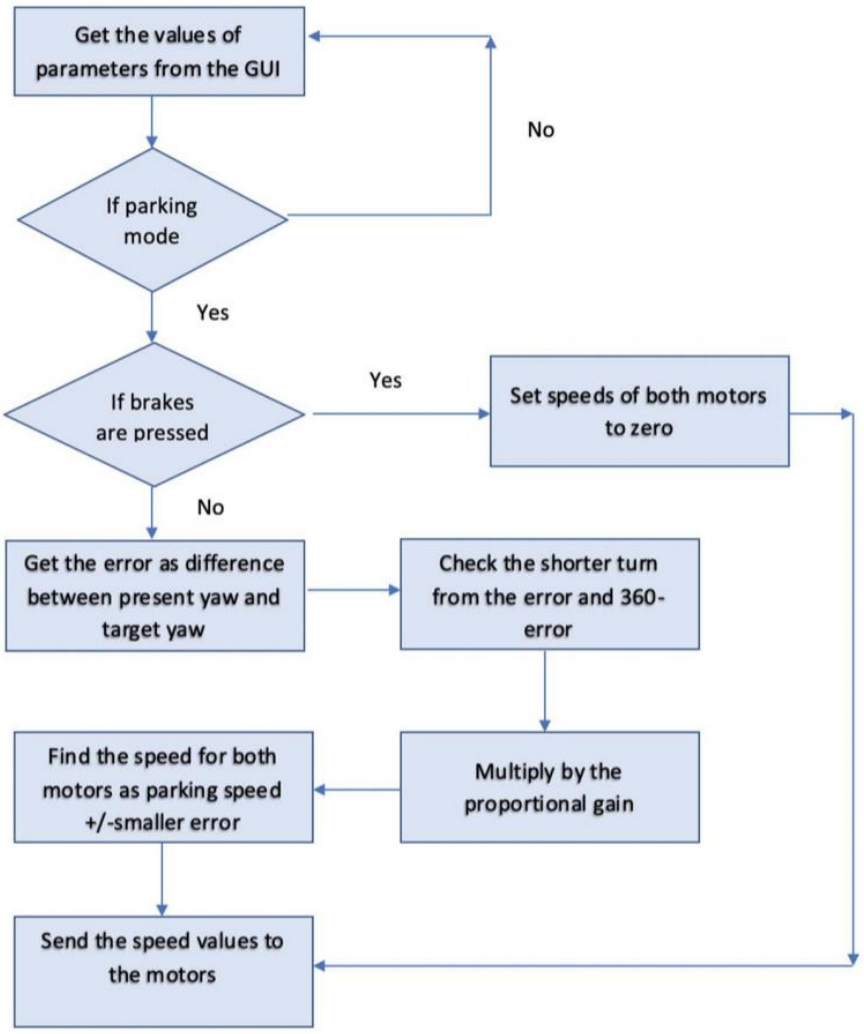


Figure 60. Flow of parking algorithm

The difference between the target position and the present position is taken and the angle with respect to a refence is taken. This refence is set to the z axis. Thus, the angle is obtained and the EV3 is given a command to follow a straight line at that angle. A simple P controller is used with a gain of one. The error is the difference between the present yaw and the final yaw. The parking block receives 2 inputs the present yaw, the target yaw and the base speed. There is a saturation block connected such that the rotation speed does not increase a value of above a value of60. When the EV3 has to travel in a straight line the base speed is set to 30 and when in reverse -30. The speed is set to 0 when it has to rotate in its place

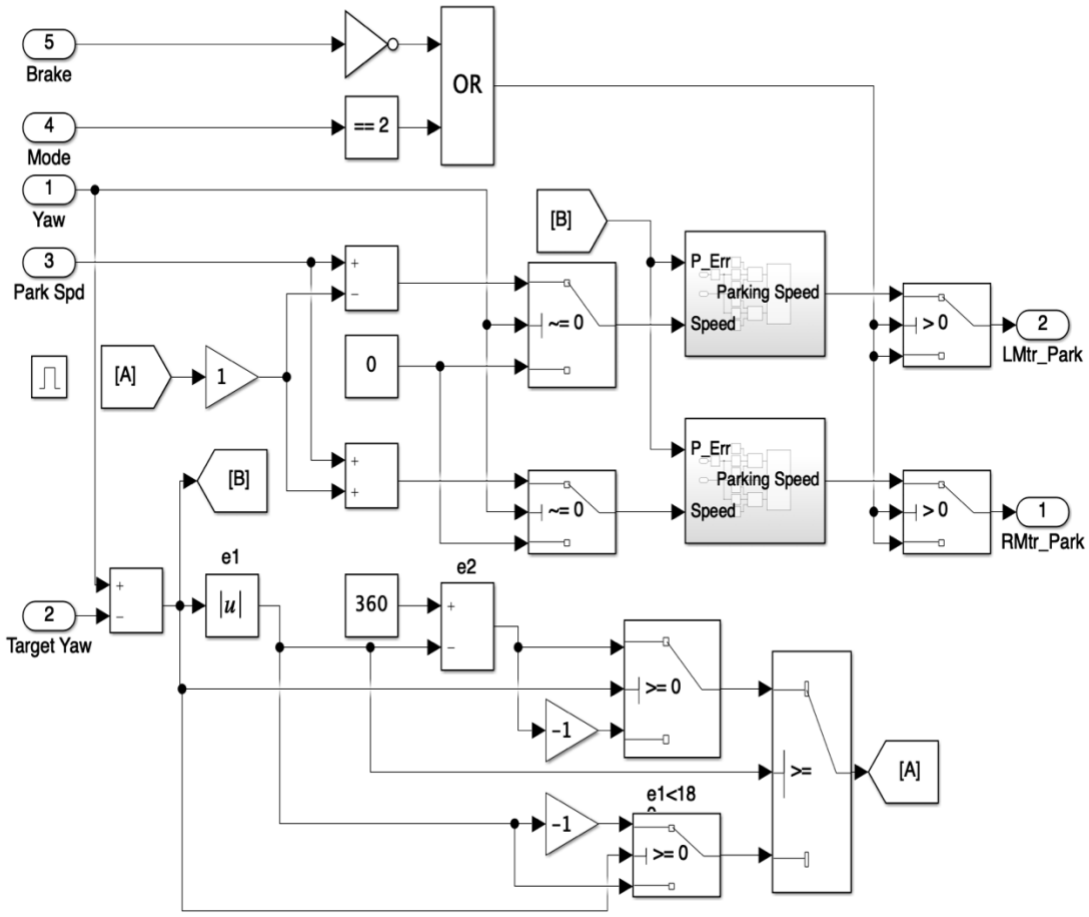


Figure 61. Parking Simulink block

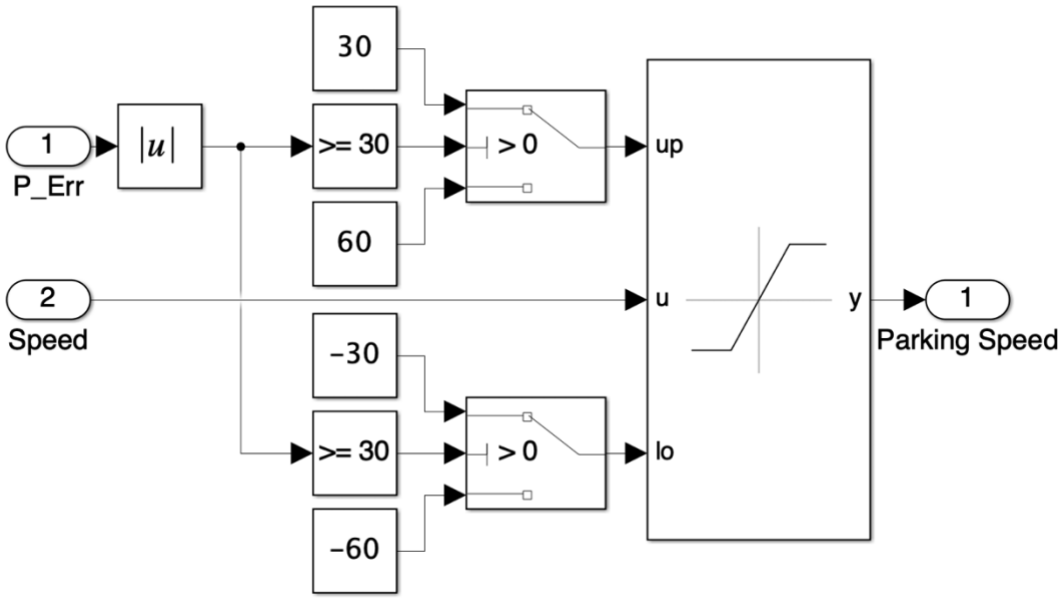


Figure 62. Speed control of Parking Block

$$error = present\ yaw - target\ yaw$$

$$Left\ Motor\ speed = Parking\ Base\ Speed + \min(|error|, 360 - |error|)$$

$$Right\ Motor\ speed = Parking\ Base\ Speed - \min(|error|, 360 - |error|)$$

Path Planning of EV3s

As part of our extra credits, we tried to emulate a Maps version of iSpace. For this purpose, we looked into algorithms to find the shortest path between any point on the map to any destination. We looked into two algorithms to complete these tasks

1) Dijkstra's shortest path algorithm

Dijkstra's algorithm is used to find the shortest path between nodes in a graph. It first initializes the nodes and then the weights i.e. lengths of the paths.

2) A* shortest path algorithm

The A* algorithm is very similar to the Dijkstra's algorithm. The difference between the two is that the A* algorithm has a heuristic function which is like a guess of the shortest path. It is like the Dijkstra's algorithm with a knowledge of the path already in it

We used the Dijkstra's algorithm to complete the task because our application only had about 18 nodes and the graph matrix was not really complex. The A* algorithm would give the results faster but they are not always right because the correctness depends on the heuristic

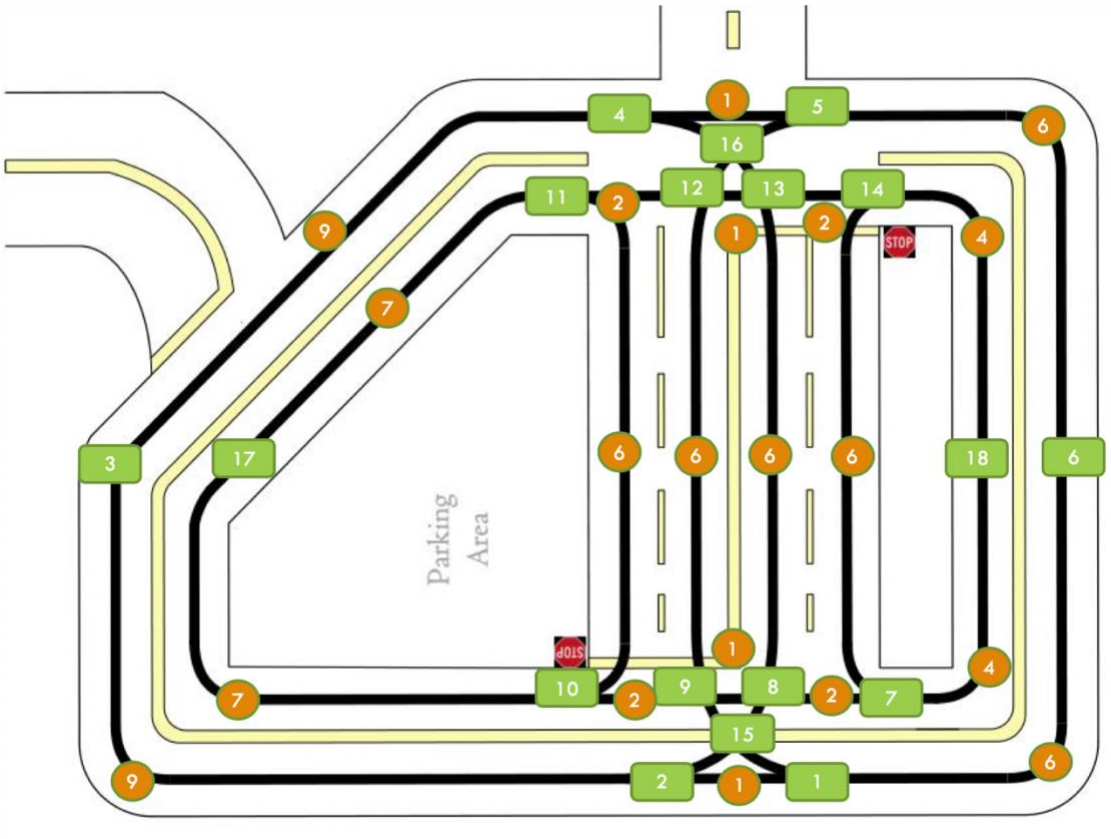


Figure 63. Assignment of nodes and weights for the map

The nodes and the weights were named as in figure 63. The values in green are the nodes and the ones in yellow are the weight values.

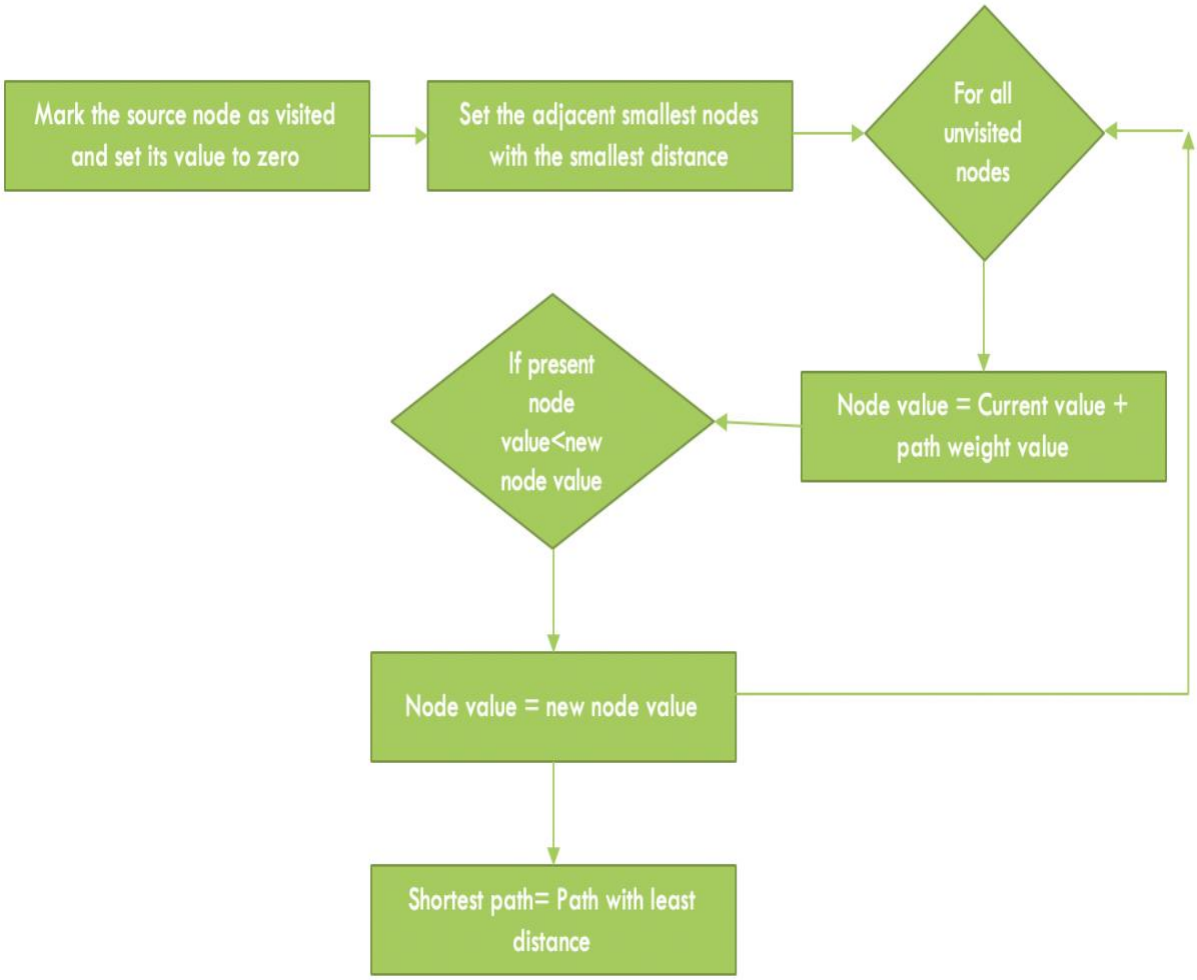


Figure 64. Flow of the Dijkstra's Algorithm

Dijkstra's algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.
Algorithm

1) Initialize the distance between the nodes and the array of all the visited and unvisited nodes
2) Assign the source node to the list of visited nodes and find the closest nodes to it
3) Assign weight values to each of the closest nodes.
4) Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance"
5) Pop the next vertex
6) If the vertex is visited, skip it
7) Apply the same algorithm till all nodes have been visited

With the Dijkstra's algorithm we were able to find the shortest path to go to a node and the EV3 followed the node. We were also able to find the shortest node, if some of the paths were not accessible. This was done considering that one of the paths would have construction work

going on and it would become important for the EV3 to find a better path to save time and reach its destination.

GUI (David)

Introduction

The most important characteristics of the GUI final version are:

- Modular and efficient programming.
- Local and remote control of the vehicles.
- Individual control and monitoring for each EV.
- Real time map information.
- Parking mode.
- Shortest path mode.

In this section of the report, an overview of these characteristics, and how they affect to the general behavior of the system, will be presented.
Additionally, at the end we will present a block diagram of how the GUI works, and a reference for each function.

Modular and efficient programming

The final version of the GUI is a program of approximately 2500 lines. When working with such a huge code, some special efforts must be made, in order to maintain its efficiency and the handleability. Throughout all the process, the team has been refining the code, making it more modular, and trying to reduce the execution time of each function.
For example, when connecting all four vehicles to the GUI, we observed a big delay. The program was really slow and even froze. After some study of the code, we saw that the function UpdateOptitrack was slowing the general performance. The original function had two loops, that had to be completely executed for every update of the map -0.1 seconds. After that, we designed a new version, much simpler, that only needed one loop. This change improved significantly the performance. (More information about this new function will be given in the code annex).
Additionally, in terms of performance, the use of timers had a big impact in the code. As explained in the last version, the use of this function allows the code to execute repetitive processes without using loops. This gives the code the ability to execute several tasks at the same time.
The implementation of the timers was used for the following functions:

- Update of sensor information (SensorTimer): The GUI will receive and record the values of the sensor. $\Delta\, SensorTimer = 1\, s$

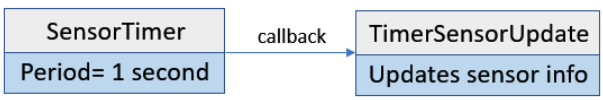


Figure 65. SensorTimer Diagram

- Update of Optitrack information and Map (OptitrackTimer): The GUI will receive the information related to the position and yaw of each EV and display it on the map. $\Delta\, OptitrackTimer = 0.1\ s$.

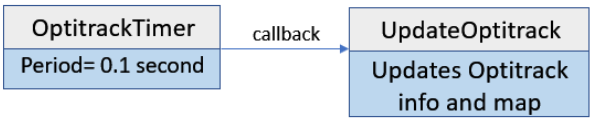


Figure 66. OptitrackTimer Diagram

- Parking mode (ParkingEntranceTimer). Each EV will have its own timer when the parking command is given. This timer will call PARKING, a state flow function that depends on the status of each robot. $\Delta\, ParkingEntranceTimer = 0.1$.

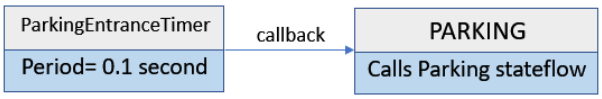


Figure 67. ParkingEntranceTimer Diagram

- Shortest path mode (ShortPathTimer). As only one EV will use the Short path mode, this timer is unique, unlike the Parking timer. It will call the function SHORT_PATH, that depends on the short path status of the vehicle. $\Delta\, ShortPathTimer = 0.05\ s$.

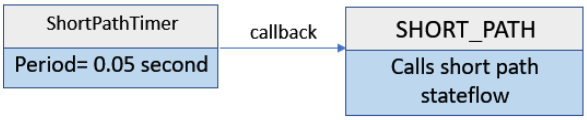


Figure 68. ShortPathTimer Diagram

Also, the team made a great effort in making the code more modular, making more use of classes and functions. In order to clarify the program, a lot of comments have been placed in most of the code's sections. For example, the next block diagram illustrates how the function ConnectEV works. Having the EV number and its parameters, one function is capable of connect any of the four EVs, reducing the amount of needed lines.

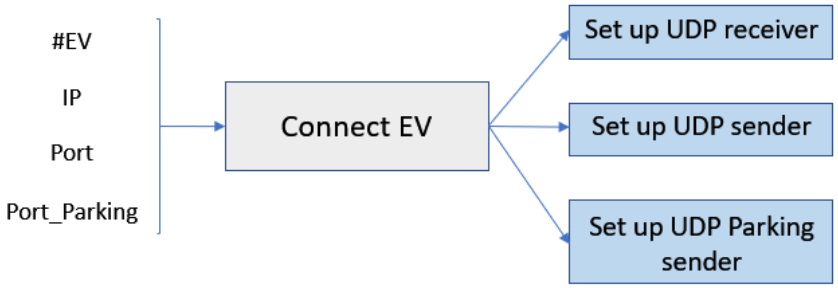


Figure 69. ConnectEV Diagram

Final version of the GUI

The final Graphical User interface has the following characteristics:

- The connection panel allows to connect any of the 4 main EVs. It also has a secondary tab, used for the two extra vehicles built for extra credit. In order to be more user friendly, the EV names have been replaced by Mario kart characters.
- The Mode allows the user command any of the 4 vehicles. Path tracking and platooning modes use Local control: The program burnt in the EV is in charge of the full control of the vehicle. Parking and Shortest path use remote control: The PC sends the commands periodically to the EV.
- Reading tab allow the user to monitor the information of the sensor of each robot, as well as the battery.
- Individual tabs for each of the EVs allow an individual control. With them, the trim, base speed and platooning distance of each vehicle can be independently changed. It also displays the C1 and C2 values for the 4 sensors in a line (EV 5) and allow the user to select the node destination in Shortest path mode.
- The log prints information about the execution of the code.
- Sounds. Each vehicle has its own sound when is connected. The start button and the parking have also special sounds.

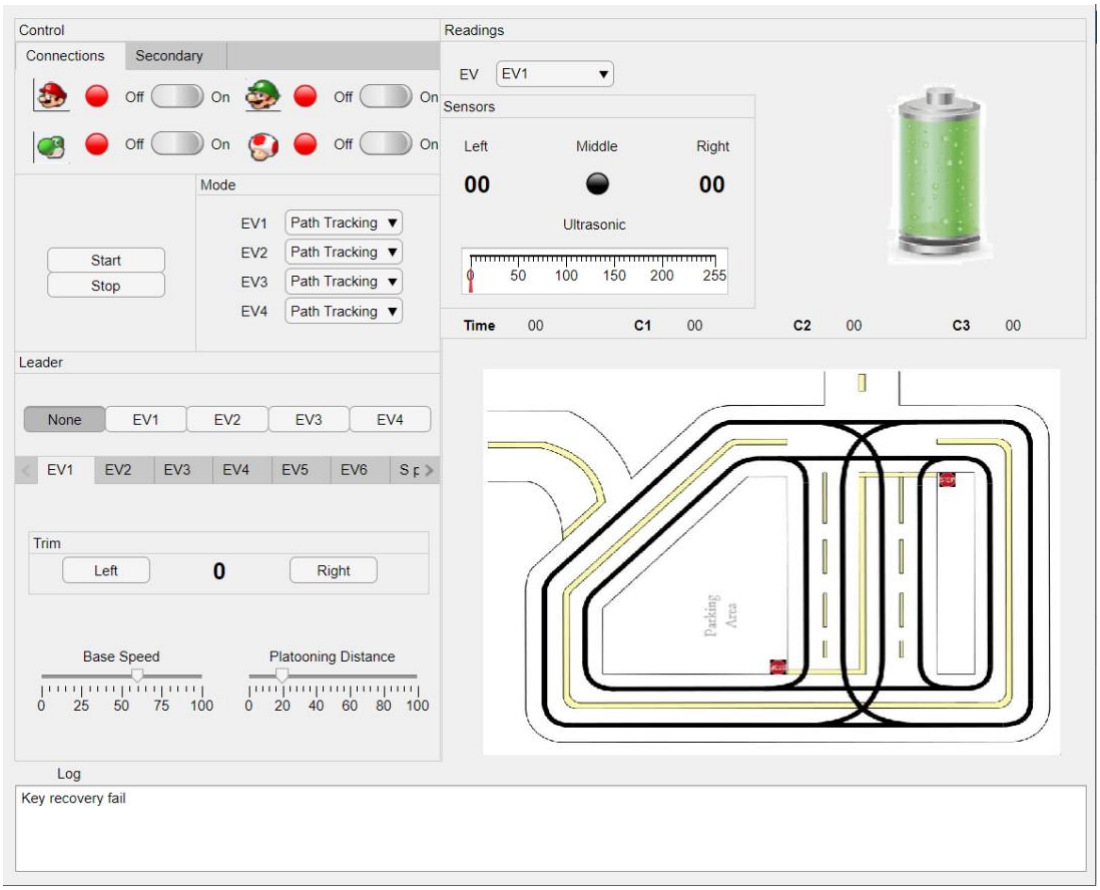


Figure 70. GUI

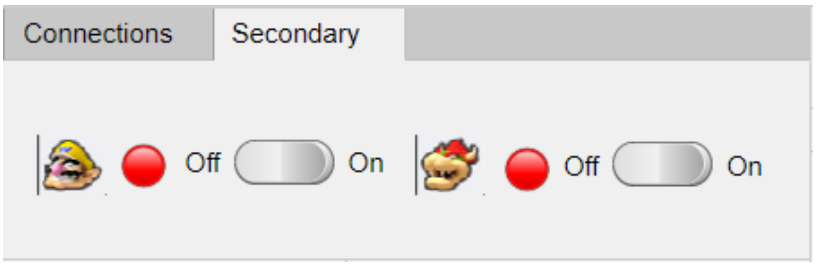


Figure 71. Secondary Connections tab

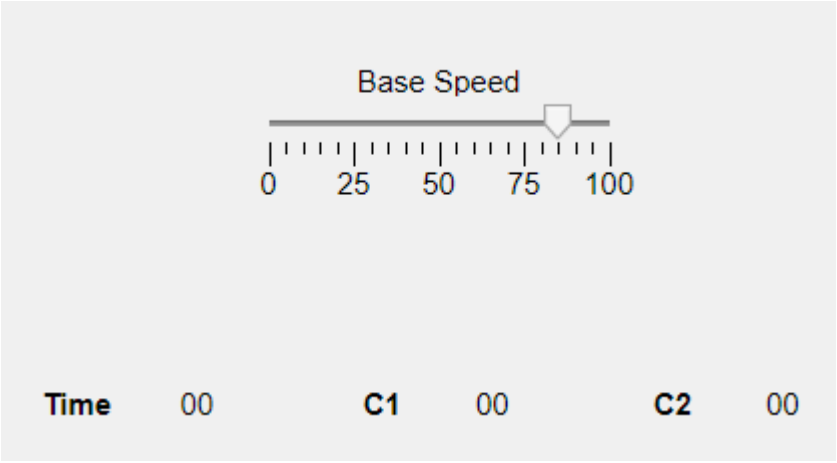


Figure 72. 4-sensors-in-a-line control

The design has been made trying to keep the GUI as simple and intuitive as possible.

Real time map information

The most intuitive way of giving the user information about the position of each robot is through a map.
With the Optitrack appears the issue of calibrating all the needed positions for the parking process. This should be done frequently, as the system is not very accurate, and the track is sometimes moved during the use.
Additionally, some areas of the track are blind spots of the Optitrack system. These spots were identified, in order to avoid using commands dependent of the position inside them. The next image shows these areas:

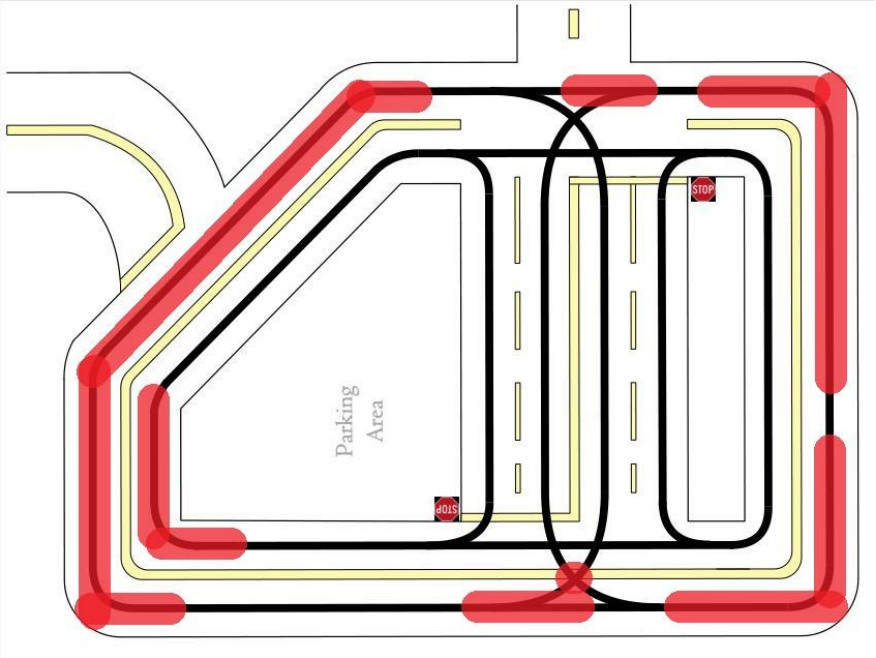


Figure 73. Optitrack blind spots

Parking mode

The Parking function is implemented with state flow. EntranceParkingTimer calls PARKING once every 0.1 seconds. The function then, according to the current state, sends the appropriate orders to the EV.
The next flowchart represents the different states inside PARKING function:

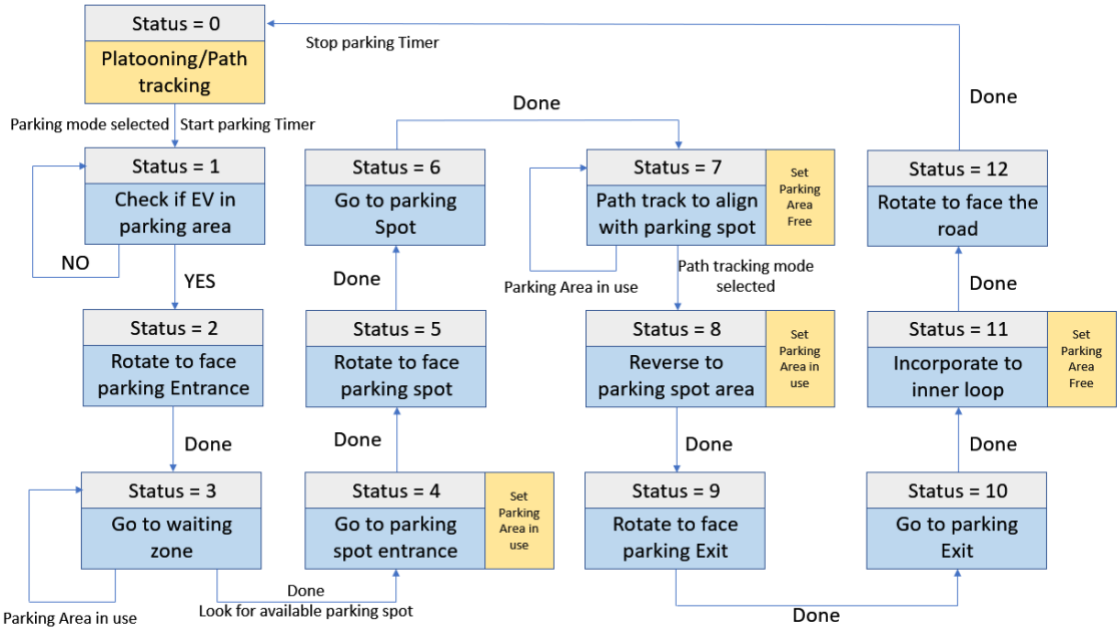


Figure 74. Parking Flow Chart

To execute each of the orders, the program makes use of four basic functions: EVStop, EVGoStraight, EVGoBack and EVRotate. All four of these functions sends a variation of the UDP packet to give parking orders. This packet has 4 components: The current yaw of the robot, the yaw to reach the target position, the base speed and the command to brake (0 brake, 1 no brake).
EVGoStraight and EVGoBack orders the EV to move in a rectilinear trajectory, with a fixed base speed (60 forwards, 30 backwards)

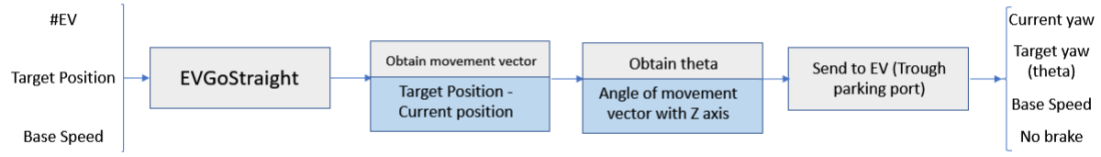


Figure 75. EVGoStraight Diagram

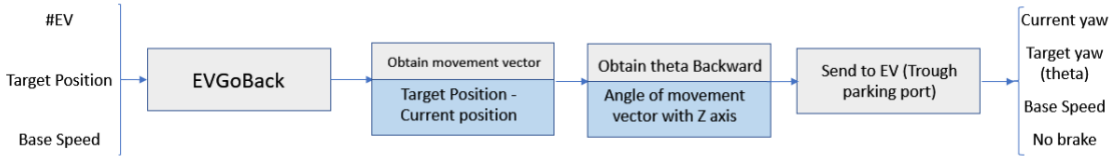


Figure 76. EVGoBack Diagram

The only difference between these two functions is how the final value of theta is obtained:

- The first step is to obtain the movement vector:

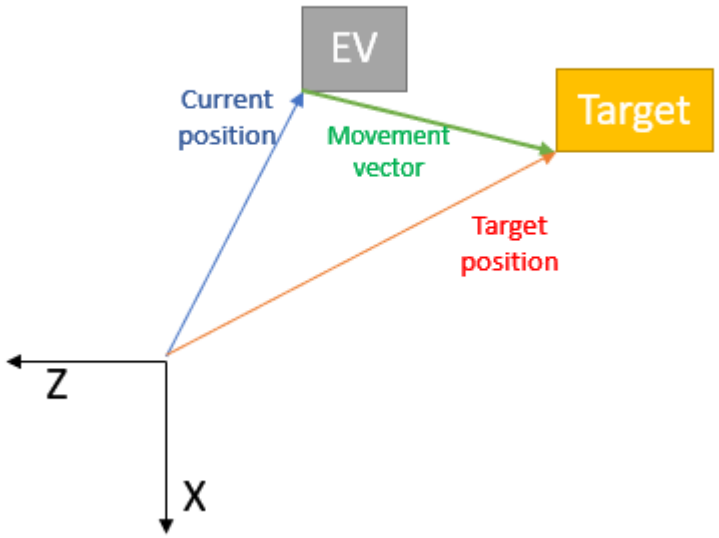


Figure 77. Movement vector

$$MovementVector = TargetPosition - CurrentPosition$$

- Then, the angle of the vector with respect to the Z axes is obtained:

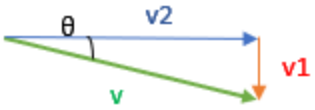


Figure 78. Theta angle

$$\theta = \left(\frac{v1}{v2}\right)$$

- Next, the program makes calculations to know in which quadrant the vector is, to accordingly assign the theta:

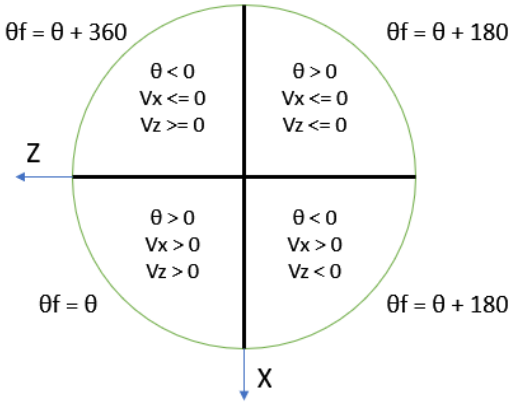


Figure 79. Final value of Theta

- Additionally, for the EVGoBack, the theta is reversed, which means that the final theta will be at 180º from the original theta. With this simple operation we are able to use the same Simulink code as EVGoStraight, without any additional modifications.

$$\theta_{fBack} = \theta_f + 180; If\ \theta_{fBack} \geq 360 \rightarrow \theta_{fBack} = \theta_{fBack} - 360$$

EVRotate orders the EV to rotate until it is facing the target yaw, without moving in the X and Z axis (Base speed is 0). Additionally, when this function is sent, the maximum speed of each wheel is set to 30. With this, we ensure a slower turn, avoiding overshoots.

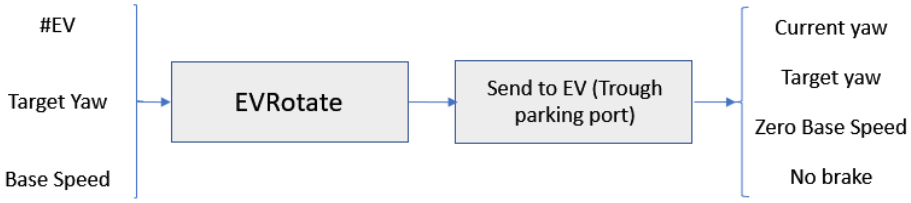


Figure 80. EVRotate Diagram

EVStop orders the EV to stop moving in all directions (No rotation, no forwards, no backwards). It also activates the brake, to stop as fast as possible.

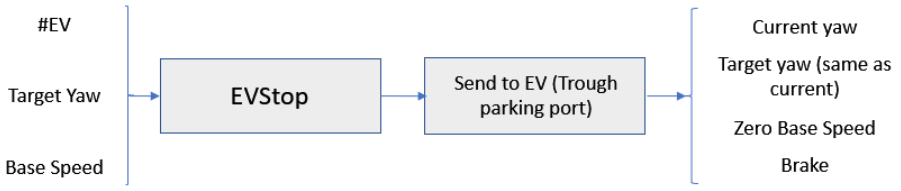


Figure 81. EVStop Diagram

With these four parameters, the Simulink code implemented in the EVs allows them to face the objectives set and to move from their current position to any target position. The main problem faced with the parking was that the information coming from the Optitrack was not consistent, and it varied for each of the EVs. The team had to set some tolerances for each point in order to make it work in a robust way. More details about how each of this functions works is given in Appendix A.

Shortest Path mode

The design of this function is very similar to the PARKING one. SHORT_PATH is called by ShortPathTimer, once every 0.05 seconds. The function takes as input the EV to be controlled and the current state and sends the corresponding orders to the robot. It is also implemented as a state flow.
The next flowchart represents the different states inside SHORT_PATH function:

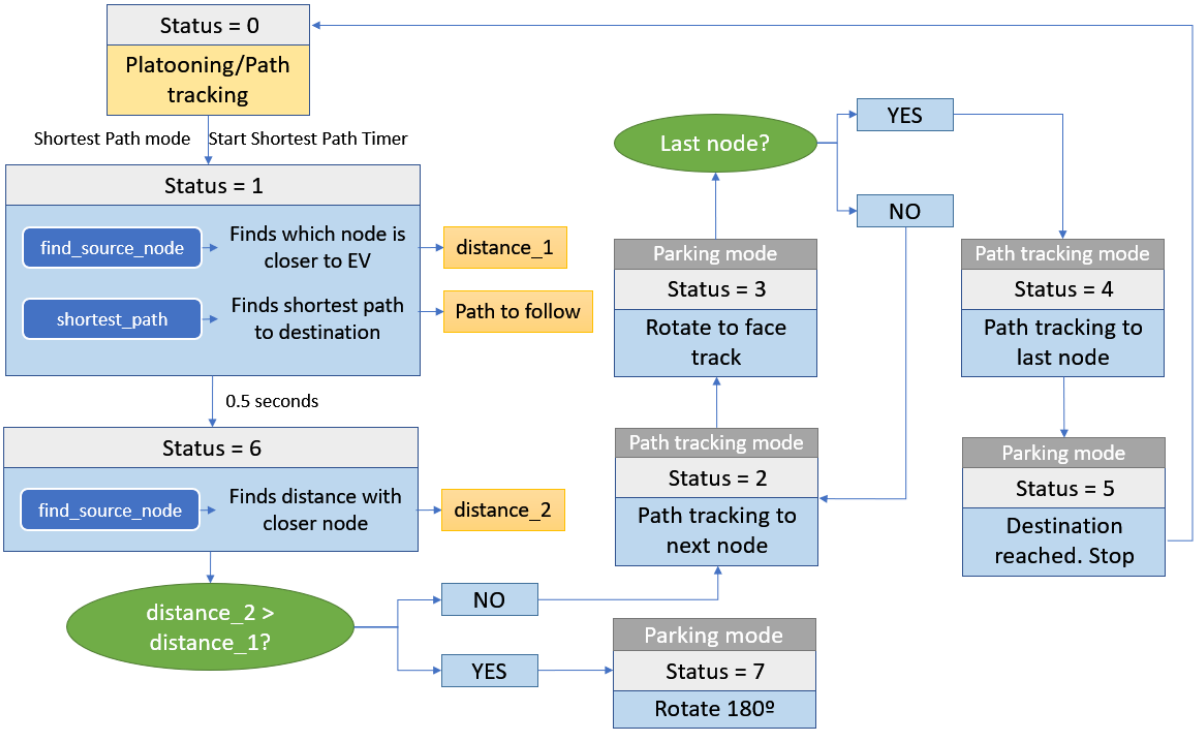


Figure 82. SHORT_PATH flowchart

It relies in the same functions as parking to control the vehicle in the nodes (EVRotate and EVStop).
It also uses two extra functions: find_source_node and shortest_path. These functions allow the program to calculate the closest node to the current position of the EV, if the robot is facing said node or going in the opposite direction, and the shortest sequence of nodes that the vehicle has to follow to reach its destination. From the second one it also obtains the rotations that have to be made on each node.
The algorithm used for these functions is explained in another section of this documentation.

GUI flow diagram

The next diagram shows the basic functioning of the GUI's code. Basically, it has three main functionalities: Optitrack, Sensor information and EV commands. Additionally, it has two special modes: Shortest path and Parking. These special modes flowcharts have already been explained before; therefore, they are not showed in this diagram

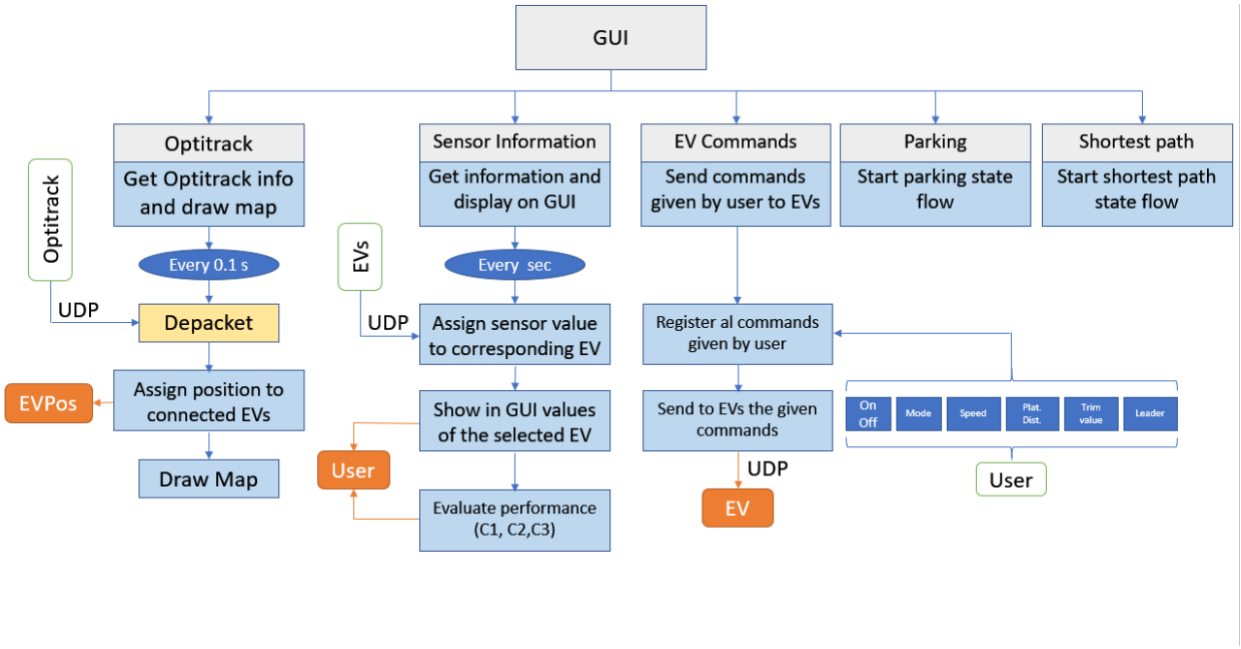


Figure 83. SHORT_PATH flowchart

GUI Analysis

Parking

We wanted to be sure about the reliability of the system. One of the most delicate processes in the program is the Parking process. To ensure its reliability the complete parking process was executed 30 times.

In all the tests, the parking only failed twice, due to Optitrack loosing track of the vehicle. The reliability of the system is around 93%.

Parking malfunctioning

During the testing process for exam 3 the team realized that, even tough the parking worked perfectly for one EV, when two were tried to park at the same time, the vehicles showed an erratic behavior, and the parking process could not be completed.
Our first thought was that the reason of this misbehavior could be the amount of data through the Wi-Fi. However, the results -explained in the Analysis of Network traffic- showed that the total amount of information was not excessive, and the bandwidth of the network should be more than enough to handle it. Therefore, it didn't seem the reason of our problem.
The next idea that we had was the execution time for the functions involved in the periodic processes – those called by the timers. We searched in those functions the ones that, in our perception, could have the longest execution times: UpdateOptitrack (receives information from Optitrack trough UDP, process the information, and assigns it to each EV) and DrawMap.
To analyze the execution time we used the tic-toc functions (measure time between the calls of each one). 200 executions of each function were tested each time, and the average execution time was calculated:

$$Execution\_Time = \frac{\sum_{i=1}^{200} Execution\_time\_i}{200}$$

To avoid deviations in the results due to other processes in the computer, the same procedure was applied 5 times. The final execution time was the average of these five tests.

- UpdateOptitrack:

- DrawMap:

As can be observed from the results, obviously the execution time increases when there are more EVs connected. The UpdateOptitrack could be dismissed as the main reason, because the longest time it needed was 1.24 ms, and the function is called every 100 ms.
Even tough DrawMap takes a bit longer and could be a bigger factor in the malfunctioning of the system, still did not explain the erratic behavior of the vehicles. The longest delay that it can inflict into the program was of 24 ms and taking in account that this function is called once every 100ms still didn’t fit as the source of the problem.

Once the execution times were also dismissed, we went back to analyzing the behavior of the EVs during the parking. It seemed to us that some kind of delay was appearing when two or more EVs were given the order to park simultaneously. Somehow, the new orders took more than 100ms to reach the vehicle.
So, we went back to the analysis of the UDP packets, but instead of looking for the total amount of information, we focused in the frequency the packets were sent.
For one EV parking, the Wireshark capture was:

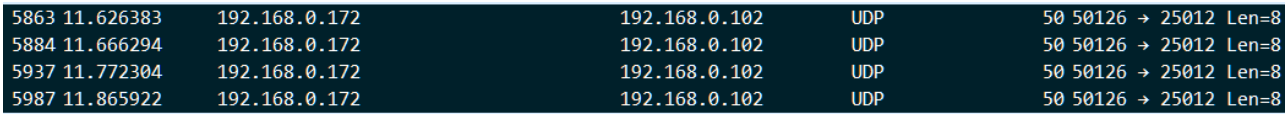


Figure 84. Wireshark capture of 1 EV parking

That seemed right, all the packets were sent every 0.1 seconds, even tough all for EVs were connected and displayed in the map. That meant that the delay given by the execution of these functions and the bandwidth used for sending sensor information should not be a problem.
For two EVs parking, however, the result was really different. This capture shows only the packets sent to one of the EVs:

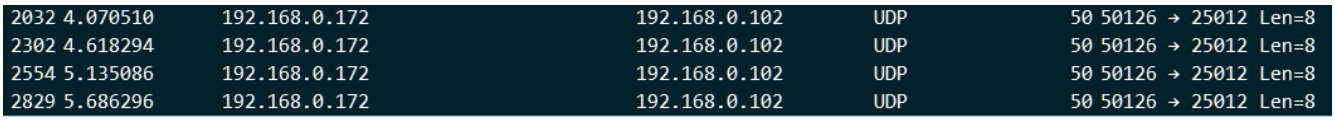


Figure 85. Wireshark capture of 2 EV parking

In here we can clearly appreciate a huge delay in the sending of the packets. Nothing analyzed up to now could be the reason of seconds of delay in the program. Also, it was really suspicious that the new frequency was almost exactly half a second, which made us dismiss problems with overloads in the UDP sender.

The final step was to take a close view of the functions involved in the sending of the UDP packets. Finally, we found that the source of the problem was the EVStop function.

```
function EVStop(app,EV) %Sends order to stop the EV, both rotation and movement
    parkingsend(1) = int16(app.EV(EV).yaw); %Current yaw
    parkingsend(2) = int16(app.EV(EV).yaw); %Target yaw
    parkingsend(3) = int16(0); %Base Speed
    parkingsend(4) = int16(0); %No brake

    pause(0.5)

    app.EV(EV).ParkingSend(parkingsend);
```

And there it was the answer, clear and obvious. Each time the PARKING function called EVStop, the WHOLE program was paused for half a second. Initially, this pause was used to make sure that the EV would have time to brake, but we didn't realize at the moment that it would stop the rest of the functions. The solution was, therefore, trivial: simply erase that line. After that, the parking function worked perfectly.

Even tough the problem was something simple and “obvious”, the extension of the code, and the complexity of the system made finding that error very difficult. Several systems are implemented in our code (Optitrack, sensor information, modes, EV commands, etc). However, trough analysis and the full understanding of every aspect of the program we were able to dismiss possible sources one by one, finally finding and fixing the problem.

Conclusions (Davis)

The EV3s performed well during the life of the project. The analytical discussions suggest that some hardware systems may need to be upgraded to enhance the applications of the project. It is hoped that future ECE 756 students will be able to easily recreate the current setup as well as easily expand upon the functionalities of the project. Given how the programs for the EV3s and GUI are written, future students should be able to quickly understand the code and use it as a base to develop their own new code.

Appendix A (GUI Code) (David)

In this appendix the used functions are included and explained.

Function reference

ConnectEV	
Description	Creates UDP sender and receiver objects, allowing communication with EV.
Called by	Switch_1ValueChanged, Switch_2ValueChanged, Switch_3ValueChanged, Switch_4ValueChanged
Inputs	#EV, IP, Port, ParkingPort
Outputs	

DisconnectEV	
Description	Disconnects UDP sender and receiver objects, ends communication with EV.
Called by	Switch_1ValueChanged, Switch_2ValueChanged, Switch_3ValueChanged, Switch_4ValueChanged
Inputs	#EV
Outputs	

updateLog	
Description	Updates the log area with the given message. Allows the log to print more than one message.
Called by	All functions that print information or errors in the Log
Inputs	Message

Outputs	LogTextArea.Value

BatteryLevel	
Description	Changes battery image according to current charge.
Called by	UpdateSensor
Inputs	Battery level
Outputs	Displayed image

timerSensorUpdate	
Description	Callback of SensorTimer once per second. Receives EV information. Triggers UpdateSensor
Called by	SensorTimer
Inputs	
Outputs	data received

UpdateSensor	
Description	Registers and displays EV information
Called by	timerSensorUpdate
Inputs	#EV, data received, vector index
Outputs	Information of sensors to be displayed

SendMessage

Description	Sends UDP packet to Evs
Called by	Every function or action that modifies behavior of Evs
Inputs	#EV, status, mode, base speed, platooning distance, trim value, leader
Outputs	Information of sensors to be displayed

LeftTrimValueChange	
Description	Updates trim value. Subtracts 1 to the actual value
Called by	Trim buttons in GUI
Inputs	#EV, actual Trim Value
Outputs	New Trim Value

RightTrimValueChange	
Description	Updates trim value. Adds 1 to the actual value
Called by	Trim buttons in GUI
Inputs	#EV, actual Trim Value
Outputs	New Trim Value

UpdateOptitrack	
Description	Receives and assigns position and yaw (first calculate from quaternion to degrees) data to Evs. Calls UpdateMap
Called by	Optitrack timer. Once every 0.1 seconds
Inputs	#EV, actual Trim Value
Outputs	New Trim Value

Depacket	
Description	Recieves UDP packet from Optitrack, translate it to useful data.
Called by	UpdateOptitrack
Inputs	UDP array. Values from 0 to 255
Outputs	X, Z position, yaw for all tracked Evs

UpdateMap	
Description	Takes EVs position and draws them on the Map.

Called by	UpdateOptitrack
Inputs	EVs position
Outputs	Map

EVGoStraight	
Description	Gives order to the EV in parking mode to go straight towards the target position, with a set base speed.
Called by	PARKING function
Inputs	#EV, target position, Base speed
Outputs	Current yaw of EV, target yaw, base speed, No brake

EVGoBack	
Description	Gives order to the EV in parking mode to go backwards towards the target position, with a set base speed.
Called by	PARKING function
Inputs	#EV, target position, Base speed
Outputs	Current yaw of EV, target yaw, base speed, No brake

EVRotate	
Description	Gives order to the EV in parking mode to rotate to face the target orientation
Called by	PARKING function
Inputs	#EV, target yaw
Outputs	Current yaw of EV, target yaw, zero base speed, No brake

EVStop	
Description	Gives order to the EV in parking mode to stop moving

Called by	PARKING function
Inputs	#EV
Outputs	Current yaw of EV, zero base speed, Brake

CheckParkingInbox	
Description	Checks if the EV is in the entry area for parking
Called by	PARKING function
Inputs	#EV, ev position
Outputs	Parking status

PARKING	
Description	State flow for parking the Evs
Called by	ParkingEntranceTimer. Once every 0.1 seconds
Inputs	#EV, parking status
Outputs	Commands to the EV. Mode, parking status

SPcheckbox	
Description	Checks if an EV is in the given position
Called by	SHORT_PATH function
Inputs	#EV, x and z coordinates of the point to check
Outputs	Short Path status

SHORT_PATH	
Description	State flow for shortest path function

Called by	ShortPathTimer. Once every 0.05 seconds
Inputs	#EV, short path status
Outputs	Commands to the EV. Mode, parking status

find_source_node	
Description	Finds the closest node to the position of the EV. Takes in account the loop in which the EV is (inner/outer)
Called by	SHORT_PATH
Inputs	EV position
Outputs	Closest node, distance to node

shortest_path	
Description	Finds the sequence of nodes that ensures shortest path.
Called by	SHORT_PATH
Inputs	Source, destination
Outputs	Node path, angle to turn in each node

ChangeMode	
Description	Change the EVs modes of operation
Called by	GUI command given by User, PARKING
Inputs	#EV, previous mode, new mode, parking status
Outputs	Mode, parking status

Characters	
Description	Draws the images of each character in the connections panel

Called by	StartupFcn
Inputs	Character, image
Outputs	-

PlaySound	
Description	Plays a given audio
Called by	Parking, EV connection switchs
Inputs	Audio file
Outputs	-

Commented code

The EV class has been created. Its objects are:

```
%Object
EV %array of EVs with its own parameters
  Receiver
  Sender
  ParkingSend
  dataindex 1 // index of the array for placing information
  mode_operation 0 // path tracking(0)/Platooning(1)/parking(2) for EVs. Default mode Path trackig
  Trim_value 10  // Allows the robot to change path in an intersection. Sent value 10, display value 0
  position
  yaw
  ParkingEntranceTimer //Controls the PARKING function
  Status //Status of parking flow
  BaseSpeed
  PlatooningDistance
  parkingsound //Flag that ensures parking sound to be played only once each time
```

Timers Callbacks:

Updating Sensors:

```
function timerSensorUpdate(app) %callback of SensorTimer timer
    %Every second this function recieves UDP packet and sets
    %Sensor Values(UpdateSensor) and Timestamp. For each of the connected EVs
    for i = 1:1:4
        if app.EVConnected(i) == 1
            %receive data from EV
            dataReceived = app.EV(i).Receiver();
            %increase the index of the vectors. for i=1 values will be 0
            app.EV(i).dataindex = app.EV(i).dataindex + 1;
            %Call fnc that updates sensor values
            UpdateSensor(app, i,dataReceived, app.EV(i).dataindex);
            %update timestamp. Accumulated time value + time between SensorTimer ticks
            app.timestamp(i, app.EV(i).dataindex) = app.SensorTimer.InstantPeriod + app.timestamp(i, app.EV(i).dataindex-1);
            %avoid NaN in the first call (supposing a exact time of 1 sec)
            if app.EV(i).dataindex == 2
                 app.timestamp(i,2) = 1;
            end
        end
    end
```

This function Calls UpdateSensor:

```
function UpdateSensor(app, ev, dataReceived, i)
        %receives vector with UDP Data and the vector index
        %UDP packet values: left sensor, color sensor, right sensor, US, left encoder, right encoder, battery level
    try

        %assign values to variables

        left_sensor = dataReceived(1);
        middle_sensor = dataReceived(2);
        right_sensor = dataReceived(3);
        US = dataReceived(4);
          left_encoder = dataReceived(5);
          right_encoder = dataReceived(6);
        battery = dataReceived(7);

        %update tags. Only if this EV is the one selected for display
        if ev == app.EV_readings_display
            app.LeftSensorValue.Text= int2str(left_sensor);
            app.RightSensorValue.Text= int2str(right_sensor);
            app.UltrasonicGauge.Value = US;
            BatteryLevel(app,battery);


                switch middle_sensor
                    case 1 % black
                    app.MidLamp.Color = 'k';
                    case 6 % white
                    app.MidLamp.Color = 'w';
                    case 3 % blue
                    app.MidLamp.Color = 'b';
                    case 5 % red
                    app.MidLamp.Color = 'r';
                    otherwise
                    app.MidLamp.Color = 'k';
                end

              app.TimeTag.Text = int2str(app.timestamp(ev,end));
        end
```

```
     %Update vectors for evaluation

     app.RLI(ev,i) = left_sensor; %sending left sensor value
     app.Ultrasonic(ev,i) = US;



     catch
         updateLog(app,'Error in sensor communication.');
     end
end
```

Which also calls Battery Level:

```
function BatteryLevel(app,battery_level) %Receives the battery percentage and changes the img according to that
    if battery_level >= 90
        %bat full
        imshow('full.jpg','Parent',app.battery_image);

    elseif 65 <= battery_level && battery_level < 90
        %bat 75
        imshow('75.png','Parent',app.battery_image);

    elseif 35 <= battery_level && battery_level < 65
        %bat 50
        imshow('50.png','Parent',app.battery_image);

    elseif 15 <= battery_level && battery_level <35
        %bat 25
        imshow('25.png','Parent',app.battery_image);
    elseif battery_level < 15
        %bat empty
        imshow('empty.png','Parent',app.battery_image);
    end
end
```

Updating Optitrack information

```
function UpdateOptitrack(app)
    %receives information from optitrack and updates Map
    try
        DataOptitrack = app.Optitrack(); %Receive trhough UDP communication

        [position,orientation] = Depacket(DataOptitrack);

        j=1;
    catch
    end
    for Ev=1:4
        try
            if app.EVConnected(Ev) == 1 && position(j,1) ~= 0
                app.EV(Ev).position(1) = position(j,1);
                app.EV(Ev).position(2) = position(j,3);
                %Calculate yaw in degrees -> 0-360º
                qx = orientation(j,1);
                qy= orientation(j,2);
                qz= orientation(j,3);
                qw = orientation(j,4);

                yaw = rad2deg(atan2(2.0*(qy*qw + qx*qz), 1 - 2*(qy*qy + qz*qz)));

                if yaw <=0
                    app.EV(Ev).yaw = 360+yaw;
                else
                    app.EV(Ev).yaw = yaw;
                end

                j=j+1;
            end
        catch
        end
    end
    UpdateMap(app)

end
```

Which calls UpdateMap:

```
function UpdateMap(app) %Updates map with the new information form optitrack

    %Redraws map
    imagesc( [1.3,-3.7],[-1,2.5],app.MapFig,'Parent',app.UIAxes);


    hold(app.UIAxes,'on')
    for ev = 1:4 %Selects correct image
        if app.EVConnected(ev) == 1

            switch ev
                case 1
                    figure = app.Ev1Fig;
                case 2
                    figure = app.Ev2Fig;
                case 3
                    figure = app.Ev3Fig;
                case 4
                    figure = app.Ev4Fig;
            end

            %Draws EV
            imagesc( [app.EV(ev).position(2),app.EV(ev).position(2)-0.2],[app.EV(ev).position(1),app.EV(ev).position(1)+0.2],figure,'Parent',app.UIAxes);

        end
    end
    hold(app.UIAxes,'off')

end
```

And Depacket:

```
function [position, orientation] = Depacket(datarec)
    output=double(datarec);


    umv=output(25)*12+29;
    i=0;
    while output(umv) == 0
        umv=output(25)*12+34+i;
        i=i+1;
    end
    rbc=output(umv)*12+4+umv;
    rbc_ev = zeros(1,output(rbc));

    rbc_ev(1)=rbc + 4;
    ev_position = 0; single(ev_position);
    ev_orientation = 0; single(ev_orientation);
    ev_pos = zeros(output(rbc),12);
    ev_ori = zeros(output(rbc), 16);
    ev_marker = zeros(1,output(rbc));
    ev_position = zeros(output(rbc),3);
    ev_orientation = zeros(output(rbc),4);
```

```
 for j=1:output(rbc)
     for i=1:12
         ev_pos(j,i)=output(rbc_ev(j)+3+i);
     end
     for i=1:16
         ev_ori(j,i)=output(rbc_ev(j)+15+i);
     end
     ev_marker(j) = rbc_ev(j)+32;
     rbc_ev(j+1) = output(ev_marker(j))*16+20+ev_marker(j);
 end

 for i=1:output(rbc)
     for j=1:3
         for w = 1:4
             temp(w)=(ev_pos(i,(j-1)*4+w));


         end
          abc=uint8(temp);
         Y=typecast(abc,'single');
         ev_position(i,j)=Y;
     end
     for j=1:4
         for w = 1:4
             temp(w)=(ev_ori(i,(j-1)*4+w));



             end
             abc=uint8(temp);
             Y=typecast(abc,'single');
             ev_orientation(i,j)=Y;
         end
    end
    position = ev_position;
    orientation = ev_orientation;
end
```

Parking:

Movement functions. Summoned by PARKING to control the EVs:

```
function EVGoStraight(app,EV,target_pos,BaseSpeed) %Ev goes straight to the target point

    current_pos = [app.EV(EV).position(1) app.EV(EV).position(2)]; %current position

    v= target_pos-current_pos; %vector from current position to target position
    theta= rad2deg(atan(v(1)/v(2))); %Angle of the vector in respect to Z axis (origen)
            if theta > 0
                if v(2) <= 0 && v(1) <= 0
                    theta_f = 180+theta;
                else
                    theta_f = theta;
                end
            else
                if v(1)<=0 && v(2) >= 0
                    theta_f = 360+theta;
                else
                    theta_f = 180+theta;
                end
            end
     parkingsend(1) = double(app.EV(EV).yaw); %Current yaw
     parkingsend(2) = double(theta_f); %objective yaw
     parkingsend(3) = double(BaseSpeed); %base speed
     parkingsend(4) = double(1); %No brake

     app.EV(EV).ParkingSend(parkingsend); %Send trough UDP
end

function EVGoBack(app,EV,target_pos,BaseSpeed) %Ev goes straight to the target point

    current_pos = [app.EV(EV).position(1) app.EV(EV).position(2)];

    v= target_pos-current_pos; %vector from current position to target position
    theta= rad2deg(atan(v(1)/v(2))); %Angle of the vector in respect to Z axis (origen)
            if theta > 0
                if v(2) <= 0 && v(1) <= 0
                    theta_f = 180+theta;
                else
                    theta_f = theta;
                end
            else
                if v(1)<=0 && v(2) >= 0
                    theta_f = 360+theta;
                else
                    theta_f = 180+theta;
                end
            end
            theta_f = theta_f +180;
            if  theta_f >= 360
                theta_f = theta_f - 360;
            end
     parkingsend(1) = double(app.EV(EV).yaw); %Current yaw
     parkingsend(2) = double(theta_f); %Target yaw
     parkingsend(3) = double(BaseSpeed); %Base speed
     parkingsend(4) = double(1); %No brake

     app.EV(EV).ParkingSend(parkingsend); %Send trough UDP
end
```

```
function EVRotate(app,EV,target_yaw) %EV turns in spot to the target yaw

    parkingsend(1) = double(app.EV(EV).yaw); %Current yaw
    parkingsend(2) = double(target_yaw); %Target yaw
    parkingsend(3) = double(0); %Base Speed
    parkingsend(4) = double(1); %No brake
    app.EV(EV).ParkingSend(parkingsend);


end

function EVStop(app,EV) %Sends order to stop the EV, both rotation and movement
    parkingsend(1) = double(app.EV(EV).yaw); %Current yaw
    parkingsend(2) = double(app.EV(EV).yaw); %Target yaw
    parkingsend(3) = double(0); %Base Speed
    parkingsend(4) = double(0); %No brake

    app.EV(EV).ParkingSend(parkingsend);

end
```

PARKING function. Switch with the input “status” of each EV.

```
function PARKING(app,EV,status) %Parking flow. Depending on the "status" of the EV, different
    % orders are executed

    switch status

        case 0 %Not in parking mode

            stop(app.EV(EV).ParkingEntraceTimer); %Stop the parking timer

            SendMessage(app,EV);

        case 1 %Check if EV is in the entry point
            app.EV(EV).Status = CheckParkingInbox(app,EV);

        case 2 %In parking Entrance. Rotate until facing the parking zone

              message = sprintf('Status EV%d 2',EV);
              updateLog(app,message);

            if app.EV(EV).yaw >90+app.AngleTolerance || app.EV(EV).yaw <90-app.AngleTolerance
                EVRotate(app,EV,90); %If the target angle is not reached, keep rotating.
            else
                app.EV(EV).Status = 3; %If the target angle is reached, go to next stage.
                EVStop(app,EV);
            end

case 3  %Go to the waiting point. The EV will stop if any other EV is parking
    %or deparking

      message = sprintf('Status EV%d 3',EV);
      updateLog(app,message);

    if app.EV(EV).position(1) > app.WaitingPoint(1)-app.BoxTolerance && app.EV(EV).position(1) < app.WaitingPoint(1)+app.BoxTolerance && app.EV(EV).position(2) > app.Wa
        %If the Waiting point is reached, the EV stops
        EVStop(app,EV);

        if app.SomeoneIsParking == 0 %If no one is parking, looks for the first available parking spot
            for i = 1:3
                if app.ParkingSpots(i) == 0 %sets the first available spot as parking for this EV
                    app.ParkingSpots(i) = 1;
                    app.EV(EV).ParkingSpot = i;
                    app.EV(EV).EntrancePoint = [app.EntrancePointsParking(i,1) app.EntrancePointsParking(i,2)];
                    app.EV(EV).ParkingRamp = [app.RampParkings(i,1) app.RampParkings(i,2)];
                    break
                end
            end
            app.EV(EV).Status = 4; %As no one is parking, the EV goes to next stage
        end

    else
        %If waiting point not  reached, keep going straight
        EVGoStraight(app,EV,app.WaitingPoint, app.ForwardBaseSpeed)

    end

  case 4 %Goes to the perpendicular point of the assigned parking spot

       message = sprintf('Status EV%d 4',EV);
       updateLog(app,message);

      app.SomeoneIsParking = 1; %Update flag. Now this EV is parking


      if app.EV(EV).position(1) > app.EV(EV).EntrancePoint(1)-app.BoxTolerance && app.EV(EV).position(1) < app.EV(EV).EntrancePoint(1)+app.BoxTolerance && app.EV(EV
          app.EV(EV).Status = 5; %If the Entrance point is reached, go to next stage
          EVStop(app,EV);

      else
          %If entrance point is not reached, keep going straight
          EVGoStraight(app,EV,app.EV(EV).EntrancePoint, app.ForwardBaseSpeed)
      end

  case 5 %Rotate to face parking spot
       message = sprintf('Status EV%d 5',EV);
       updateLog(app,message);

      if app.EV(EV).yaw <app.AngleTolerance && app.EV(EV).yaw >=0   || app.EV(EV).yaw >360-app.AngleTolerance && app.EV(EV).yaw <360
          app.EV(EV).Status = 6; %If angle is reached, go to next stage
          EVStop(app,EV);
      else
          EVRotate(app,EV,0); %Otherwise keep rotating.
      end
```

```
case 6 %Go straight until Parking spot

      message = sprintf('Status EV%d 6',EV);
      updateLog(app,message);
    if app.EV(EV).position(1) > app.EV(EV).ParkingRamp(1)-app.BoxTolerance && app.EV(EV).position(1) < app.EV(EV).ParkingRamp(1)+ap
        app.EV(EV).Status = 7; %If spot is reached, go to next stage
        EVStop(app,EV);
    else
        %Otherwise keep going straight
        EVGoStraight(app,EV,app.EV(EV).ParkingRamp, app.ForwardBaseSpeed)
    end
case 7 %Enter in path tracking mode, to align with parking spot

      message = sprintf('Status EV%d 7',EV);
      updateLog(app,message);

    app.SomeoneIsParking = 0; %Out of parking area. This EV is no longer parking

    try
        app.EV(EV).mode_operation = 0; %change mode to path tracking
        SendMessage(app,EV);
          message = sprintf('Mode path tracking, mode %d',app.EV(EV).mode_operation);
          updateLog(app,message);
    catch
         message = sprintf('Mode not path tracking, mode %d',app.EV(EV).mode_operation);
         updateLog(app,message);
    end
```

```
case 8 %Order to depark sent -> Mode changed from parking to path tracking

      message = sprintf('Status EV%d 8',EV);
      updateLog(app,message);

    if app.SomeoneIsParking == 1 && app.EV(EV).mode_operation == 0
        EVStop(app,EV) %Wait until parking area is clear
    else
        app.EV(EV).mode_operation = 2; %Return to parking mode
        SendMessage(app,EV);
        app.SomeoneIsParking = 1; %Update flag, now this EV is parking


        if app.EV(EV).position(1) > app.EV(EV).EntrancePoint(1)-app.BoxTolerance && app.EV(EV).position(1) < app.EV(EV).EntrancePoint(1)+
            app.EV(EV).Status = 9; %if target point is reached go to next stage
            EVStop(app,EV);
              message = sprintf('Reached');
              updateLog(app,message);
        else
              message = sprintf('Going Backwards to Entrance Point');
              updateLog(app,message);
            %Otherwise reverse to entrance point
            EVGoBack(app,EV,app.EV(EV).EntrancePoint, -30)
        end
    end

case 9 %rotate to face exit point

      message = sprintf('Status EV%d 9',EV);
      updateLog(app,message);

    if app.EV(EV).yaw >90+app.AngleTolerance || app.EV(EV).yaw <90-app.AngleTolerance
        EVRotate(app,EV,90); %If angle is not reached keep rotating
    else
        app.EV(EV).Status = 10; %Angle reached. Go to next stage
        EVStop(app,EV);
    end

case 10 %Go to parking exit

      message = sprintf('Status EV%d 10',EV);
      updateLog(app,message);

    if app.EV(EV).position(1) > app.ParkingExit(1)-app.BoxTolerance && app.EV(EV).position(1) < app.ParkingExit(1)+app.BoxTolerance && app.EV(EV).p
        app.EV(EV).Status = 11; %Exit reached. Go to next stage
        EVStop(app,EV);
        app.SomeoneIsParking = 0; %Parking area cleared

        app.ParkingSpots(app.EV(EV).ParkingSpot)=0; %parking spot used cleared
    else
        %otherwise keep going straight
        EVGoStraight(app,EV,app.ParkingExit, app.ForwardBaseSpeed)
    end


case 11 %Incorporate to inner loop
      message = sprintf('Status EV%d 11',EV);
      updateLog(app,message);
    if app.EV(EV).position(1) > app.ExitLane(1)-app.BoxTolerance && app.EV(EV).position(1) < app.ExitLane(1)+app.BoxTolerance && app.EV(EV).posi
        app.EV(EV).Status = 12; %target reached, next stage
        EVStop(app,EV);
    else
        %otherwise keep going
        EVGoStraight(app,EV,app.ExitLane, app.ForwardBaseSpeed)
    end
case 12 %Rotate to face road

      message = sprintf('Status EV%d 12',EV);
      updateLog(app,message);

    if  app.EV(EV).yaw <app.AngleTolerance && app.EV(EV).yaw >=0   || app.EV(EV).yaw >360-app.AngleTolerance && app.EV(EV).yaw <360
        app.EV(EV).Status = 0; %Angle reached, END OF PARKING
        EVStop(app,EV);
        app.EV(EV).mode_operation = 0; %Mode set to path tracking
        SendMessage(app,EV)
        stop(app.EV(EV).ParkingEntraceTimer); %STOPS parking timer
    else
        EVRotate(app,EV,0); %otherwise keep rotating.
    end
```

Case 1 calls CheckParkingInbox

```
function [status] = CheckParkingInbox(app,ev) %Checks if the EV is in the entry point to parking

        if app.EV(ev).position(2) >= app.parking_box_z(1) && app.EV(ev).position(2) <= app.parking_box_z(2) && app.EV(ev).position(1) >= ap
            status = 2; %Status 2 will go to the next STAGE in PARKING
            app.EV(ev).mode_operation = 2; %If it ins inside, mode changes to parking
            try
            message = sprintf('Mode EV%d Parking',ev);
            EVStop(app,ev);
            updateLog(app,message);

            SendMessage(app,ev);
            catch
                message = sprintf('Error');
                updateLog(app,message);
            end



        else
            status = 1; % Status 1 will maintain current stage in PARKING


        end

end
```

The lines that appear cut are just coordinates x and z.

Shortest Path functions:

```
function SHORT_PATH(app,EV,status)
      message = sprintf('Status sp %d ',app.SPstatus);
      updateLog(app,message);

    switch status
        case 1 %find shortest path
            %find source node. Node closest to current EV position
            [source, app.distance_1] = find_source_node(app.EV(EV).position(1),app.EV(EV).position(2));

            %Retreive destination from the GUI
            destination = str2num(app.DestinationDropDown.Value);

            %Find sequence of nodes and angle to rotate on each node
            [app.node_path,app.angle_path]=shortest_path(source,destination);

            %flags
            app.NumNodes = length(app.node_path);
            app.NodeFlag = 1;

            %Goes to Status 6
            app.SPstatus = 6;

        case 2 %Path tracking to point
            app.EV(EV).mode_operation = 0; %path tracking mode
            SendMessage(app,EV);
            %checks if the EV reaches its destination (Next node)
            app.SPstatus = SPcheckbox(app,EV,app.CoordinatesNodes(app.node_path(app.NodeFlag),1),app.CoordinatesNodes(a
```

```
case 3 %Rotate if needed
    %Remote control mode
    app.EV(EV).mode_operation = 2;
    SendMessage(app,EV);

    if app.angle_path(app.NodeFlag) == 0 %if the angle to face is 0 degrees
        %Check if EV has correct yaw
        if app.EV(EV).yaw <app.AngleTolerance && app.EV(EV).yaw >=0   || app.EV(EV).yaw >360-app.AngleTolerance && app.
            EVStop(app,EV);

            if app.NodeFlag == app.NumNodes-1
                app.SPstatus = 4; %If next node is the last node, go to status 4
            else
                app.SPstatus = 2; %If not, go again to status 2
                app.NodeFlag = app.NodeFlag + 1;
            end
        else %If the target yaw is not reached, keep rotating
            EVRotate(app,EV, app.angle_path(app.NodeFlag));
        end
    elseif app.angle_path(app.NodeFlag) == 500 %arbitrary value. In case the path is only of 1 node.
        app.SPstatus = 4;
    else %if the angle to face is not 0
        %Check if EV has correct yaw
        if app.EV(EV).yaw >app.angle_path(app.NodeFlag)+app.AngleTolerance || app.EV(EV).yaw <app.angle_path(app.NodeF
            %If the target yaw is not reached, keep rotating
            EVRotate(app,EV, app.angle_path(app.NodeFlag));

        else
            EVStop(app,EV);
            if app.NodeFlag == app.NumNodes-1
                app.SPstatus = 4; %If next node is the last node, go to status 4
            else
                app.SPstatus = 2; %If not, go again to status 2
                app.NodeFlag = app.NodeFlag + 1;
            end
        end
    end
```

```
    end
case 4 %path tracking to final point
    app.EV(EV).mode_operation = 0; %path tracking mode
    SendMessage(app,EV);
    %checks if the EV reaches its destination (Next node)
    app.SPstatus = SPcheckbox(app,EV,app.CoordinatesNodes(app.node_path(end),1),app.CoordinatesNodes(app.node_path(
case 5 %Destination reached
    app.EV(EV).mode_operation = 2; %Remote control mode
    SendMessage(app,EV);
    EVStop(app,EV); %Stop EV
    stop(app.ShortPathTimer); %Stop periodic execution of SHORT_PATH

case 6 %Check if EV is facing closest node or opposite direction
    pause(0.5) %Waits half a second for the new position

    %Calculates again distance to closest node
    [~, app.distance_2] = find_source_node(app.EV(EV).position(1),app.EV(EV).position(2));


    if app.distance_2 > app.distance_1 %Compares the two distances
        %If the second value is greater, EV is going away from closest node.
        %The EV would need to turn 180º to face node
        app.new_yaw = app.EV(EV).yaw +180;
        if app.new_yaw > 360
            app.new_yaw = 360 - app.new_yaw;
        end
        disp(app.new_yaw)

        app.SPstatus = 7; %Goes to status 7
    else
        app.SPstatus = 2; %Goes to status 2
    end

case 7 %Turns 180º to face closest node

    app.EV(EV).mode_operation = 2; %Remote control mode
    SendMessage(app,EV);
    if app.new_yaw > 360-app.AngleTolerance || app.new_yaw < app.AngleTolerance
        if  app.EV(EV).yaw > app.AngleTolerance && app.EV(EV).yaw <360-app.AngleTolerance
            EVRotate(app,EV,app.new_yaw);
        else
            app.SPstatus = 2;
            EVStop(app,EV)
        end

    else
        if app.EV(EV).yaw >app.new_yaw +app.AngleTolerance || app.EV(EV).yaw <app.new_yaw - app.AngleTolerance
            EVRotate(app,EV,app.new_yaw);
        else
            app.SPstatus = 2;
            EVStop(app,EV)
        end
    end
```

Calls find_source_node

```
function [source_node,distance_to_point] = find_source_node(present_x,present_y)
    CoordinatesNodes=[2.2,-2.12; %1
                1000,1000;%2
                1000,1000;%3
                -.72,-1.81;%4
                -.69,-2.21;%5
                0.71,-3.38;%6
                1.88,-2.54;%7
                1.87,-2.02;%8
                1.87,-1.77;%9
                1.82,-1.3;%10
                -0.38,-1.4;%11
                -0.37,-1.88;%12
                -0.37,-2.13;%13
                -.37,-2.57;%14
                2.07,-1.92;%15
                -.52,-1.99;%16
                0.68,-3.04;%17
                0.75, 0.17];%18

            %present location
            x=present_x;
            y=present_y;
            outer_loop=[1,2,3,4,5,6];
            inner_loop=[7,8,9,10,11,12,13,14,17,18];
            flag=0;
if(y-0.9858*x+0.4465<0)
    if(x<1.88 && x>-0.46)
        if (y<0.45 && y>-3.2)
            flag=0;
        else
            flag=1;
        end
    else
        flag=1;
    end
 else
      flag=1;
end
if(flag==0)
for i=1:length(inner_loop)
    a=(CoordinatesNodes(inner_loop(i),1)-x).^2;
    b=(CoordinatesNodes(inner_loop(i),2)-y).^2;
    sum=a+b;
    distance(i)=sqrt(a+b);
    %distance(i)=sqrt(sum((CoordinatesNodes(inner_loop(i),1)-present_location_x).^2,
    %(CoordinatesNodes(inner_loop(i),2)-present_location_y).^2))
end
c=find(distance==min(distance));
distance_to_point=min(distance);
source_node=inner_loop(c);
end
```

```
c=find(distance==min(distance));
distance_to_point=min(distance);
source_node=inner_loop(c);
end
if(flag==1)
for i=1:length(outer_loop)
    a=(CoordinatesNodes(outer_loop(i),1)-x).^2;
    b=(CoordinatesNodes(outer_loop(i),2)-y).^2;
    sum=a+b;
    distance(i)=sqrt(a+b);
    %distance(i)=sqrt(sum((CoordinatesNodes(inner_loop(i),1)-present_location_x).^2,
    %(CoordinatesNodes(inner_loop(i),2)-present_location_y).^2))
end
c=find(distance==min(distance));
outer_loop(c);
source_node=outer_loop(c);
distance_to_point=min(distance)
end
```

Calls shortest_path

```
function [nodes_1,angles] = shortest_path (present,target)
    G=[ 0 1 0 0 0 6 0 0 0 0 0 0 0 0 1 0 0 0;%1
      1 0 12 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;%2
      0 12 0 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0;%3
      0 0 16 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;%4
      0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 1 0 0;%5
      6 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0;%6
      0 0 0 0 0 0 0 2 0 0 0 0 0 6 0 0 4 0;%7
      0 0 0 0 0 0 2 0 1 0 0 0 6 0 1 0 0 0;%8
      0 0 0 0 0 0 0 1 0 2 0 6 0 0 1 0 0 0;%9
      0 0 0 0 0 0 0 0 2 0 6 0 0 0 0 0 0 6;%10
      0 0 0 0 0 0 0 0 0 6 0 2 0 0 0 0 0 12;%11
      0 0 0 0 0 0 0 0 6 0 2 0 0 0 0 1 0 0;%12
      0 0 0 0 0 0 0 6 0 0 0 0 0 2 0 1 0 0;%13
      0 0 0 0 0 0 6 0 0 0 0 0 2 0 0 0 4 0;%14
      1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0;%15
      0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0;%16
      0 0 0 0 0 0 4 0 0 0 0 0 0 4 0 0 0 0;%17
      0 0 0 0 0 0 0 0 0 6 12 0 0 0 0 0 0 0;];%18

    present_node=present;
    target_node=target;
    [e L] = dijkstra(G,target_node,present_node);
```

```
    angles(1) = 500;
    for i =1:(length(L)-1)
        if L(i) ~= 2 && L(i) ~=3
            nodes_1(i)=L(i);
            th= next_angle(L(i),L(i+1));
            angles(i)=th;

        end

    end

    nodes_1(length(L))=L(length(L));
    disp('sp funct')
    disp(nodes_1)
    disp(angles)
end
```

Calls SPCheckbox:

```
function status = SPcheckbox(app,EV, point_x, point_z)

    if app.EV(EV).position(1) > point_x-app.BoxTolerance && app.EV(EV).position(1) < point_x+
       EVStop(app,EV);
       if app.SPstatus == 2
          status = 3;
       else
          status = 5;
       end
    else
        if app.SPstatus == 2
            status = 2;
        else
            status = 4;
        end
    end
end
```

Additional functions:

```
function ChangeMode(app,ev,newmode) %Change in mode of operation
     switch newmode
        case "Platooning"
            app.EV(ev).mode_operation = 1;
            message = sprintf('Mode EV%d Platooning',ev);
            updateLog(app, message);
            app.EV(ev).Status= 0; %parking status 0

        case "Path Tracking"
            switch app.EV(ev).Status
                case 7
                   app.EV(ev).Status = 8; %If the EV is parked, it needs to depark.
                   %Sets stage to 8

                otherwise
                    app.EV(ev).mode_operation = 0;
                    message = sprintf('Mode EV%d Path Tracking',ev);
                    updateLog(app,message);
                    app.EV(ev).Status = 0; %parking status 0
            end

        case "Parking"
            app.EV(ev).Status = 1; %Starts parking flow

            message = sprintf('Mode EV%d going to parking',ev);
            updateLog(app,message);

            start(app.EV(ev).ParkingEntraceTimer)

    end
end
```

```
function ConnectEV(app,EV, EV_IP,EV_Port,EV_Port_Park) %Creates UDP objects to send and receive
            %Receiver
            try
            app.EV(EV).Receiver = dsp.UDPReceiver('LocalIPPort',EV_Port);
            app.EV(EV).Receiver.RemoteIPAddress= EV_IP;
            app.EV(EV).Receiver.ReceiveBufferSize = app.BufferSize;
            setup(app.EV(EV).Receiver);
            catch
                disp('Error')
                message = sprintf('Setup EV%d Receiver failed',EV);
                updateLog(app,message);
            end

            %Sender
            try
            app.EV(EV).Sender = dsp.UDPSender('RemoteIPPort', EV_Port);
            app.EV(EV).Sender.RemoteIPAddress= EV_IP;
            catch
                message = sprintf('Setup EV%d Sender failed',EV);
                updateLog(app,message);
            end
            %ParkingSender
            try
            app.EV(EV).ParkingSend = dsp.UDPSender('RemoteIPPort',EV_Port_Park);
            app.EV(EV).ParkingSend.RemoteIPAddress= EV_IP;
            catch
                message = sprintf('Setup EV%d ParkingSend failed',EV);
                updateLog(app,message);
            end
            pause(0.001)

            app.EVConnected(EV) = 1;
end
```

```
function DisconnectEV(app,EV) %Disconnects UDP objects
        app.EVConnected(EV) = 0;
        try
            release(app.EV(EV).Receiver);
        catch
            message = sprintf('Disconnect EV%d Receiver failed',EV);
            updateLog(app,message);
        end
        try
        release(app.EV(EV).Sender);
        catch
            message = sprintf('Disconnect EV%d Sender failed',EV);
            updateLog(app,message);
        end
        try
        release(app.EV.(EV).ParkingSend);
        catch
            message = sprintf('Disconnect EV%d ParkingSend failed',EV);
            updateLog(app,message);
        end



end

% update log information in the log area, latest 5 information will be shown
function updateLog(app,message)
    app.Log{app.LogIndex} = message;
    if app.LogIndex <= 5
        app.LogTextArea.Value = app.Log(1:app.LogIndex);
    else
        app.LogTextArea.Value = app.Log(app.LogIndex-5:app.LogIndex);
    end
    app.LogIndex = app.LogIndex + 1;
end
```

```
function SendMessage(app,Ev)
    %Sends UDP packet to theEVs. Parameters: #EV (1-4), status(Start/stop), mode(path/plat),base speed, Plat distance, trim value, leader

    if Ev == 1 && app.Switch_1.Value == "On"
        app.EV(1).Sender(uint8([app.start_stop app.EV(1).mode_operation app.BaseSpeedSlider_1.Value app.PlatooningDistanceSlider_1.Value app.EV(1).Trim_value app.Leader(1)]));
    end

    if Ev == 2 &&  app.Switch_2.Value == "On"
        app.EV(2).Sender(uint8([app.start_stop app.EV(2).mode_operation app.BaseSpeedSlider_2.Value app.PlatooningDistanceSlider_2.Value app.EV(2).Trim_value app.Leader(2)]));
    end

    if Ev == 3 &&  app.Switch_3.Value == "On"
        app.EV(3).Sender(uint8([app.start_stop app.EV(3).mode_operation app.BaseSpeedSlider_3.Value app.PlatooningDistanceSlider_3.Value app.EV(3).Trim_value app.Leader(3)]));
    end

    if Ev == 4 &&  app.Switch_4.Value == "On"
        app.EV(4).Sender(uint8([app.start_stop app.EV(4).mode_operation app.BaseSpeedSlider_4.Value app.PlatooningDistanceSlider_4.Value app.EV(4).Trim_value app.Leader(4)]));
    end

end
```

```
function [display] = LeftTrimValueChange(app,ev)
    %Decreases Trim value by 1 if Trim Value > -10
    if app.EV(ev).Trim_value > 0
        app.EV(ev).Trim_value = app.EV(ev).Trim_value - 1;
    end
    display = int2str(app.EV(ev).Trim_value-10);
end

function [display] = RightTrimValueChange(app,ev)
    %Increases Trim value by 1 if Trim Value < 10
    if app.EV(ev).Trim_value < 20
        app.EV(ev).Trim_value = app.EV(ev).Trim_value + 1;
    end
    display = int2str(app.EV(ev).Trim_value-10);
end
```

```
    function Characters(~, char, img)
        title(char, []);
        xlabel(char, []);
        ylabel(char, []);
        char.XAxis.TickLabels = {};
        char.YAxis.TickLabels = {};
        char.XLim= [0 , 1];
        char.YLim = [0, 1];
        imagesc([0,1] ,[0,1], img, 'Parent', char);
    end

    function PlaySound(~, file)
        try
            [y, Fs] = audioread(file);
            sound(y, Fs, 16);
        catch
        end

    end
```

Initialization of the timers:

```
app.EV(i).ParkingEntraceTimer = timer('ExecutionMode','fixedRate','Period',.05);
app.EV(i).ParkingEntraceTimer.TimerFcn = @(obj, event)PARKING(app,i,app.EV(i).Status);

app.SensorTimer = timer('ExecutionMode','fixedRate','Period',1);
app.SensorTimer.TimerFcn = @(obj, event)timerSensorUpdate(app);



app.OptitrackTimer = timer('ExecutionMode','fixedRate','Period',.1);
app.OptitrackTimer.TimerFcn = @(obj, event)UpdateOptitrack(app);


%ShortestPath
app.ShortPathTimer = timer('ExecutionMode','fixedRate','Period',.05);
app.ShortPathTimer.TimerFcn = @(obj, event)SHORT_PATH(app,app.SPEV,app.SPstatus);
```

Appendix B (User Manual) (Binoy)

User Manual

Setting up the EVs

- Connect each of the EVs to iSpace Wi-Fi. Password “QWERTYUI”. The EVs have fixed IP, so it will always be the same.
- Connect control Station to iSpace Wi-Fi. Check PC IP, it will be needed for next step.
- Enter correct communications parameters in the Simulink program (Sending and receiving UDP ports). The “Remote IP” will be that of the control station. Both sending and receiving ports will be the same for each EV.

EV	IP	EV3_PortNo	EV3_OptitrackPortNo
1	192.168.1.101	25001	25011
2	192.168.1.102	25002	25012
3	192.168.1.103	25003	25013
4	192.168.1.104	25004	25014

- Download program to EV.
- Place EVs on the track.

Setting up the OptiTrack

Hardware

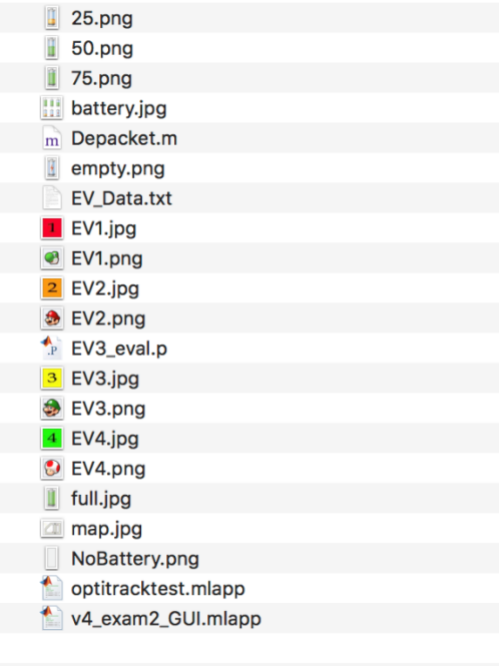
- Clear the area of the map
- Ensure no reflective material is present anywhere on the map
- Set up the EV3s such that all the reflective dots are in different orientations
- Place the EV3s along the z axis

Software

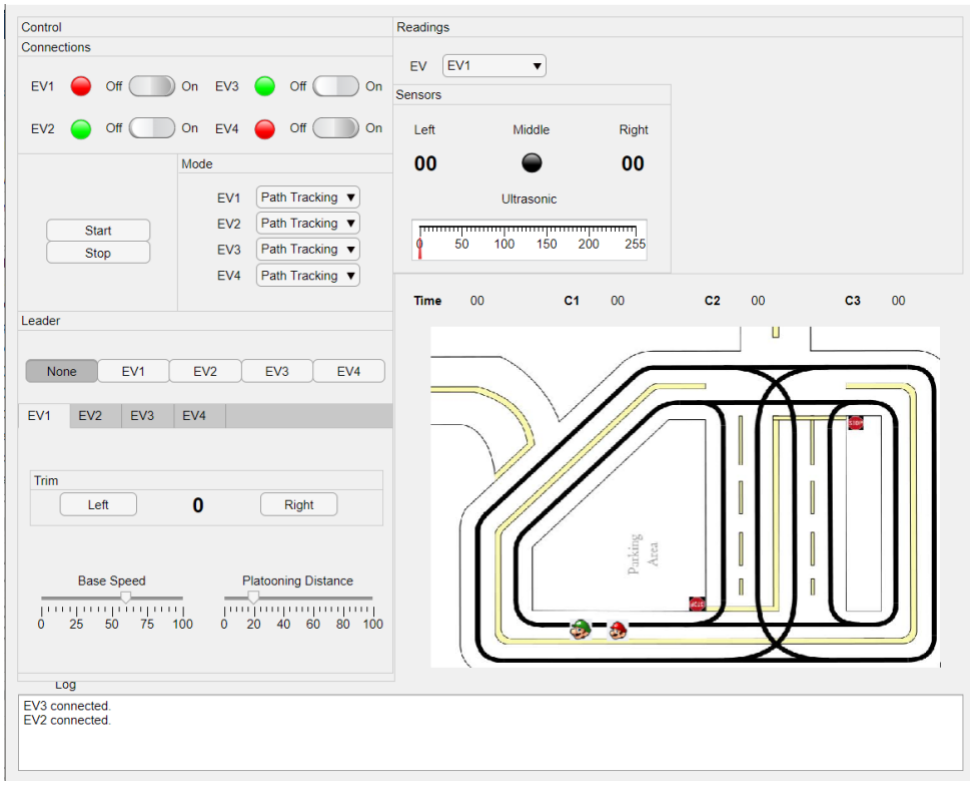
- Open TrackingTools on Host PC
- Select the trackable on the map
- Right click and select Create-Trackable
- Do this for all EV3s
- Go to orientation and click on reset to present orientation

GUI

Open Matlab and set the path to the folder with the program. The following files are needed to run it.

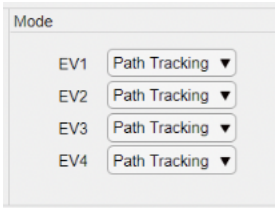


- Open v4_exam2_GUI

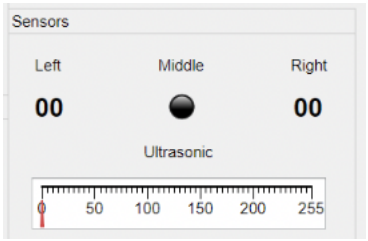


- From there click on the switch of the EV3s you want to connect and click on start

- Select the mode in which you want to run the EV

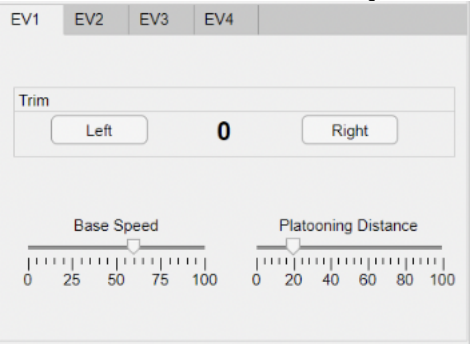


- You can observe the Sensor values in the reading tab

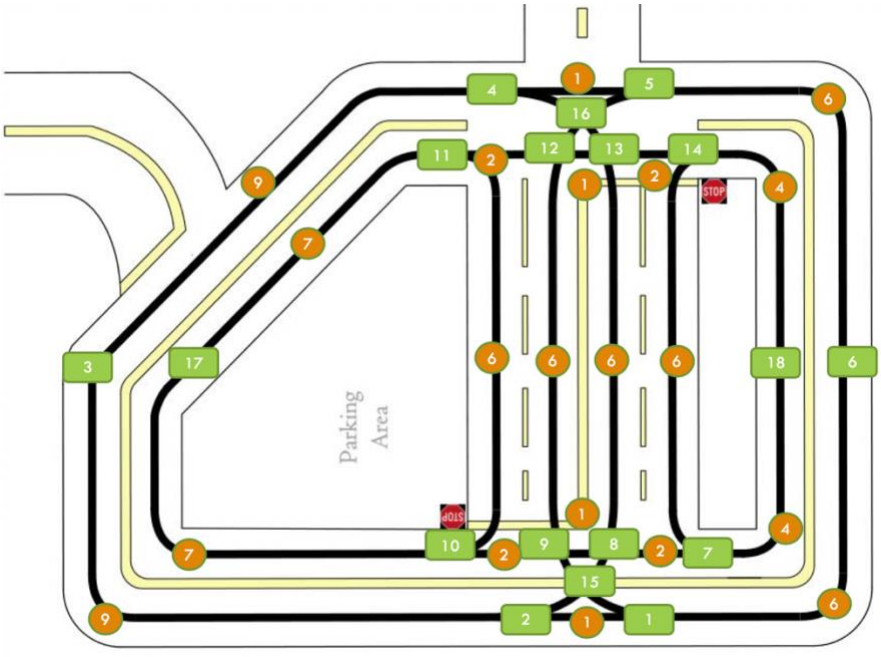


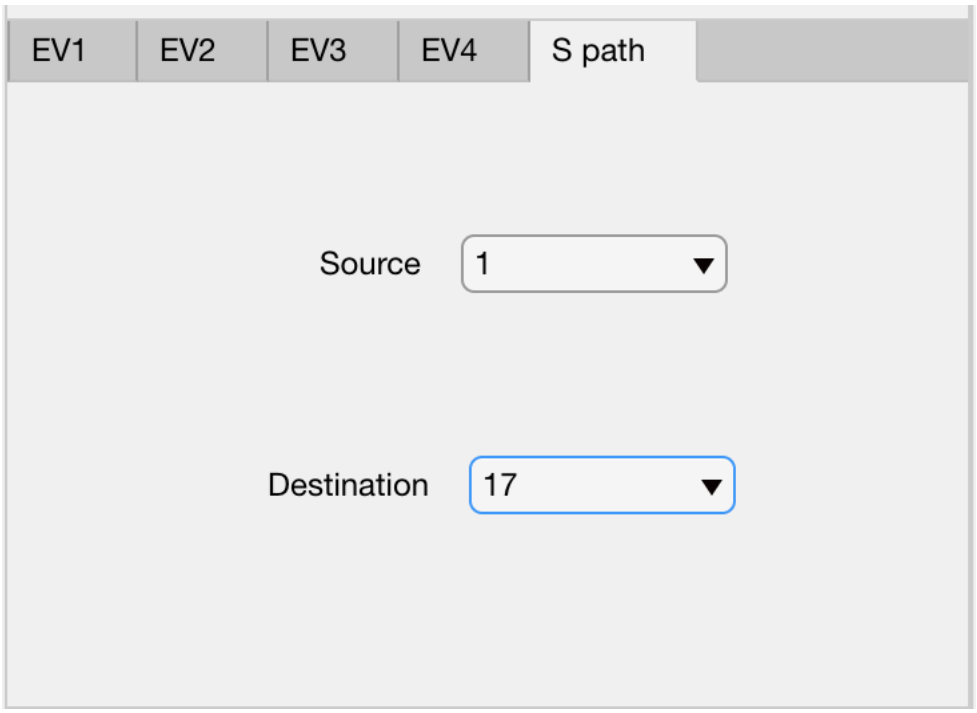
- You can observe the movement of the EV3s on the map

- You can set the base speed, trim value and platooning distance for each of the EV3s



- Change the value of the destination position to find the shortest path from the start point to the destination





Appendix C (EV3 Select Code) (Davis)

```
%% Setup EV3 for Communication
clc;
try %Setup IP Address
    if ipend == '0'
    end
catch
    ipend = string(input('IP Address == 192.168.0.'));
end
try %Initiate Security Key
    if key == 256
    end
catch
    key(1,1) = randi((2^6)-1);
    key(1,2) = key(1,1);
    key(2,1) = randi((2^6)-1) + key(1,1);
    key(2,2) = key(2,1);
end
ip_address = char(strcat('192.168.0.',ipend));
disp('EV1 == MARIO');
disp('EV2 == YOSHI');
disp('EV3 == LUIGI');
disp('EV4 == TOAD');
disp('EV5 == WARIO (4 SENSORS IN A LINE)');
```

```
disp('EV6 == BOWSER (CAR)');
EV = input('Which EV3 Are You Using? EV: ');
switch(EV)
   case 1
      PORT1 = 25001;
      PORT2 = 25011;
      PORTM = [25001 25001; 25001 25001];
   case 2
      PORT1 = 25002;
      PORT2 = 25012;
      PORTM = [25002 25002; 25002 25002];
   case 3
      PORT1 = 25003;
      PORT2 = 25013;
      PORTM = [25003 25003; 25003 25003];
   case 4
      PORT1 = 25004;
      PORT2 = 25014;
      PORTM = [25004 25004; 25004 25004];
   case 6 % CAR
      PORT1 = 25005;
      PORT2 = 25015;
      PORTM = [25005 25005; 25005 25005];
   case 5 % 3 SENSORS IN A LINE
      PORT1 = 25006;
      PORT2 = 25016;
      PORTM = [25006 25006; 25006 25006];
   otherwise
      disp('ERROR: NON-VALID EV NUMBER');
end
```

Appendix D (Path Tracking Graphs) (Davis)

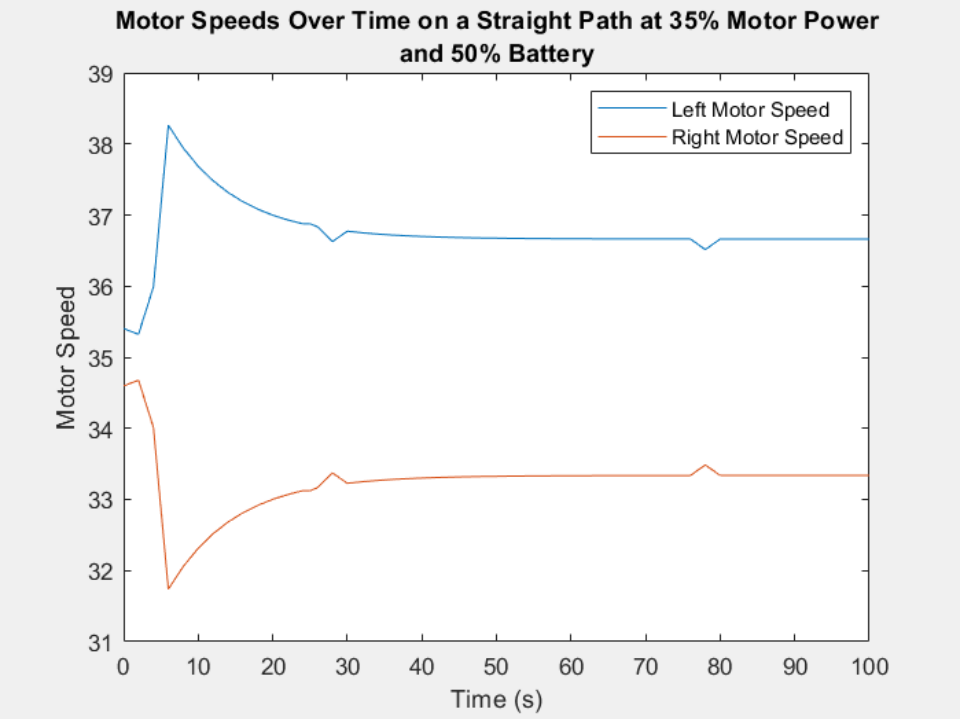


Figure D-1. Motor Speeds Graph 2

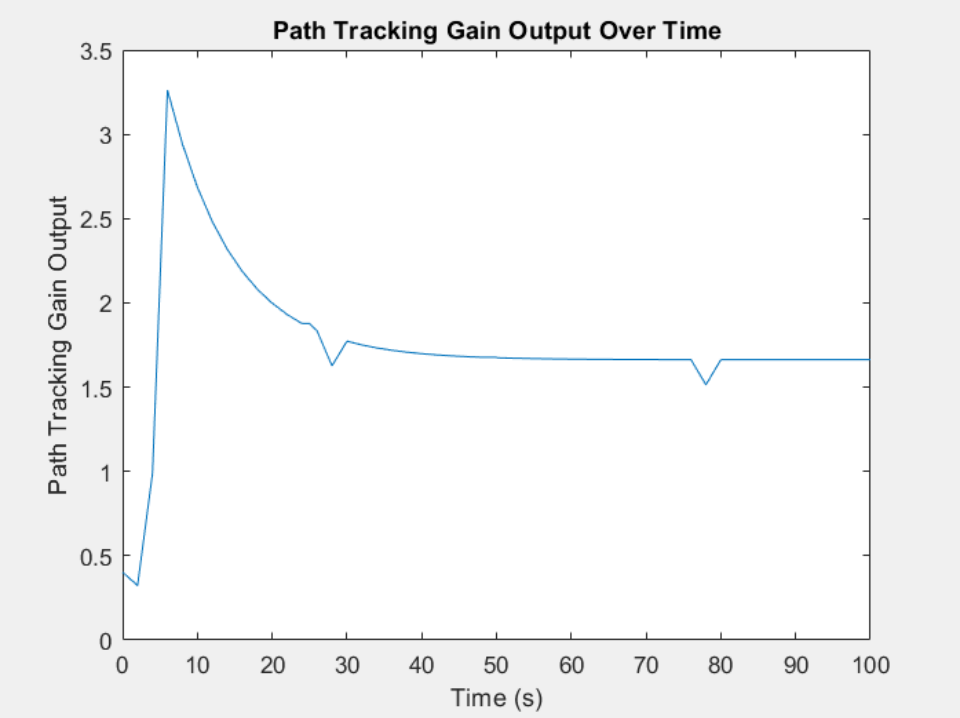


Figure D-2. Path Tracking Gain Output 2

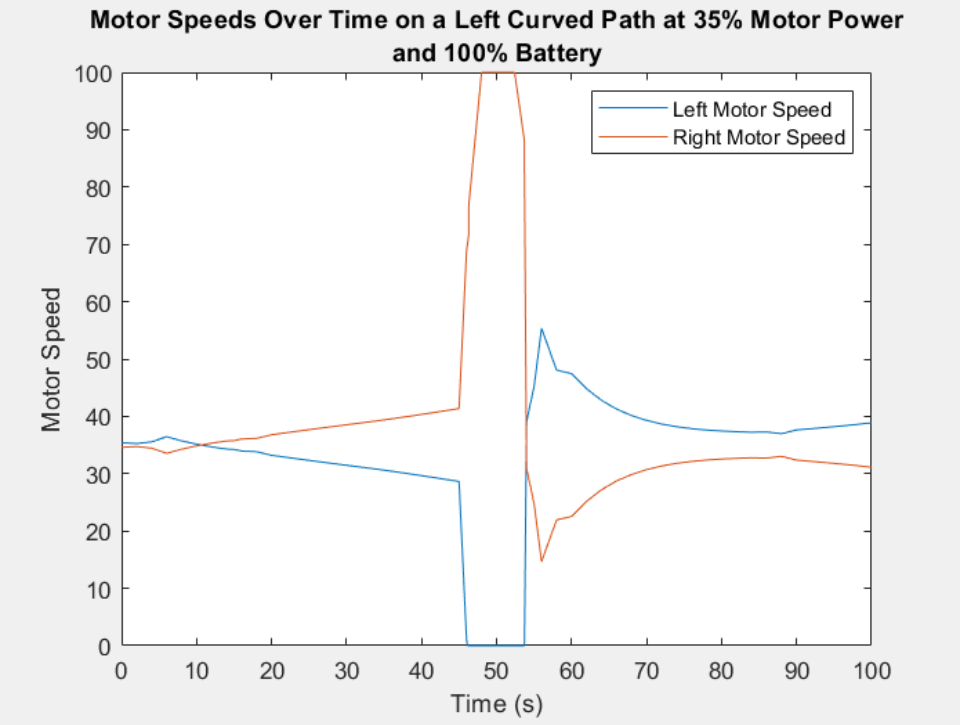


Figure D-3. Motor Speeds Graph 3

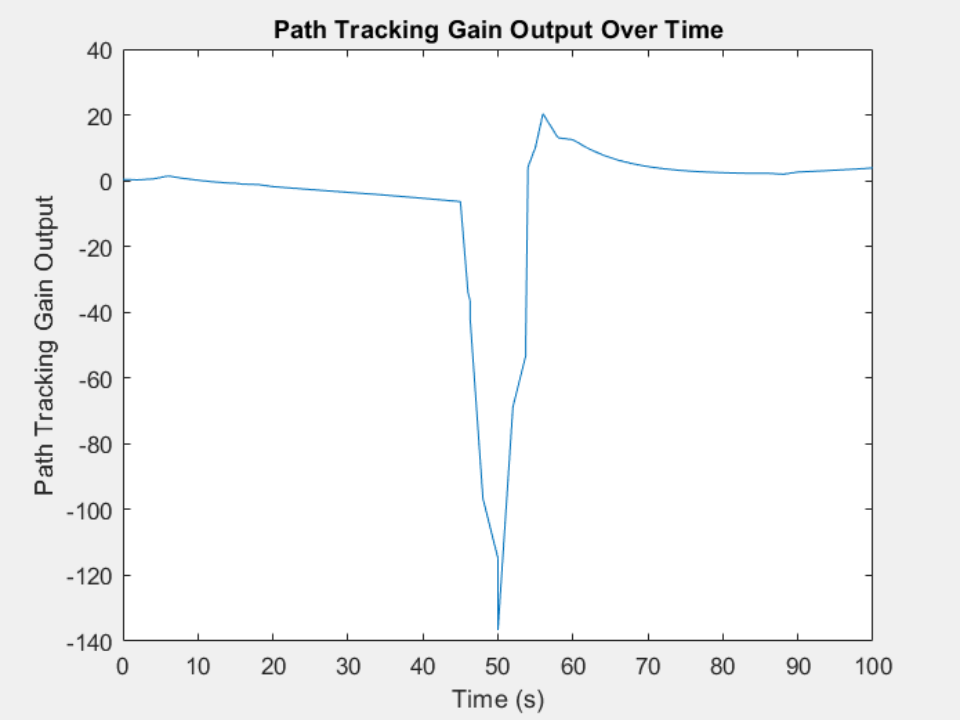


Figure D-4. Path Tracking Gain Output 3

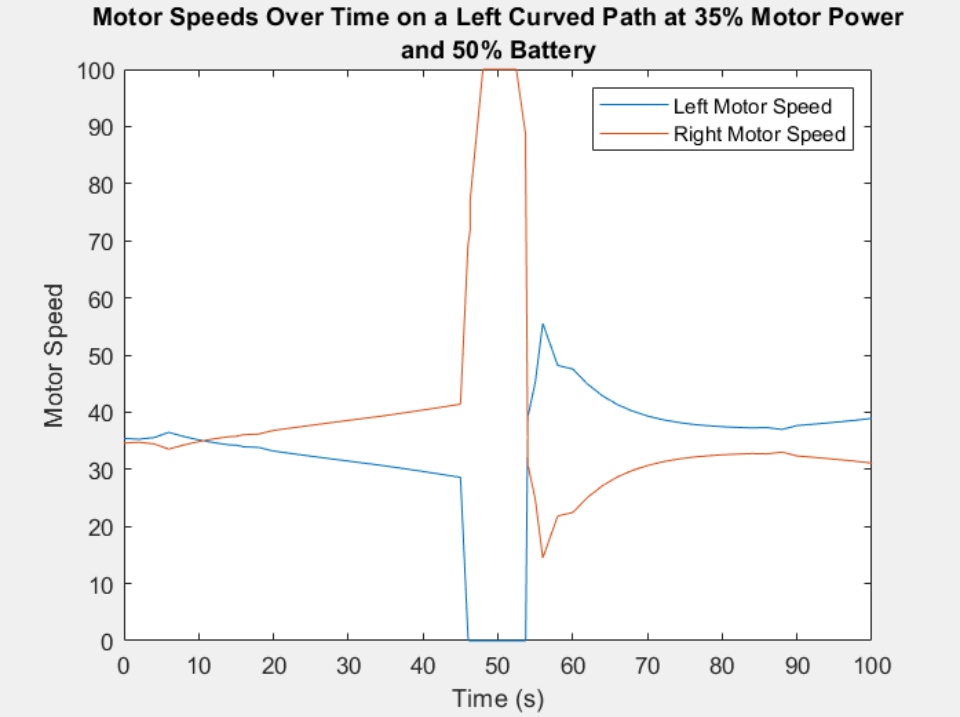


Figure D-5. Motor Speeds Graph 4

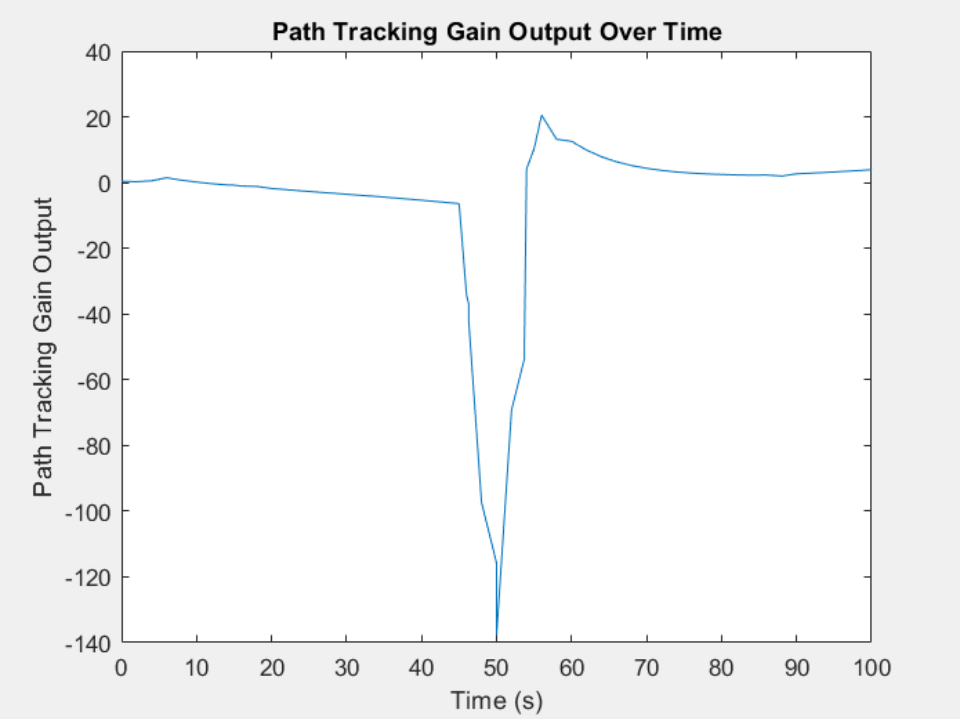


Figure D-6. Path Tracking Gain Output 4

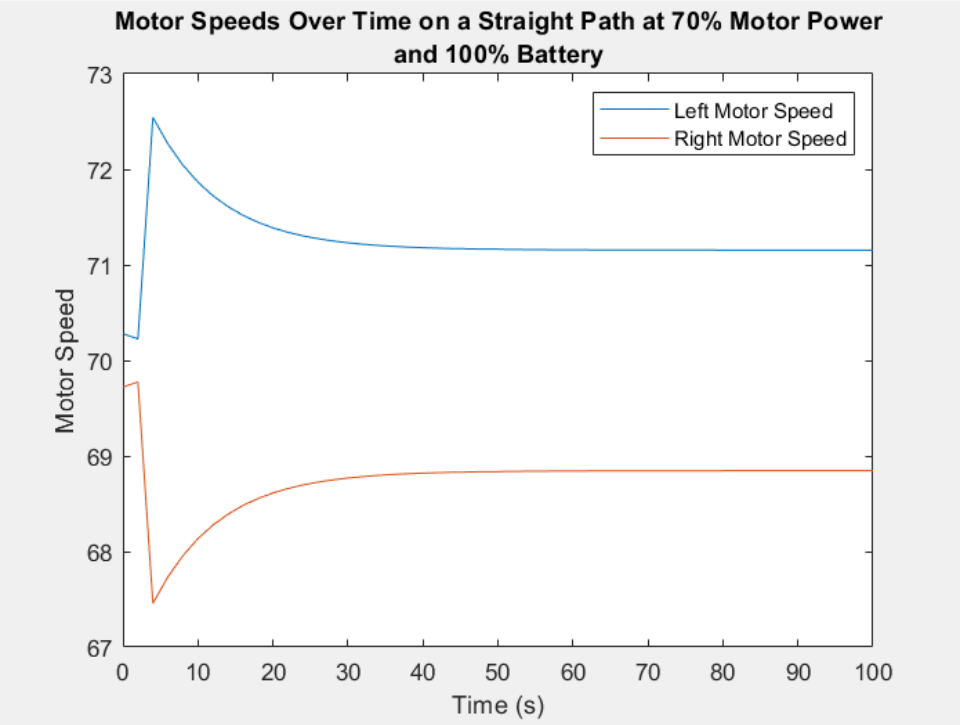


Figure D-7. Motor Speeds Graph 5

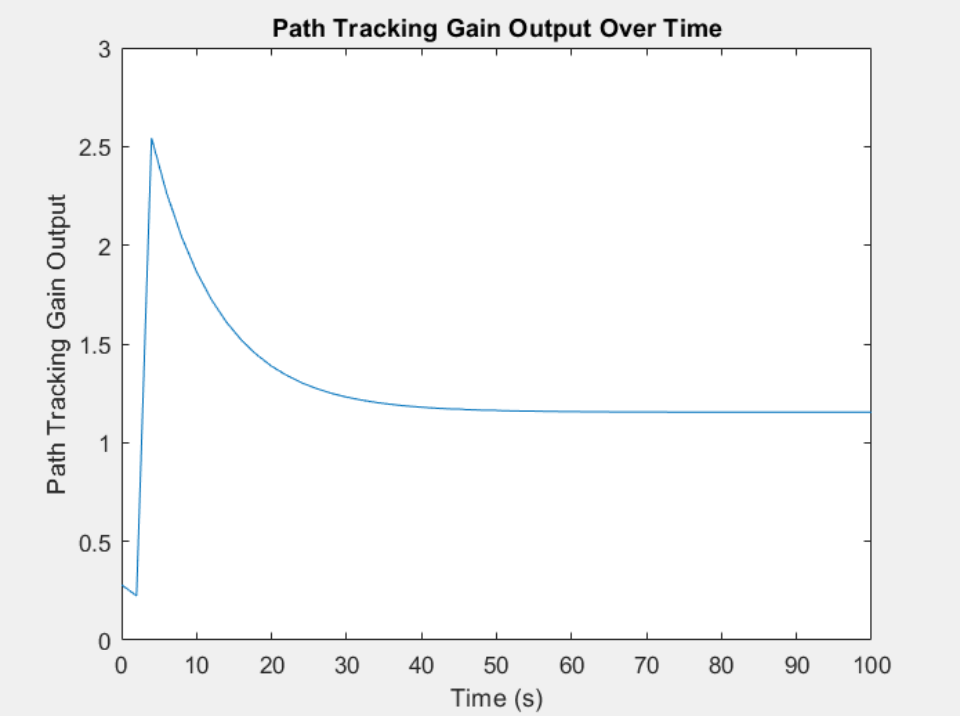


Figure D-8. Path Tracking Gain Output 5

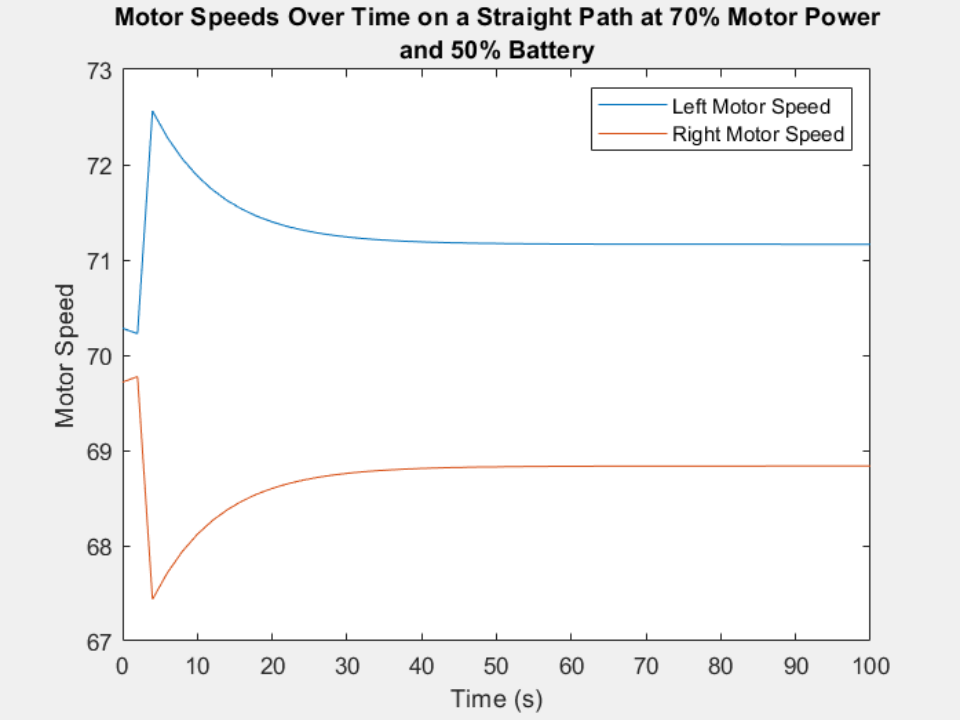


Figure D-9. Motor Speeds Graph 6

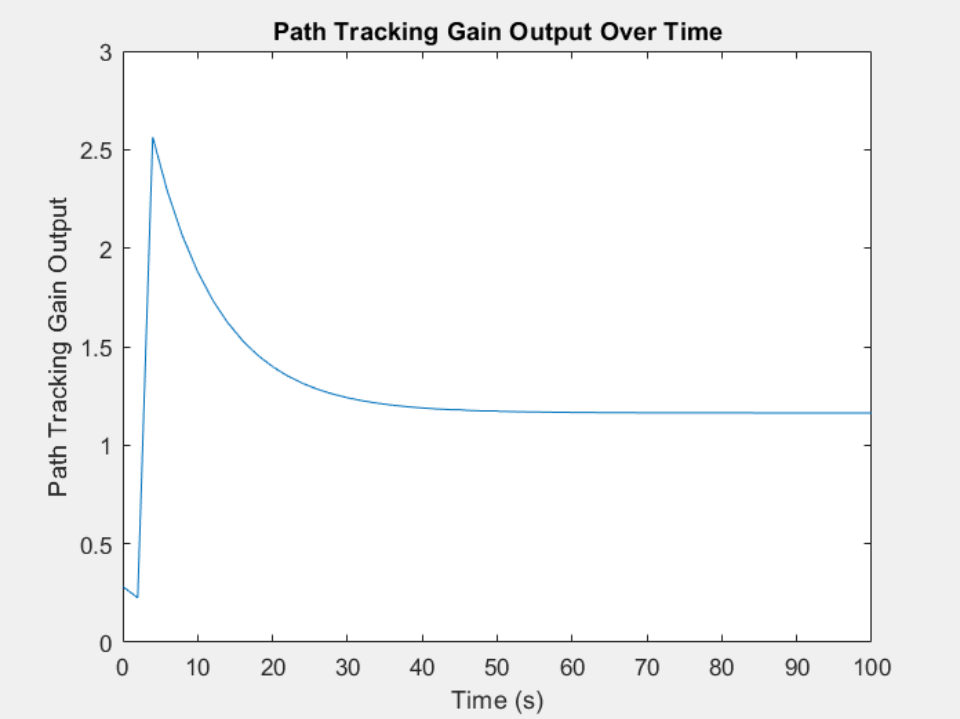


Figure D-10. Path Tracking Gain Output 6

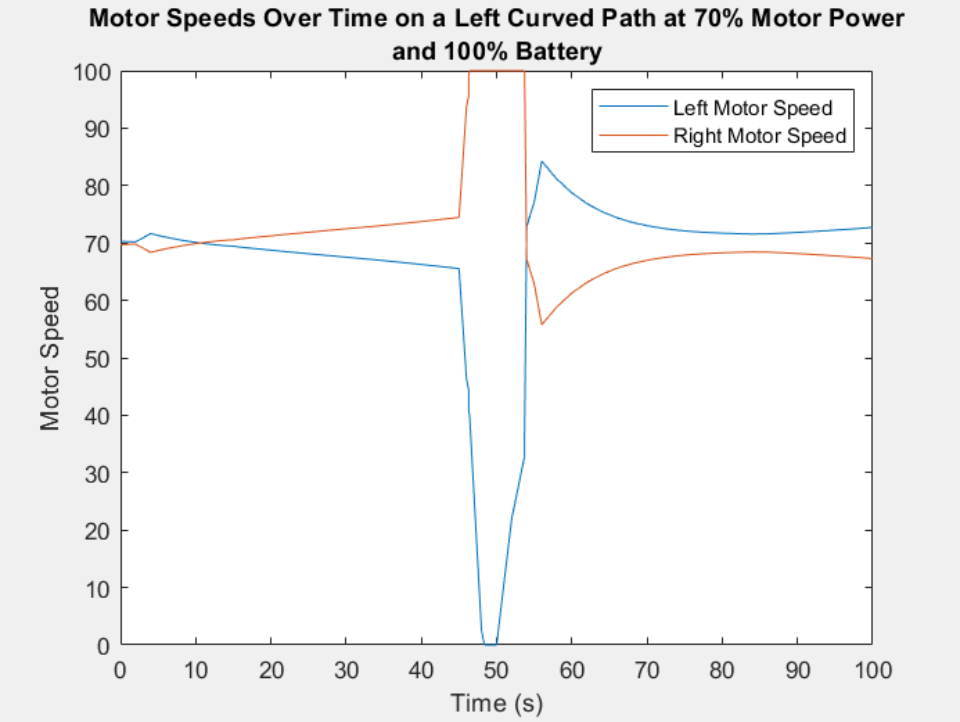


Figure D-11. Motor Speeds Graph 7

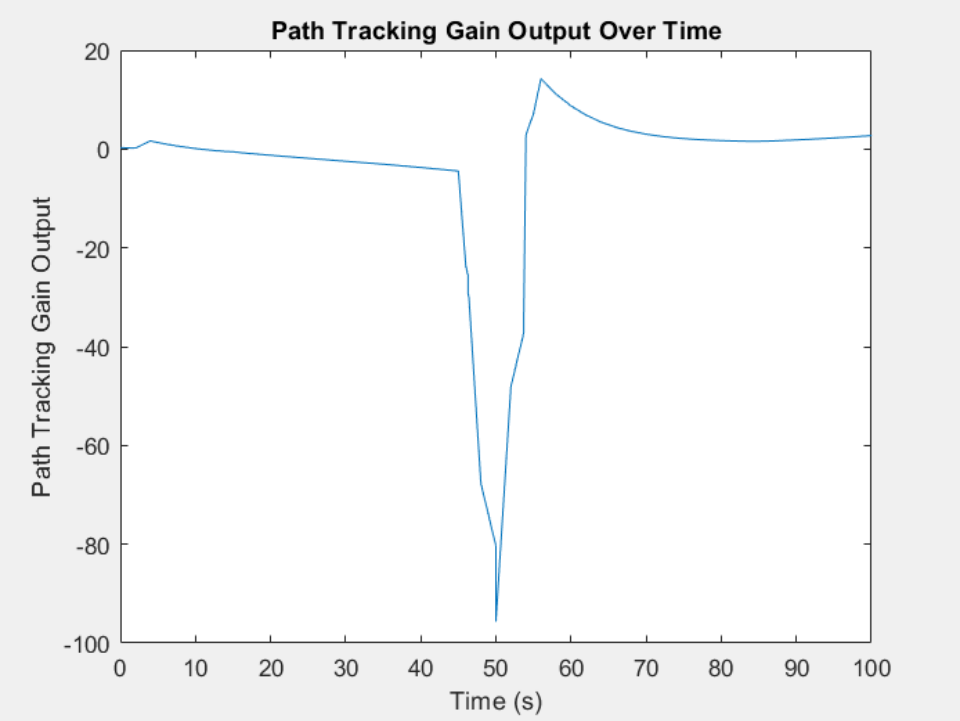


Figure D-12. Path Tracking Gain Output 7

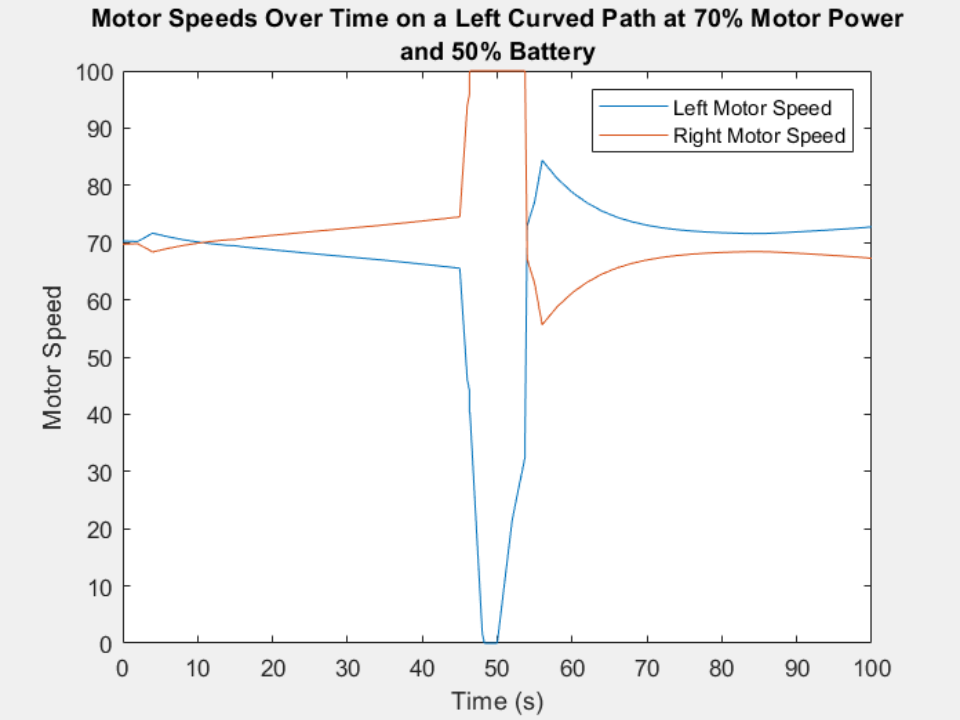


Figure D-13. Motor Speeds Graph 8

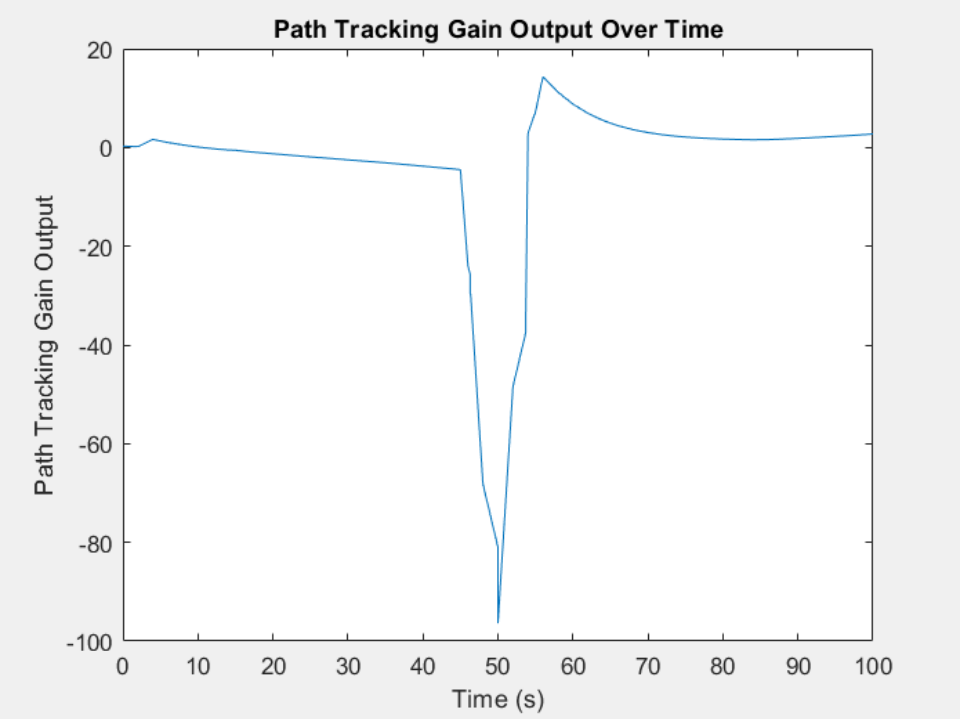


Figure D-14. Path Tracking Gain Output 8