ECE 4333 - Robotics
Final Report for Course Project

Project Name :	Traffic Light Robot
Project Partners:	Eric Henderson (ehender4@unb.ca)
	Burak Koryan (bkoryan@unb.ca)
Instructor:	Prof.Chris Diduch (diduch@unb.ca)
Date :	April 11th, 2019

University of New Brunswick

Fredericton, NB, Canada

April 2019

Table of Contents:

Summary:

In our project, the following objectives were achieved successfully: Individually speed control for both motors, with stabilization on a setpoint using PI controllers, two way communication through bluetooth for commands and data feedback, calibration of the PixyCam to accurately determine the distance from an observed marking and finally, three distinct subsystems.

Two autonomous control subsystems were designed and implemented as a part of this project: Subsystem 1 identifies a specific color combination as a command and makes the robot move accordingly and stabilizes on a user-provided setpoint while subsystem 2 makes the robot approach an observed color combination and centers it in the field-of-view of the sensor. The robot also settles on a specific distance from the observed marking. The final subsystem within this design consists of a separate mode of operation. This mode of operation allows the user to manually control the forward velocity and the angular velocity of the robotic system. This manual control is implemented using a bluetooth transceiver and w-a-s-d keys on a PC keyboard. The only objective our project fails to meet is searching surroundings for markings not currently seen in the field-of-view.

The robot also goes beyond our expectations for having a fast response to keystrokes in manual control and allowing for live tuning of control parameters with the user interface. Additionally, timeouts for given commands or speed commands when approaching a marking that can no longer be observed has been implemented, which adds a degree of safety in the design.

A video showcasing the performance of the two autonomous control subsystems can be found here:
(https://drive.google.com/file/d/1nd9kzSXGKwrQ_oRS3v3IWpPFzCQHyclW/view?usp=sharing)

A short video showing the manual control working can be found here:
(https://drive.google.com/file/d/1o69Zl48Ji8DswbgFZ6mbi1F1XEfC-OsX/view?usp=sharing)

Introduction:

The following list of work was completed successfully by Eric Henderson and Burak Koryan.

- Robot frame setup
- Wireless communication with a PC
- Robot control through software
    - Subsystem 1: Traffic Light Command Execution
    - Subsystem 2: Traffic Light Approach
    - Manual Control
- Schematic and design diagrams
- Project document preparation

Eric	Burak
Conceptualization of subsystems	Conceptualization of subsystems
PixyCam training	Bluetooth communication
Outer loop control	PixyCam calibration
Subsystem troubleshooting	Schematics and diagrams
Inner loop controller	Inner loop controller
Control system tuning	Subsystem characterization
Hardware setup	Hardware setup
Report preparation and editing	Design files and document preparation

Table 1: *Work distribution between group members*

Some assistance was provided and received from two groups within the class. For control related issues, Mike and Koceila were consulted to troubleshoot control gains and, to some extent, the control algorithm. Understanding of how control parameters can be adjusted to stabilize the system was found in discussing issues with them.

Jack and Ryan were consulted with issues with the PixyCam and reading the data from it. More specifically, the decision to implement a delay in the main loop of the program, where the PixyCam is read, was implemented based on their suggestions.

Low-Level Interface:

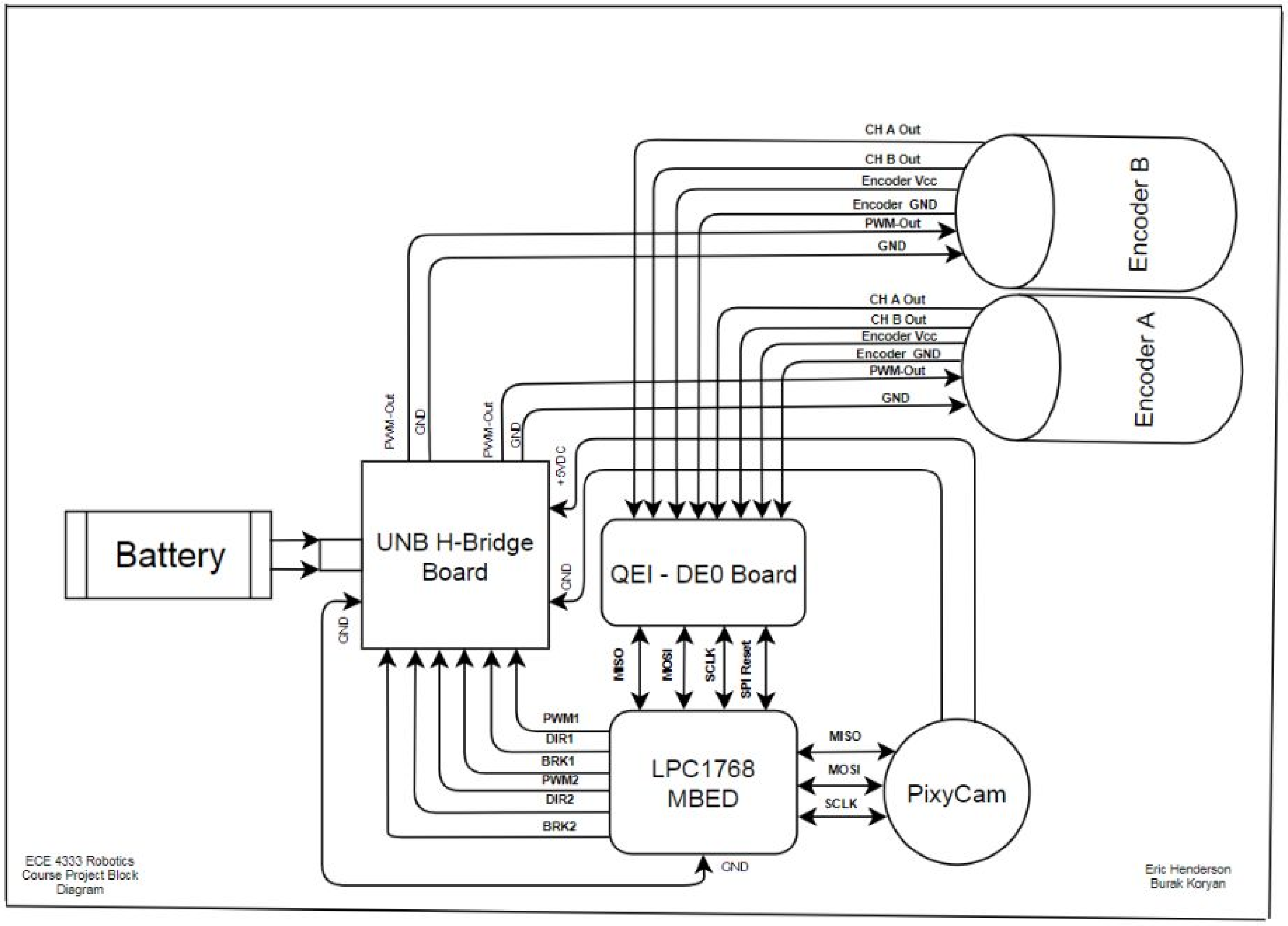


Figure 1: *Block diagram of low-level interface*

Our project includes seven main components such as two encoder motors for motion,a PixyCam as a sensor for detecting color commands,LPC1768 MBED as an embedded computer,an Altera DE0-FPGA board for quadrature encoder interface data processing,a UNB H-Bridge board for PWM amplification,and a 26.5V_{DC} 3000 mA LiFePo4 battery as a power supply.

As the robot has two drive wheels with separate motor driving control, fairly precise steering can be performed. This is achieved by decoupling control. Within the course lectures, decoupling control for both steering and forward acceleration were discussed, which is the approach used.

Below are two equations for individual motors velocities based on desired overall robot movement.

$$v_{Left} = v - \omega * \frac{L}{2} \qquad\qquad v_{Right} = v + \omega * \frac{L}{2}$$

The velocities, represented by "*v*" in the above equations, are in meters per second, while the angular velocity, "ω" in the above equations, is in radians per second. These velocities for the left and right motor are the input for the inner loop control system, and are stabilized on with feedback from the encoders.

This calculation can be seen represented in a block diagram in Figure 2 below.

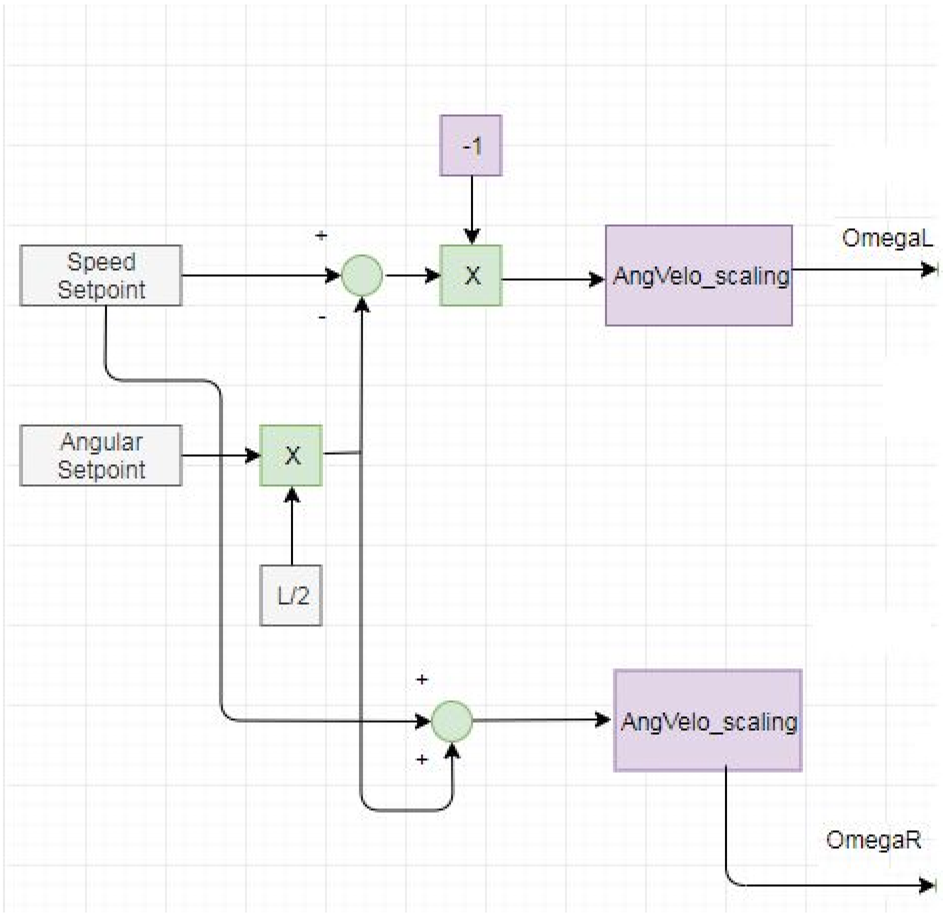


Figure 2: *Block diagram of steering of robot*

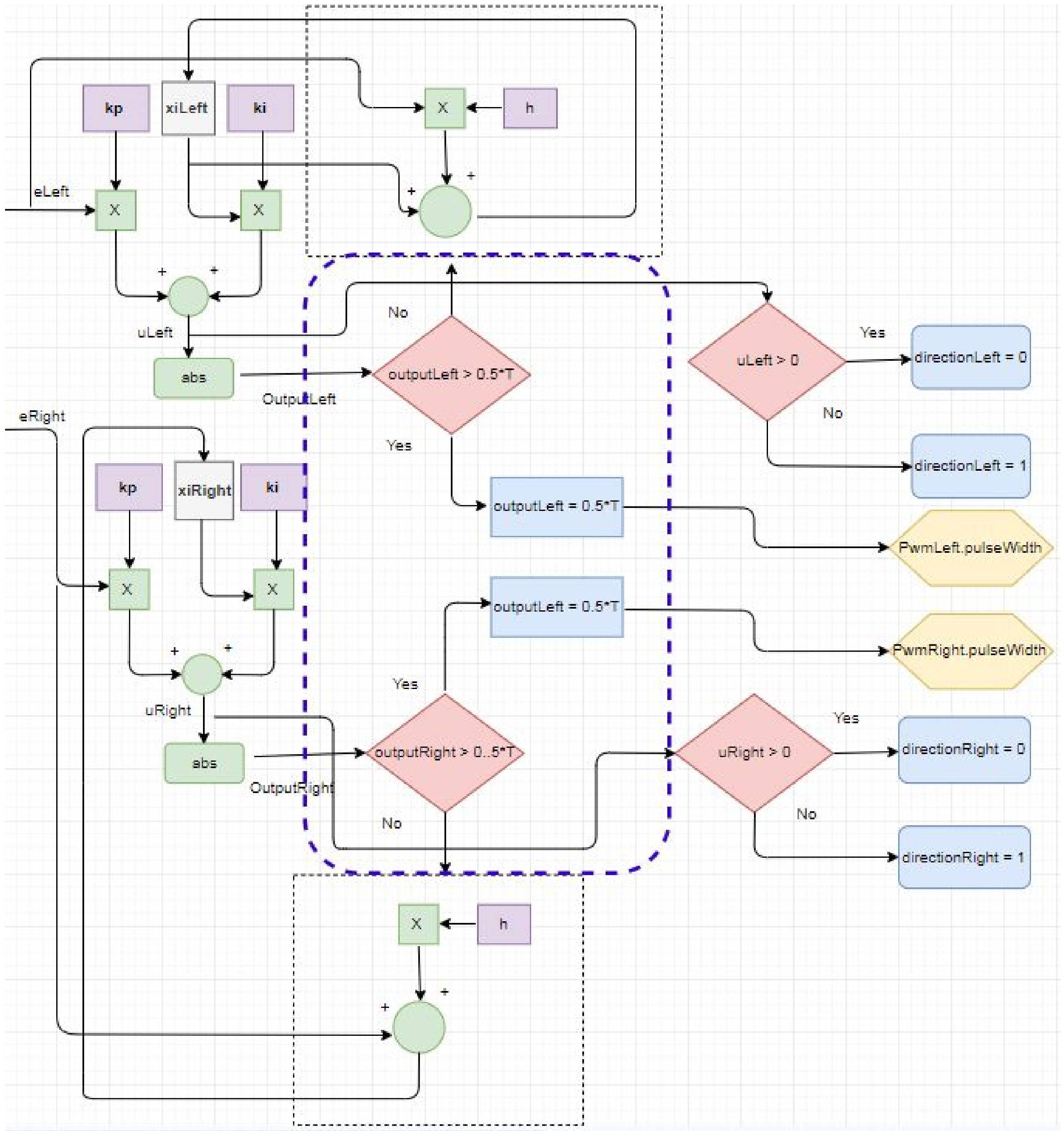


Figure 3: *Elimination of integration windup*

Figure 3 above shows the method with which integrator windup is avoided. If the output on a single motor is larger than its set maximum value, the output will be set to the maximum value and no integrator would be built up. Otherwise, the output remains as calculated by the decoupling calculations and the integrator is built up by adding the integral of the error to the carried over value.

Sensor Interface:

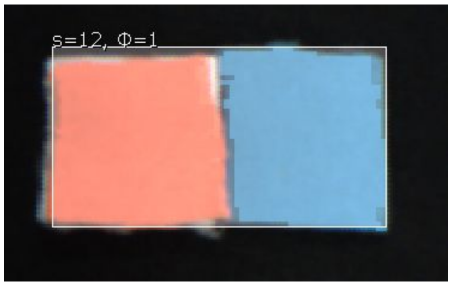


Figure 5: *Pixycam color code recognition*

In figure 5 above, how a color block is recognized after training the PixyCam on PixyMon desktop software is shown.The PixyMon software puts a rectangle on a recognized color code block with signature,with an s variable, and the angle its recognized at with a Φ variable.

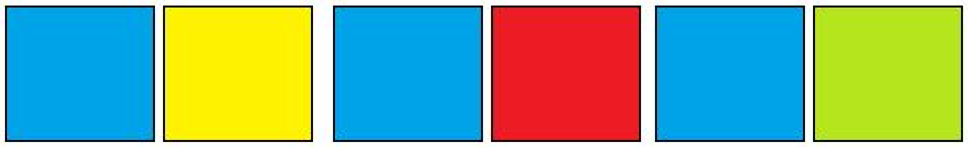


Figure 6: *Color blocks (4 cm x 4 cm)*

Figure 6 shows the color combinations used to control the robot.Blue color was chosen as a base color because its high contrast.Yellow,red,and green colors were selected because of their hue differences with blue.The color blocks were cut as squares with 4cm x 4cm dimension.

Combination	**Command**
Blue - Yellow	Right turn (90° turn)
Blue - Red	Left turn (-90° turn)
Blue - Green	U-turn (180° turn)

Table 2: *Color commands*

Table 2 shows the commands for robot control.The robot control software was designed so that if a blue-yellow color block is shown to the robot,there is a 90° right turn.Similarly if a blue-red color block is shown there is a -90° left-turn and if a blue-green color block shown,there is a 180° U-turn.

Experimental test	Result
Is the PixyCam powered as required?	Pixycam is supplied with $+5V_{DC}$ from MBED
Is color block recognition on Pixymon achieved?	As shown in figure 5, the color block recognition is possible with assigned block numbers
Is PixyCam SPI communication with MBED successful?	color block signatures,x and y coordinates,width and height of color block can be read on PuTTY
Does the robot recognize color-block commands?[1]	The robot turns left when blue-red block seen,turns right when blue-yellow block seen,makes a u-turn when blue-green block seen.
Does the robot approach to a shown color command?	The robot moves toward a color block target and stops at 20 cm to it.

Table 3: *Test plan for PixyCam communication*

Steps taken to calibrate PixyCam:

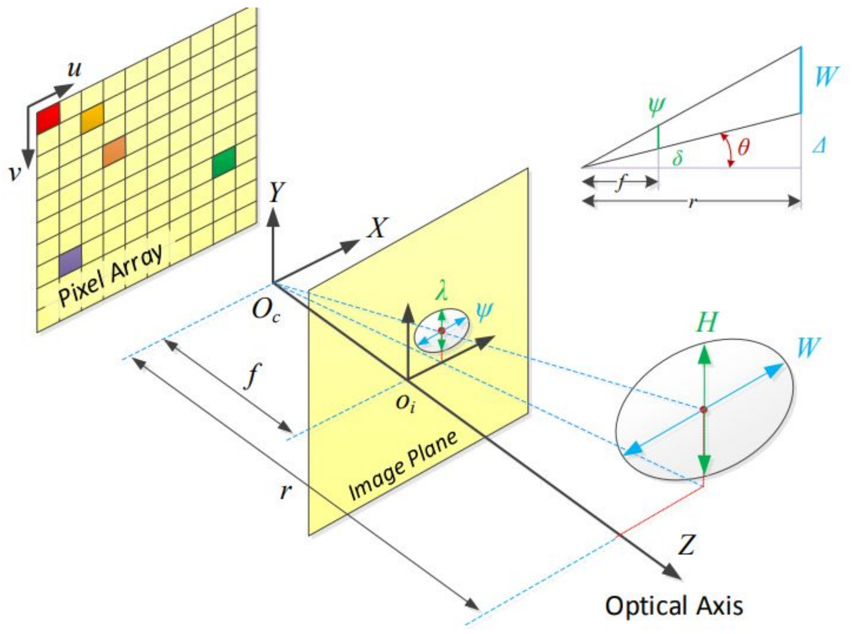


Figure 7: PixyCam depth perception[2]

- 4cm x 4cm two color squares shown above were cut out and taped onto a square block of wood
- A range of 10 cm to 55 cm measurement line were marked on the bench.
- Starting from 10 cm,width and height data (in pixels) of the color blocks were collected 10 times for each distance to the camera.The mean of the collected data was taken.
- A 20 x 1 matrix with height and width data points with lambda was created and height and weight data was divided by lambda.Another 20 x 1 matrix was also created for range (distance) values.
- MATLAB function Moore-Penrose pseudoinverse (pinv) was used for taking pseudo inverse of gamma (Γ)[1]. More information on Γ data can be found in Appendix A.
- As a result,approximately the *f* offset to calibrate the pixycam was calculated as 257.2969.
- λ and ψ data (pixels) was plotted against range (m) and best-fit lines were calculated.

$$\Gamma = \begin{bmatrix} \frac{H}{\lambda_1} \\ \frac{W}{\lambda_1} \\ \frac{H}{\lambda_2} \\ \frac{W}{\lambda_2} \\ \vdots \\ \frac{H}{\lambda_N} \\ \frac{W}{\lambda_N} \end{bmatrix} \quad R = \begin{bmatrix} r_1 \\ r_1 \\ r_2 \\ r_2 \\ \vdots \\ r_N \\ r_N \end{bmatrix}$$

Figure 8: Data matrices of gamma (Γ) and range (m)[2]

$$r = \frac{H}{\lambda} f, \quad r = \frac{W}{\psi} f$$

Figure 9: Formulas used to calculate range r[2]

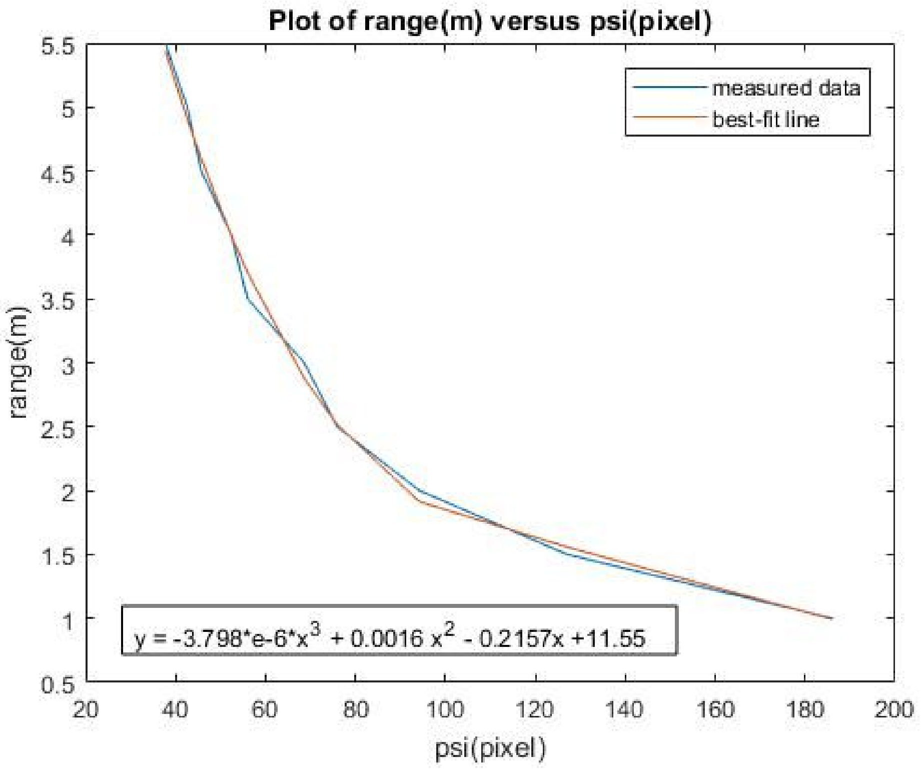


Figure 10: Plot of distance to the color block in meters versus ψ (psi) in pixels

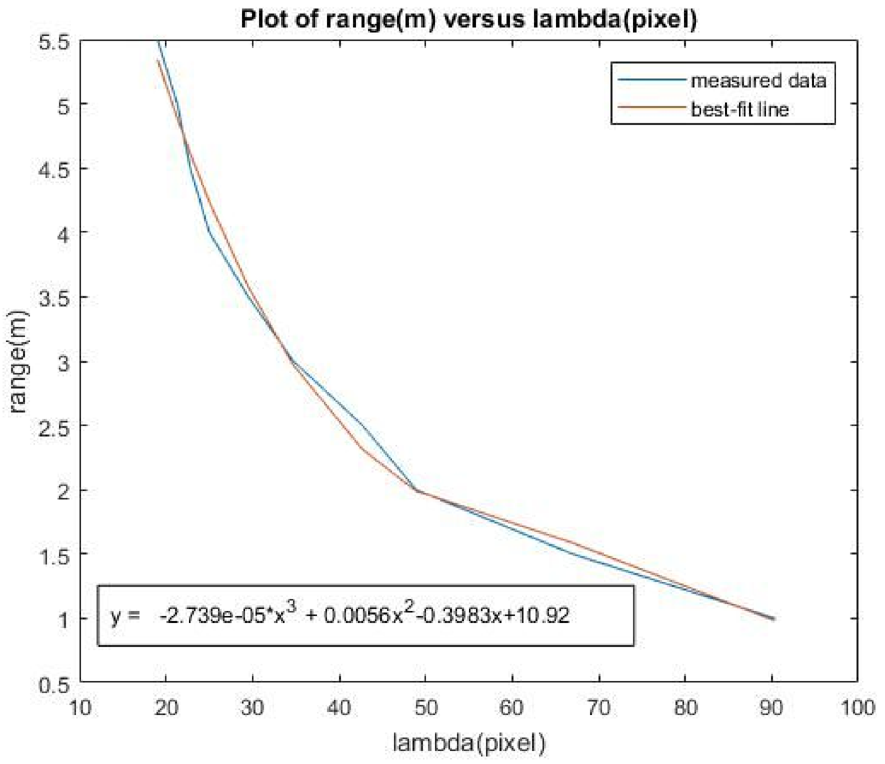


Figure 11: Plot of distance to the color block in meters versus λ (lambda) in pixels

In order to use PixyCam for approaching a color block target at a true distance,the PixyCam needed to be calibrated to get the *f offset* value that is shown in figure 7.As shown in appendix D,a Matlab script was written to manipulate pixel data gathered from PixyCam at various distances from,10 cm to 55 cm with 5 cm increments,where the color block target was placed.After some mathematical manipulation of data,the *f offset* value was calculated to be 257.2969.Figures 10 and 11 shows how λ and ψ,both in pixels,change with distance.Both plots have negative slopes which means that as the PixyCam goes closer to the target,the object is recognized larger since ψ or λ values increase with decreasing distance to the target.

User Interface:

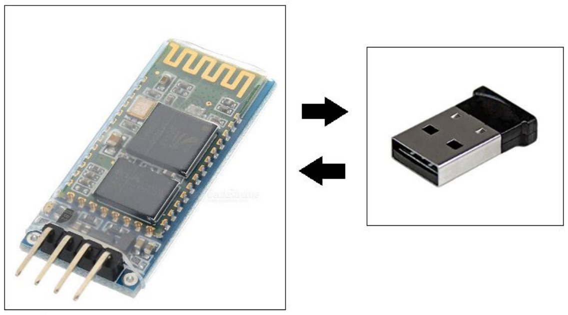
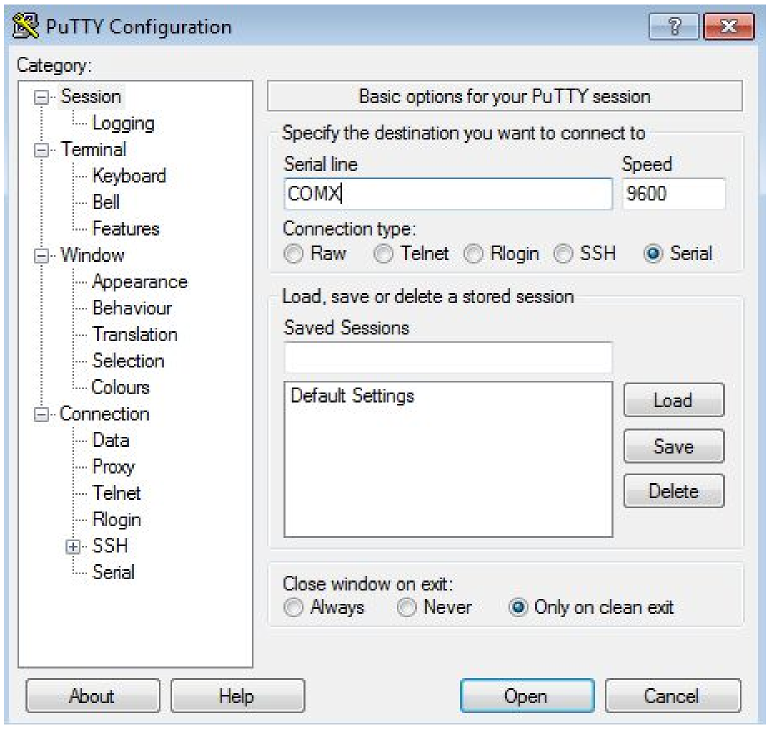


Figure 12: Bluetooth interface components[3-4]

The wireless communication between the robot and the PC is achieved by the wireless link via a bluetooth module and a bluetooth USB Dongle.One of MBED UART module is used to send/receive data serially from/to MBED via the JY-MCU Bluetooth module.(Data rate = 9600 bits/s,Max data rate: 6.25 Mbit/s.,8-bit data,1 stop bit,no parity)[5].The communication line is verified by pairing the bluetooth dongle,that is connected to PC,with the JY-MCU bluetooth module that is connected to MBED.The received data from the bluetooth module is sent to PC through USB and with PuTTY Serial terminal monitor,the COM(X)[1] port is listened.

[1] The COMM port number defined as X can be any number assigned by the OS of PC.



Specify the destination you want to connect to

Serial line: COMX

Speed: 9600

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Figure 13: PuTTY Terminal Screenshot

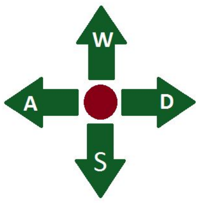
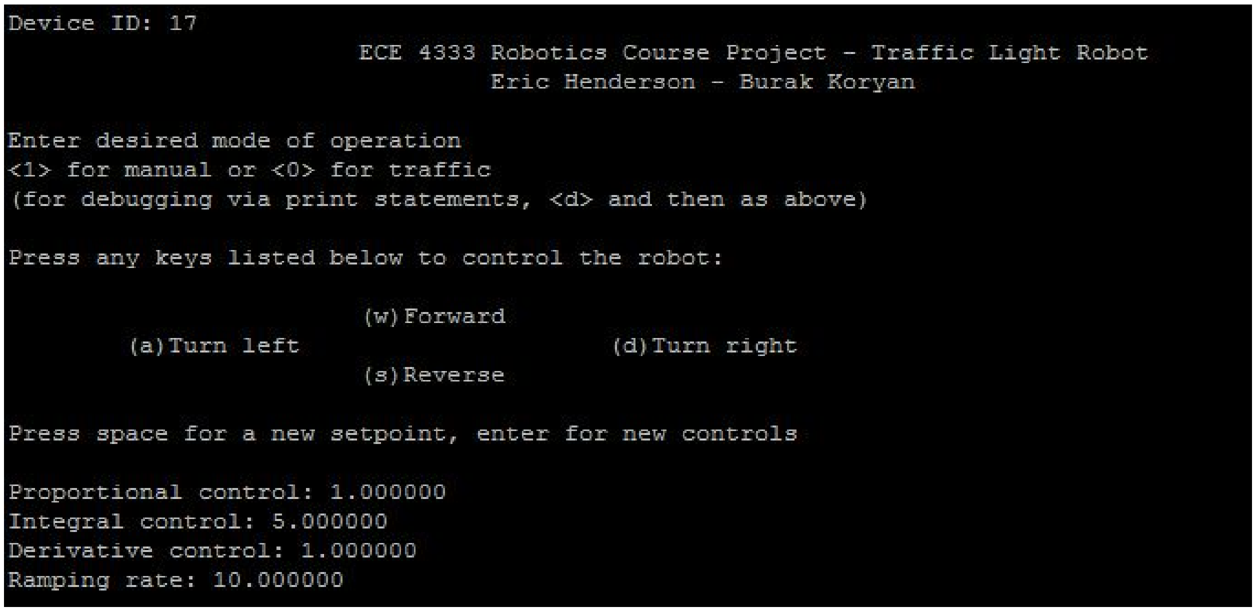


Figure 14: Manual control mode

When the manual operation mode is selected,if "W" is pressed on PC's keyboard while PuTTY terminal is open(and selected with mouse) the terminal sends 'W' character serially to the bluetooth dongle and the character is received by the bluetooth module to MBED.The robot is directed to go forward via its control algorithm.Similarly for when the key S is pressed on the keyboard,the robot is directed to go on reverse (make a U-turn 180 degrees).When the 'A' is pressed,the robot makes a left turn and when 'D' is pressed,a right turn.



```
Device ID: 17
                        ECE 4333 Robotics Course Project - Traffic Light Robot
                                 Eric Henderson - Burak Koryan

Enter desired mode of operation
<1> for manual or <0> for traffic
(for debugging via print statements, <d> and then as above)

Press any keys listed below to control the robot:

                        (w)Forward
        (a)Turn left                     (d)Turn right
                        (s)Reverse

Press space for a new setpoint, enter for new controls

Proportional control: 1.000000
Integral control: 5.000000
Derivative control: 1.000000
Ramping rate: 10.000000
```

Figure 15: Manual control mode user interface on PuTTY

Figure 15 shows how the user interface (UI) is seen on PuTTY. The device number of the Altera-DE0 FPGA board is shown on top of the UI as well as the introduction of the UI.The user is first asked to enter the desired mode of operation as either "1" or "0" for manual or traffic mode control. The system parameters proportional,integral and derivative control and ramping rate are entered by the users after pressing enter and the robot can be moved with w-a-s-d keys as desired.

System Integration:

Both subsystems (aside from manual control) lead into one another in this design, meaning that no toggling between the two is necessary. Subsystem 2, which approaches an observed mark, must be completed before Subsystem 1 is entered.

This method of switching between control system is done based on certain conditions that are specified within the code. As the general idea of this project is to develop a traffic light following robot, the robot must approach the “lights” and when it is suitably close- 20 centimeters away in the current version of the design- it will switch to following the instruction given by the “lights”.

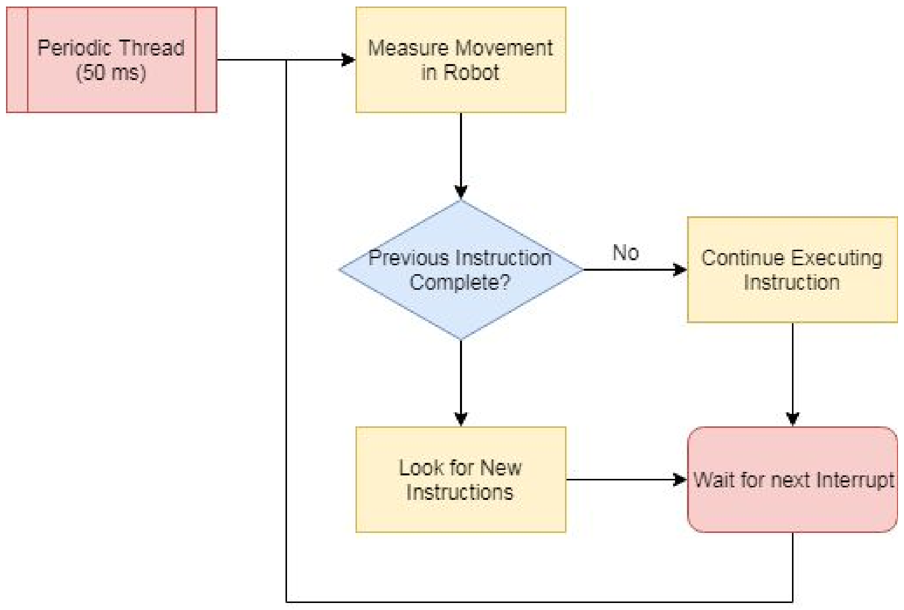

Periodic Thread (50 ms)
Measure Movement in Robot
Previous Instruction Complete?
No
Continue Executing Instruction
Look for New Instructions
Wait for next Interrupt

Figure 16: *Outer Loop Control Flow*

In Table 4 that follows, recommended control parameters are listed.

Parameter	Value
Kp	2
Ki	5
Kd	1
Ramping Rate	15

Table 4: *Recommended Control Parameters for Manual Mode*

Subsystem 1: Traffic Light Instructions

The first of the subsystems implemented is a traffic light instruction following control system. This control system will interpret an observed color combination through the PixyCam as an instruction, and when suitably close to the marking, it will execute the command corresponding to the specific combination of colors.

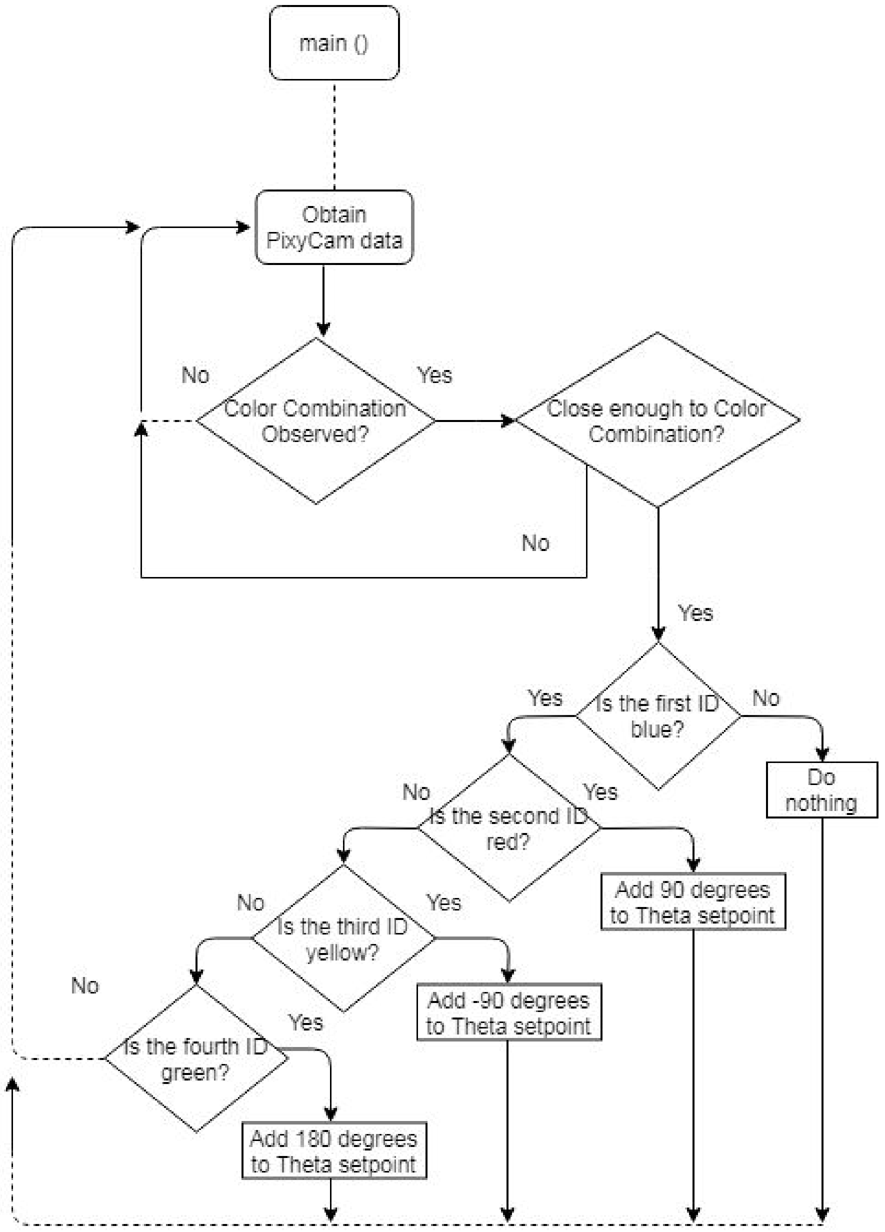

main ()
Obtain PixyCam data
No
Yes
Color Combination Observed?
Close enough to Color Combination?
No
Yes
Yes
Is the first ID blue?
No
Do nothing
No
Is the second ID red?
Yes
Add 90 degrees to Theta setpoint
No
Is the third ID yellow?
Yes
Add -90 degrees to Theta setpoint
No
Is the fourth ID green?
Yes
Add 180 degrees to Theta setpoint

Figure 17: Traffic Light Instruction Flow Diagram

The setpoint for the angular orientation of the robot will be increased or decreased in accordance to the instruction observed. This can go positive or negative, and the outer loop controller for angle setpoint will generate angular speeds to achieve the desired angle. This outer loop controller is a PID, which will continue to control the angle until it stabilizes with an error of less than positive or negative 5 degrees for 100 loops through the outer loop controller. A high level block diagram of the outer loop controller for this subsystem can be seen below in Figure 18.

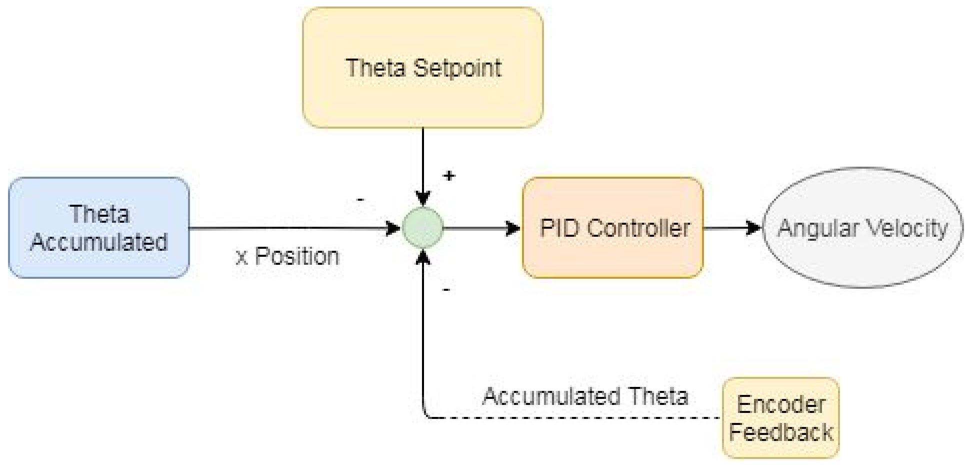


Figure 18: *Block Diagram of Outer Loop Control for Traffic Light Instruction Subsystem*

The timing of this outer loop controller is based on a periodic timer, in the same style as that of the inner loop controller. This periodic timer has a period ten times that of the inner loop controller to allow for the inner loop controller to reach its setpoints before having an updated setpoint. This design choice was made after issues were encountered in the stability of the overall system.

This subsystem prevents integrator windup in the same way the inner loop controller does. This is done by setting a limit on the speed of the motors, and if exceeded, not building up the integrator. In this way, the motors can easily switch directions if instructed to by the outer loop controller.

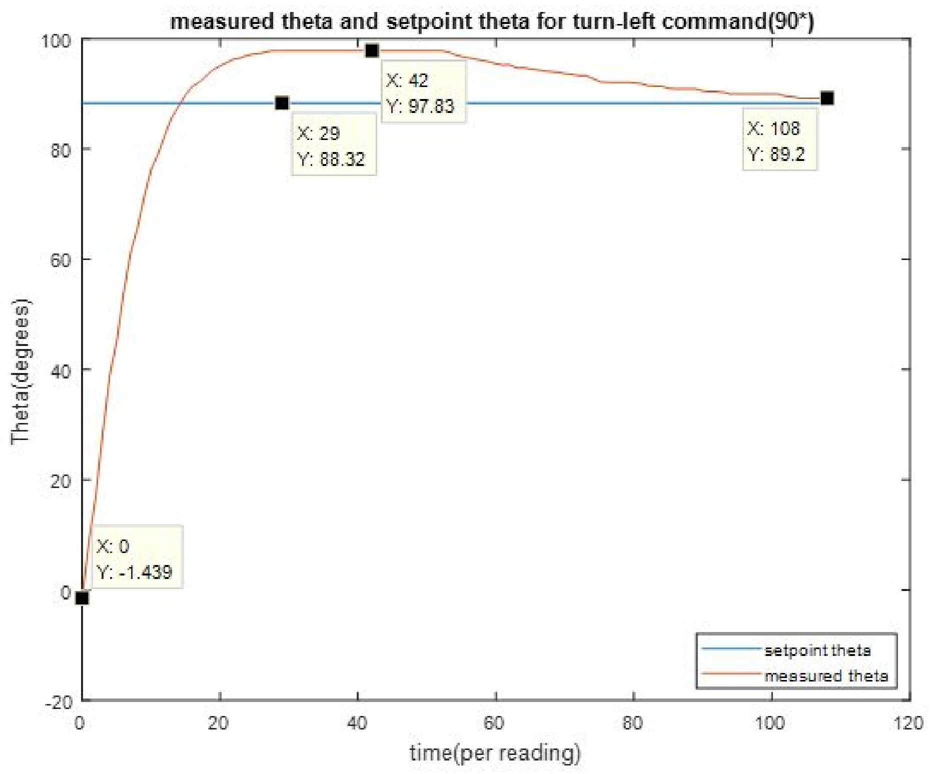


Figure 19: *Response to a 90° Command Observed*

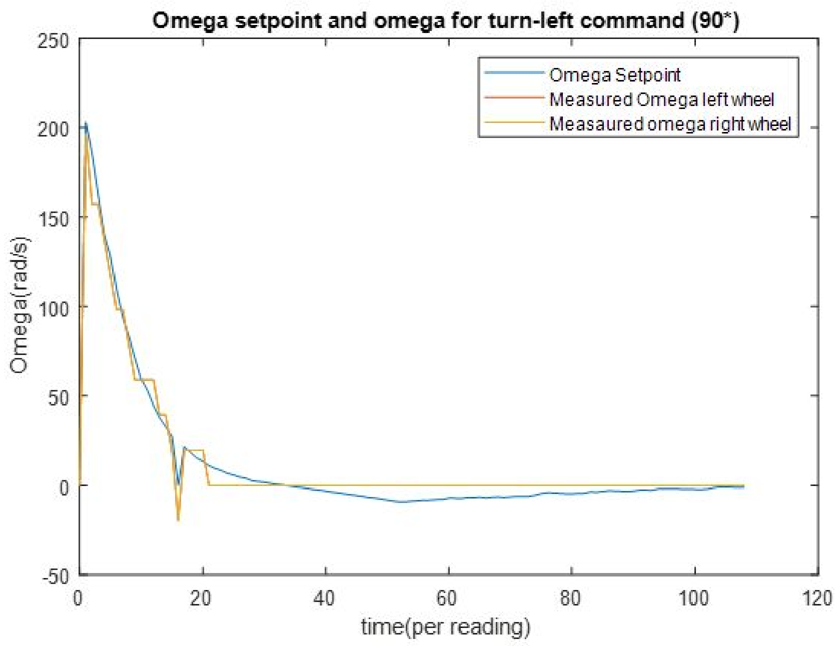


Figure 20: *Motor Speeds with Negated Left Wheel Reading*

Due to the nature of the orientation of the motors, the reading for the angular velocity of the left motor is negative to that of the right motor when moving forward. As such, with a left turn as seen above in Figure 20 (for a 90° increase in angular setpoint), the motor speeds will read as the same direction, despite them moving in opposite directions in a single observed reference frame.

This is accounted for in the calculation of accumulated change in angle, but the raw measured angular velocity of the motors is maintained to remove confusion in the inner loop control system.

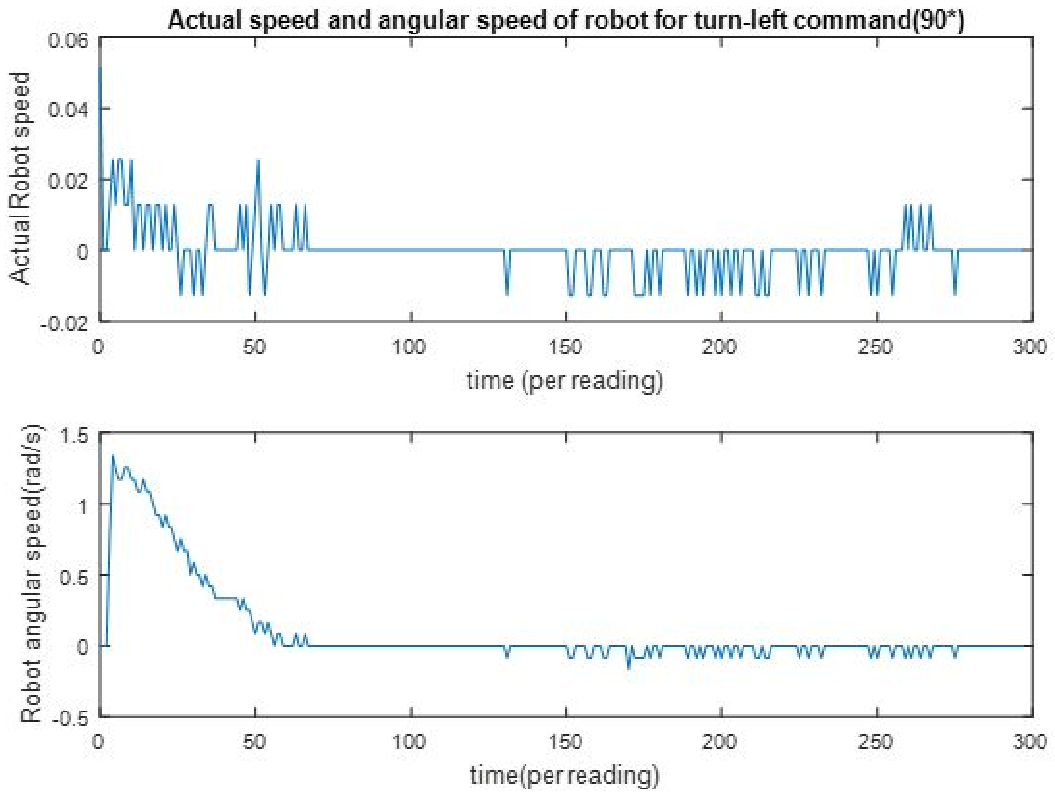


Figure 21: *Overall Robot Speeds for a 90° Turn*

In the above figures, the angle is seen to settle on the setpoint of 88.32°. This not being exactly 90° is due to a small deviation in setpoint as the robot "approached" the mark, which will be discussed in the following subsystem section. The setpoint of 88.32° is 90° away from the original starting angle setpoint.

There is a fairly small overshoot present in the response, 9.51°, which settles down to within ±1° in only approximately 100 measurements, corresponding to a time of ((100 measurements)*(50 milliseconds)) 5 seconds.

The response to both a -90° instruction and a 180° instruction are included in the appendix at the end of this document. Both of these display very similar responses to those seen above.

Subsystem 2: Traffic Light Approach

The second of the subsystems implemented is a traffic light approach control system. This control system will interpret an observed color combination through the PixyCam, and if it is recognized, will determine the distance from the observed marking and its position in the field-of-view. With this data, it will approach the marking by setting angular and forward velocities.

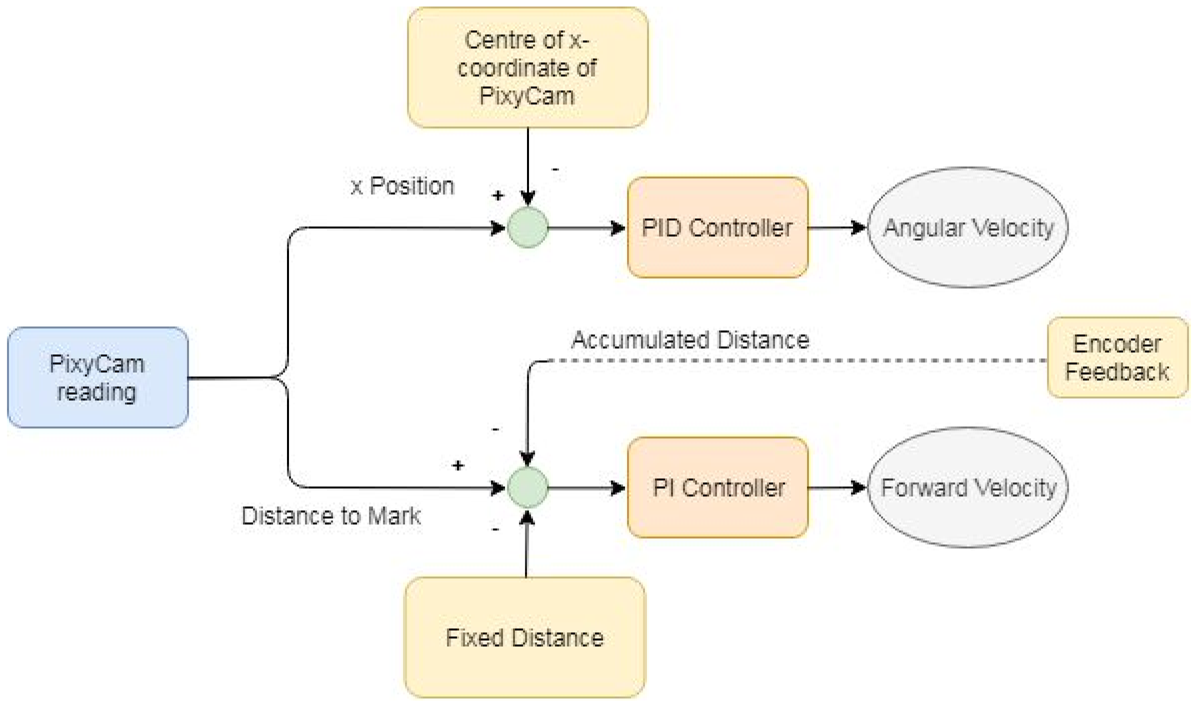
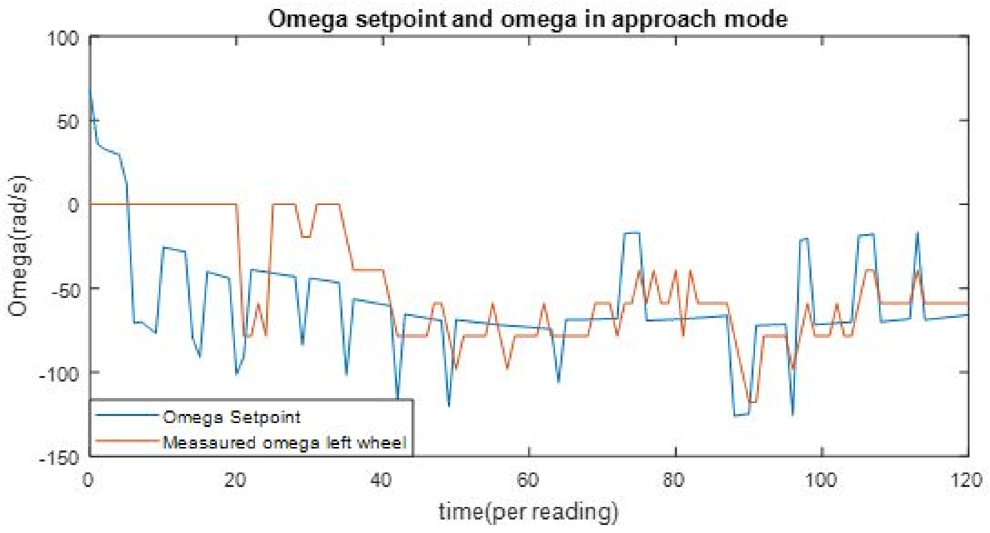


Figure 22 : *High Level Block Diagram for Outer Loop Control in Traffic Light Approach*

The heading error, which controls the angular velocity in this mode, is based on the center position of the marking observed in the field of view. This position is used in conjunction with the measured distance from the marking to determine the approximate error in orientation of the robot in degrees. The PID controller used is similar to that of the Traffic Light Instruction subsystem, aside from that it does not have a single setpoint to follow. Instead, this subsystem increases the orientation setpoint as the robot lines itself up with the marking. This ensures when an instruction is received, it will still move the appropriate amount.



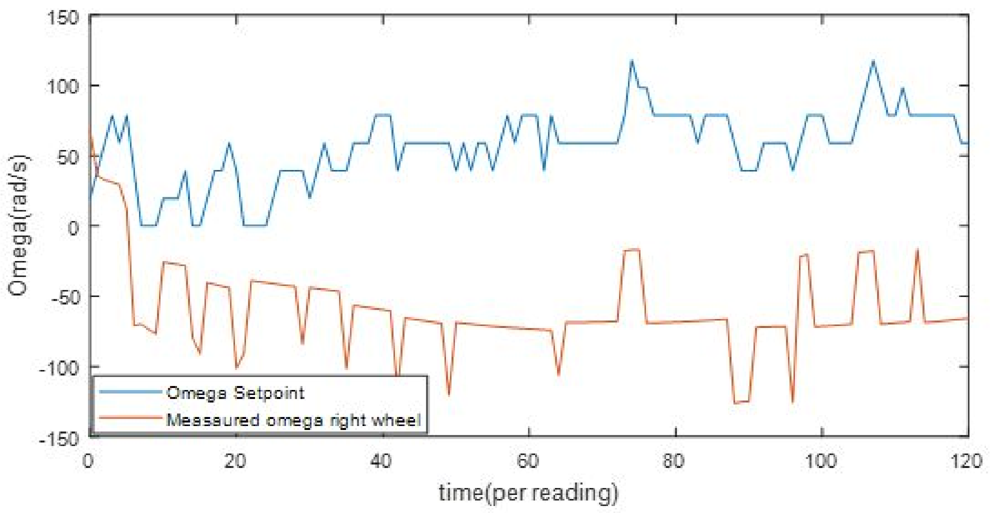


Figure 23: *Omega setpoint and measured omega of robot in approach mode*

In Figure 23, the omega setpoint (in rad/s) and measured omega (in rad/s) of the robot is shown.The left wheel of the robot has a similar flow with the omega setpoint and the right wheel has somewhat an opposite characteristic.

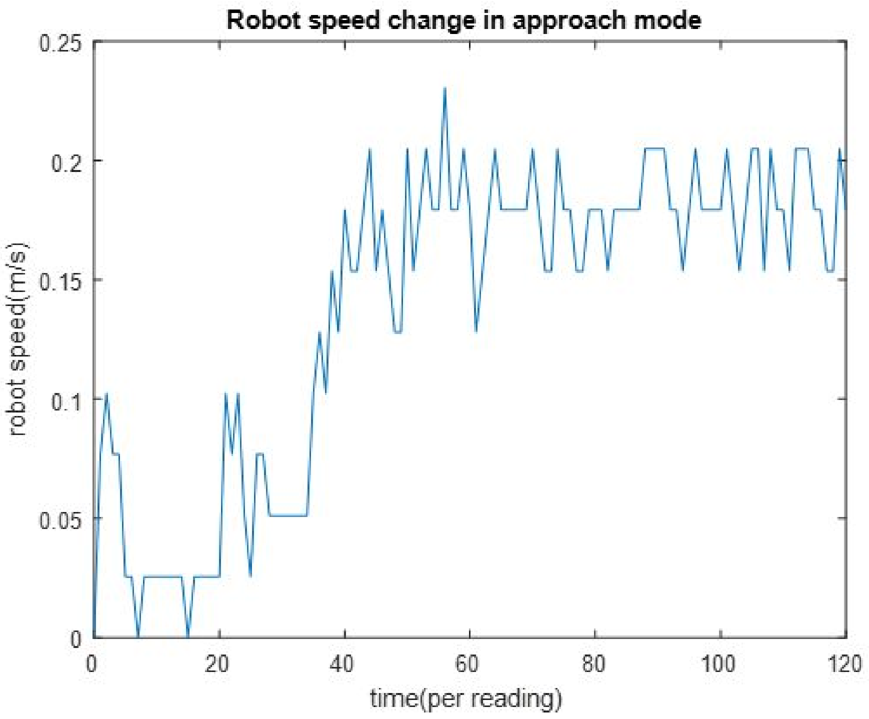


Figure 24: *Robot speed change in approaching color code block*

The robot starts at 0 m/s speed and increases its velocity while approaching the target color block.Since the robot cannot hold a constant velocity rate and angular position facing to the target,the velocity always changes with small increments and decrements that cause the continuous change in the plot.

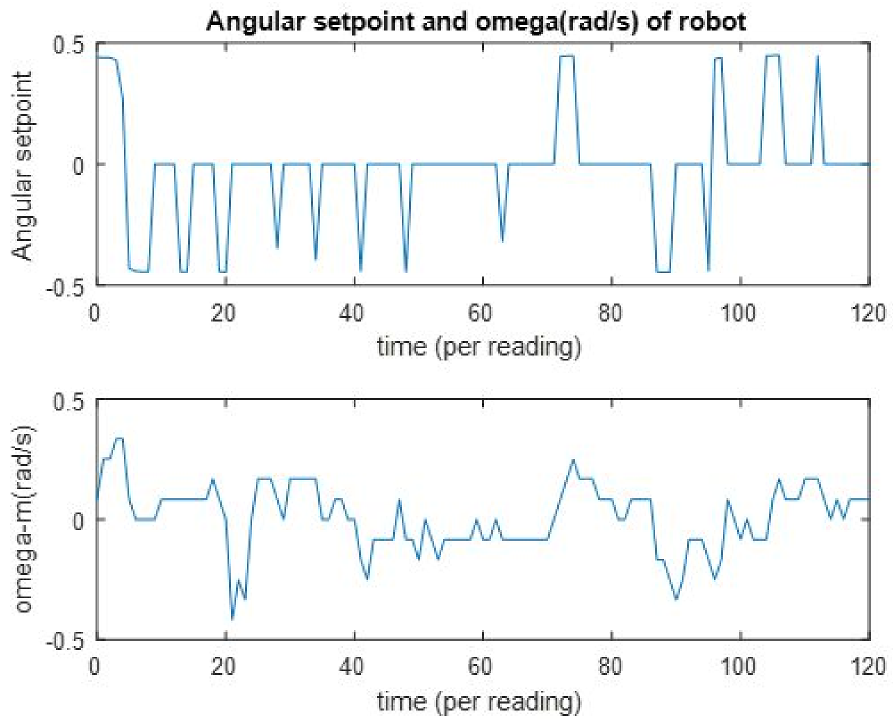


Figure 25: *Angular setpoint and omega (rad/s) of robot*

Similarly,the angular setpoint and overall omega (rad/s) of the robot is not constant while approaching a target.The omega changes between 0 rad/s and -0.5 rad/s or 0 rad/s and +0.5 rad/s continuously during approach to the target in order to centre the robot position toward the target.

Manual Control

For the manual control, as previously stated in the user interface section, W, A, S and D keys are used to move the robot forward, left, back and right, respectively. This is done by incrementally increasing the velocity setpoints, both angular and forward, each time one of the keys are pressed.

If no key is pressed, the instruction given to the robot is cleared by a watchdog timer. This is a precautionary measure in case communication is lost and also for convenience for the user.

A table of recommended parameters that can be set follows in Table 5.

Parameter	**Value**
Kp	0.5
Ki	5
Kd	<Unused in this subsystem>
Ramping Rate	15

Table 5: *Recommended Control Parameters for Manual Mode*

The algorithm diagram in figure 26 shows how the manual control of the robot works. A keystroke is expected from the PC's keyboard and when it happens the keystroke is sent to the MBED via bluetooth communication.The MBED checks whether the pressed key is one of the pre-assigned commands for control of the robot.For instance,if the pressed key is W,then forward velocity,in other words the speed setpoint,of the robot is incremented by 0.05. Similarly, if the key 'S' is pressed then the speed setpoint of the robot is decremented by 0.05 to slow down the robot or go on reverse direction. If a 'D' keystroke is received, to make the robot turn right, the angular setpoint is decremented by 0.05 and if the keystroke is an 'A' then the angular setpoint is incremented by 0.05 to make the robot turn left.

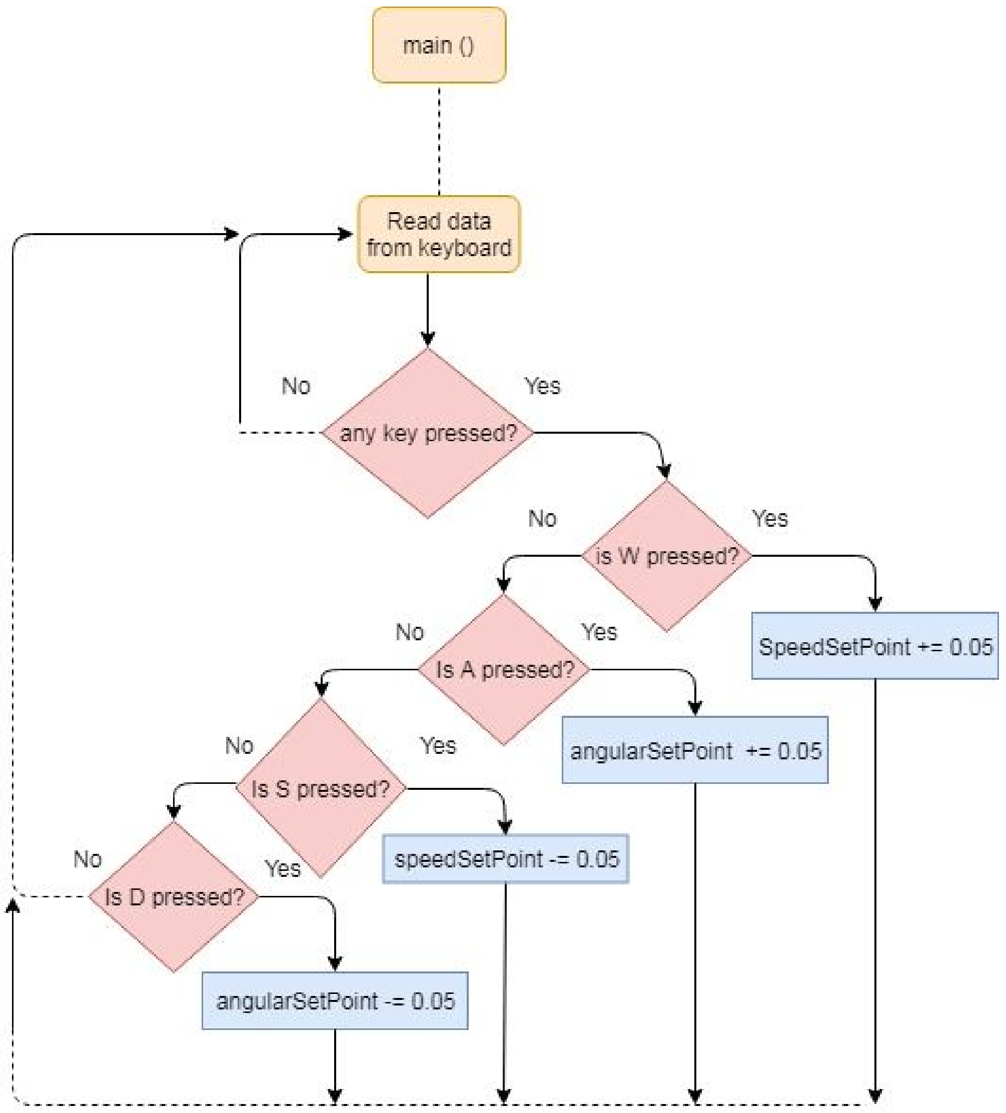


Figure 26: Simple manual control algorithm

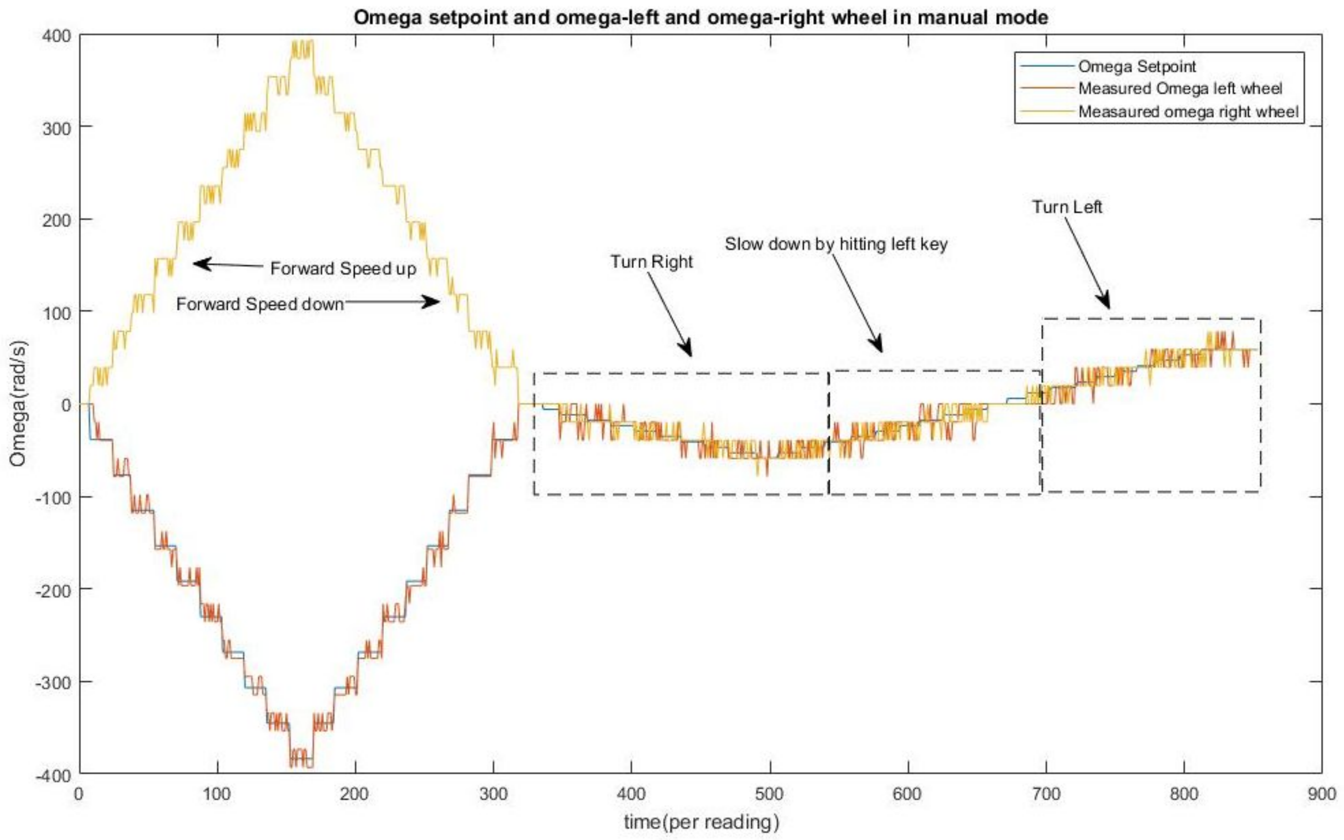


Figure 27: *Omega(rad/s) of robot in manual control mode*

Figure 27 shows omega of the left and right wheel and the omega setpoint versus time per reading. As seen in the annotations,when the forward velocity increases, the omega of the left and right wheels has opposite slopes. As the forward velocity increases the angular speed of the right wheel increases while angular speed of the right wheel decreases. When the forward speed decreases, we see the opposite of mentioned plot characteristic. With increase in right-turn keystores the angular speed of the left wheel is more than the right wheel to make the turn. However, to do the opposite turn, like a left-turn, the right wheel has more angular velocity than the left wheel.

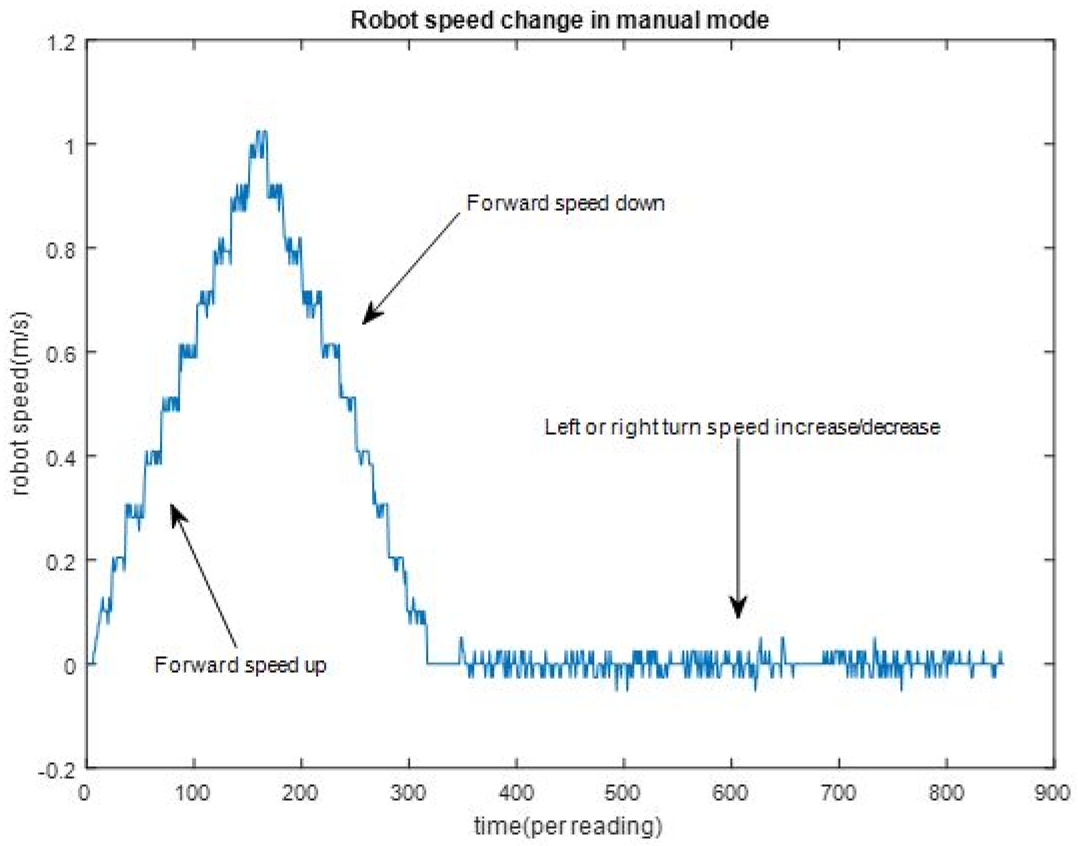


Figure 28: *Robot speed plot in manual mode*

Overall robot speed increases with each forward command as shown in figure 28.The speed increase seems like a step input because of each keystore for increasing or decreasing velocity of the robot.Velocity change in left or right turn in figure 28 cannot be seen significantly because overall velocity of the robot is an average of sum of both left and right wheel velocities.

Conclusions:

The result of this design is a traffic light following robot. Each subsystem working in conjunction will provide a basic autonomous traffic following robot with an additional operating mode where the user can manually control the robot. Each subsystem remains stable under operation, and although it does not have perfect performance, it has the potential to be tuned further to achieve this goal.

When placed on the ground, the parameters recommended for each mode of operation give very little overshoot, and a fast response. These exceeded the initial estimate of performance the group had aimed for.

In the development of this project, an appreciation for controller gains and their actual impact on the operation of a control system was attained. Understanding of multithreaded processes, various practical limitations that are required in real life control systems and many other lessons were learned as well, which resulted in the success of this design.

There is no question that with further development, many issues with the subsystems and improvements could be made. As such, a discussion of future work that would be performed if more time was allowed as been made by each group member and can be seen on the following page.

Future Work:

On a higher level, in order to achieve the overall project goals, the second subsystem would need to be improved on to search the surrounding areas for the marking.

Eric:

If given more time, I would likely push for faster response time in the system and more consistent readings from the PixyCam. The PixyCam posed a constant issue throughout the development of this project, and although many of the issues were overcome, inconsistency in lighting conditions caused issues until the very end. Additionally, tracking a “car” ahead of the robot would be an interesting extension on the second subsystem, and would be another step closer to a proper “self-driving car” type robot design.

Burak:

I usually care about a product's look almost as much as its function.I probably would 3D print a new frame for the robot other than using just a provided metal body frame.This would enable us have more hands-on experience in product design and representation of work would be much more visually appealing.I would try to give more design freedom to students to be more creative for the subsystems.For instance,it would be nice to have two robots interact with one another and behave accordingly.It would also be fun to see two more robots compete with one another one given task as a subsystem.Human-computer interface is an important part of robot technology so it would be much nicer to be more creative in this part of the project as well.Not just visual,but physical feedback from robot,like vibration,would be nice to have,let's say,when the robot turns or stops.

Appendix A: Gamma (Γ) data from PixyCam calibration

Range matrix (m)	Γ matrix(m/pixels)
0.55	0.0021
0.55	0.0021
0.5	0.0019
0.5	0.0019
0.45	0.0018
0.45	0.0018
0.4	0.0016
0.4	0.0015
0.35	0.0014
0.35	0.0014
0.3	0.0012
0.3	0.0012
0.25	0.0009
0.25	0.0011
0.2	0.0008
0.2	0.0008
0.15	0.0006
0.15	0.0006
0.1	0.0004
0.1	0.0004

Table A-1 : Calculated Γ values and range data

Appendix B : Additional Responses for Subsystem 1

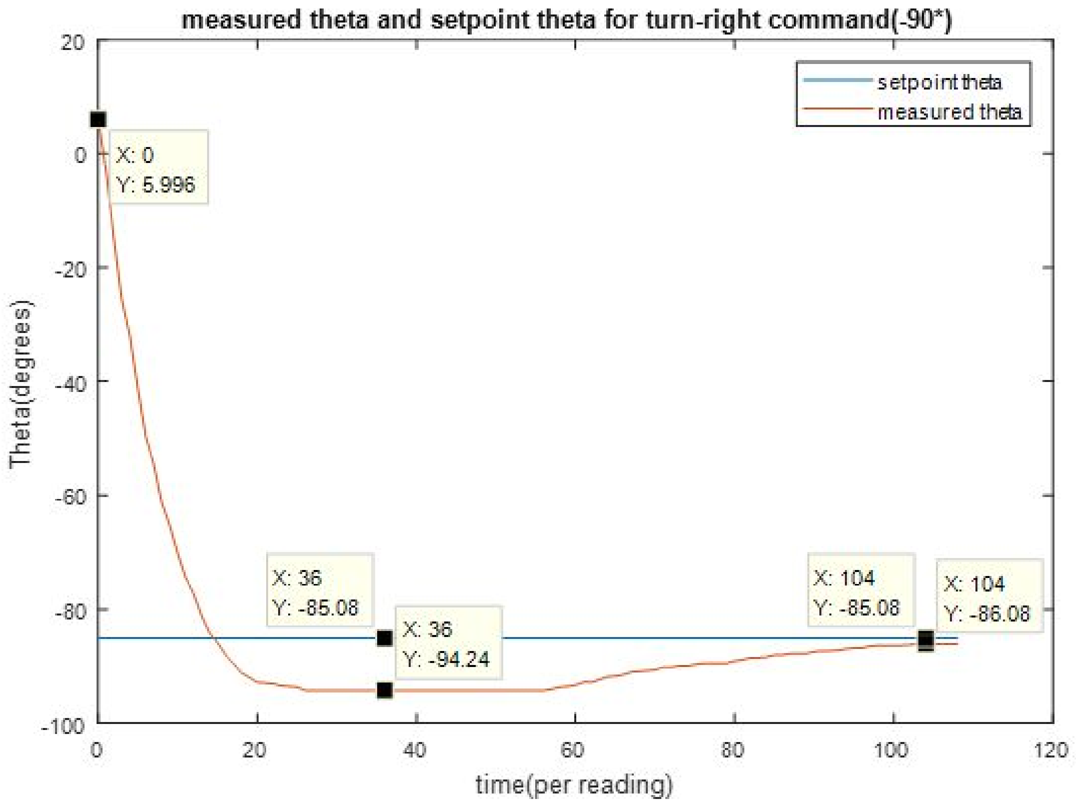


Figure B-1 : Measured theta and setpoint theta in turn-right command

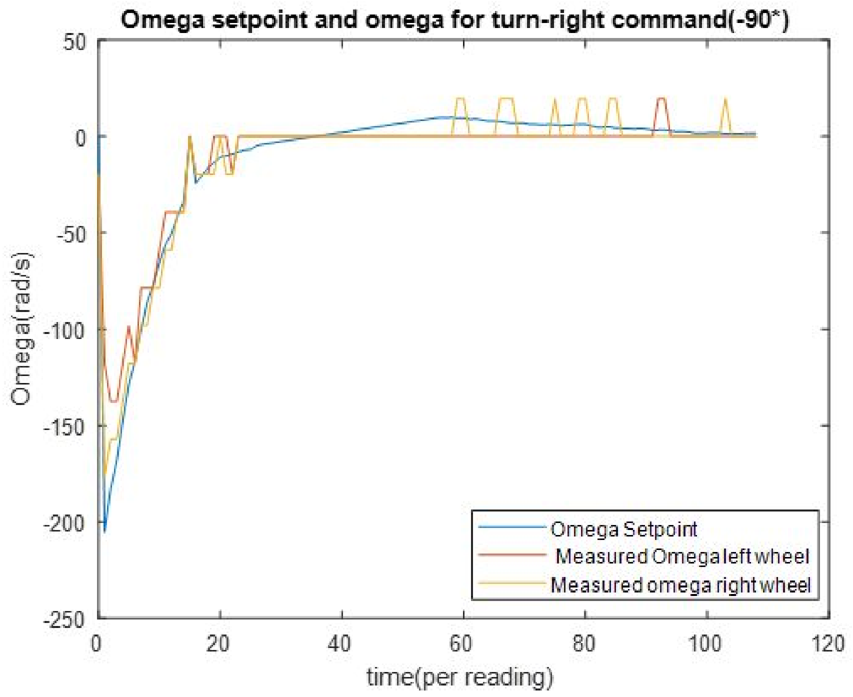


Figure B-2 : Omega setpoint(rad/s) and omega of both wheels(rad/s) in turn-right command

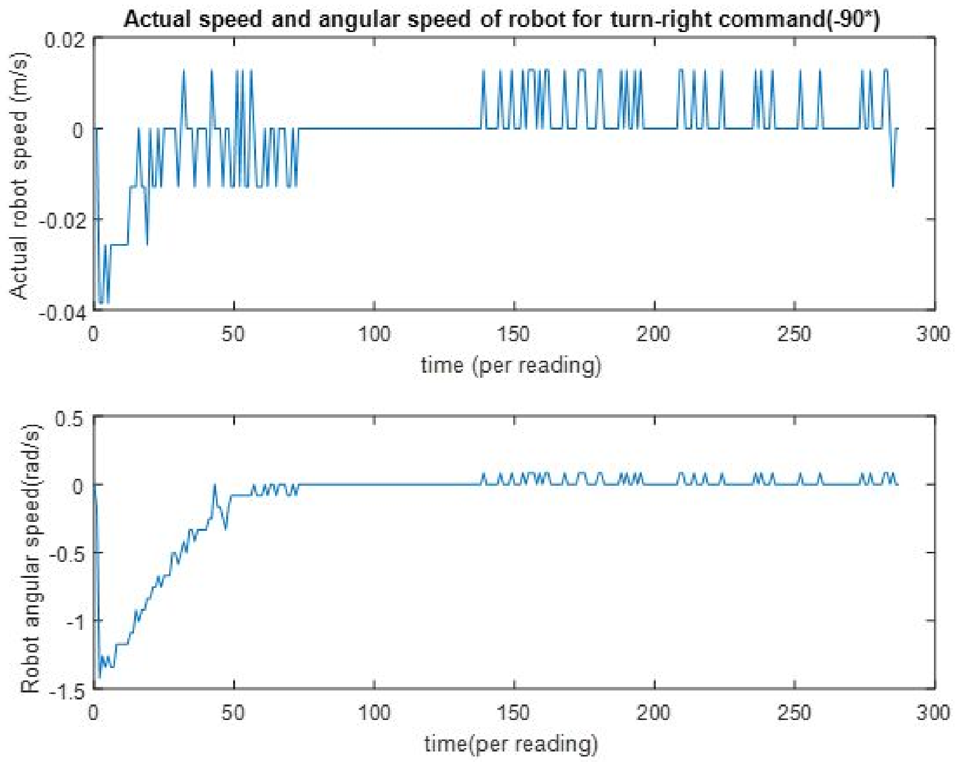


Figure B-3: Overall robot speed (m/s) and angular speed plot in turn-right command

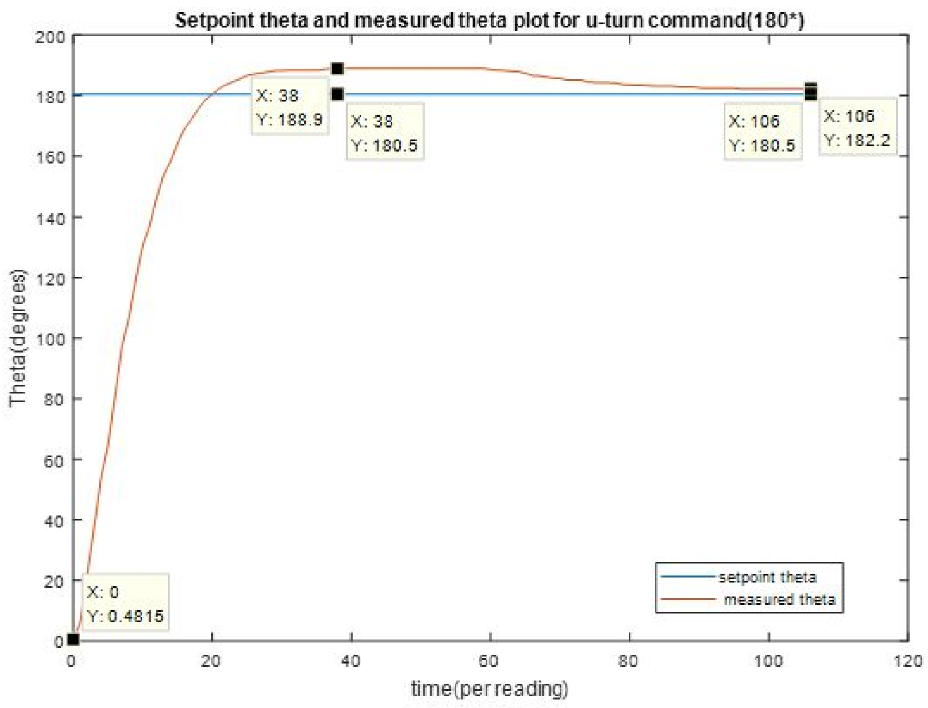


Figure B-4: Angular position of robot in u-turn command

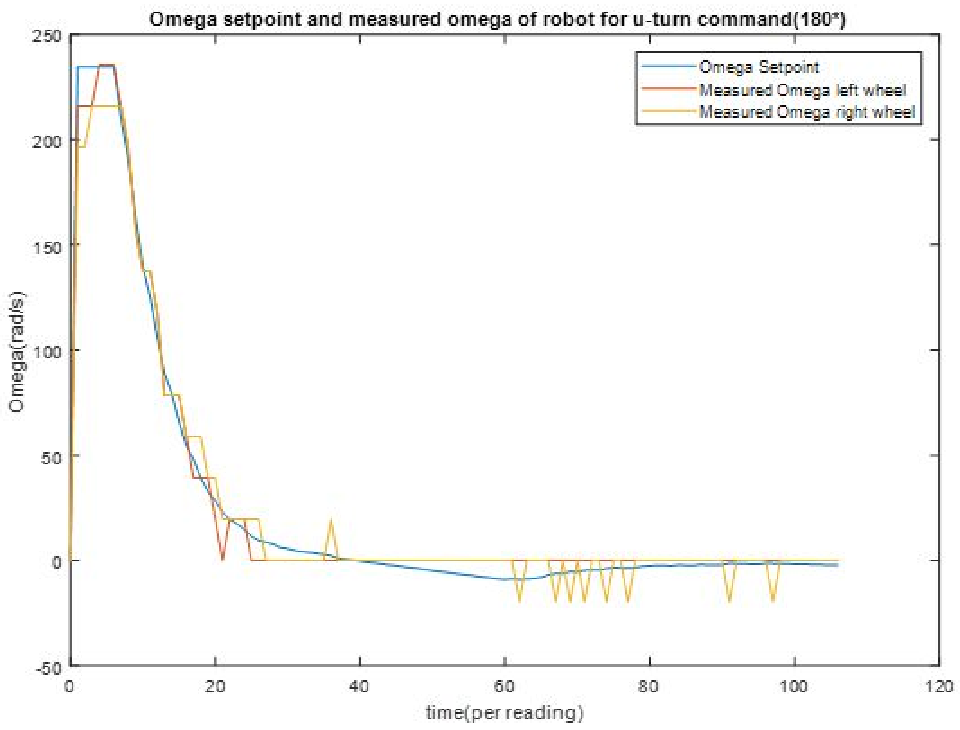


Figure B-4: Omega setpoint (rad/s) and measured omega (rad/s) from both wheels in u-turn command

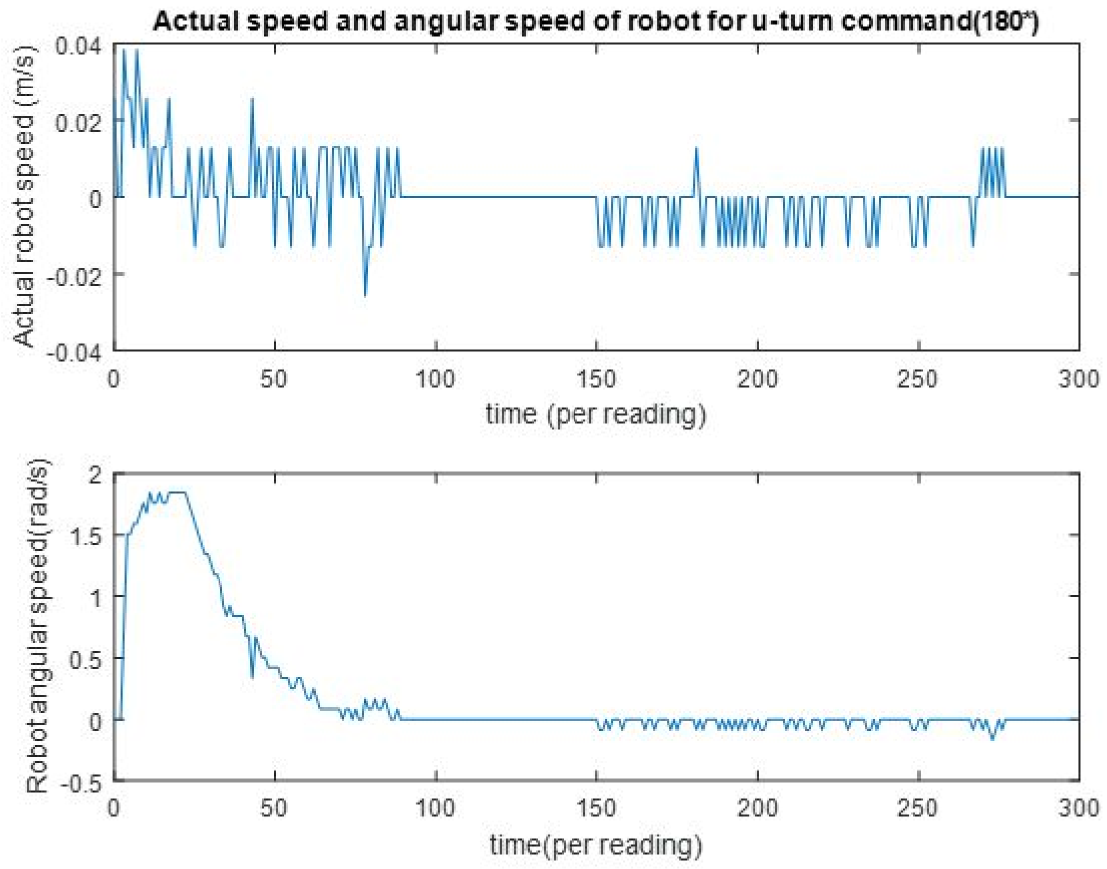


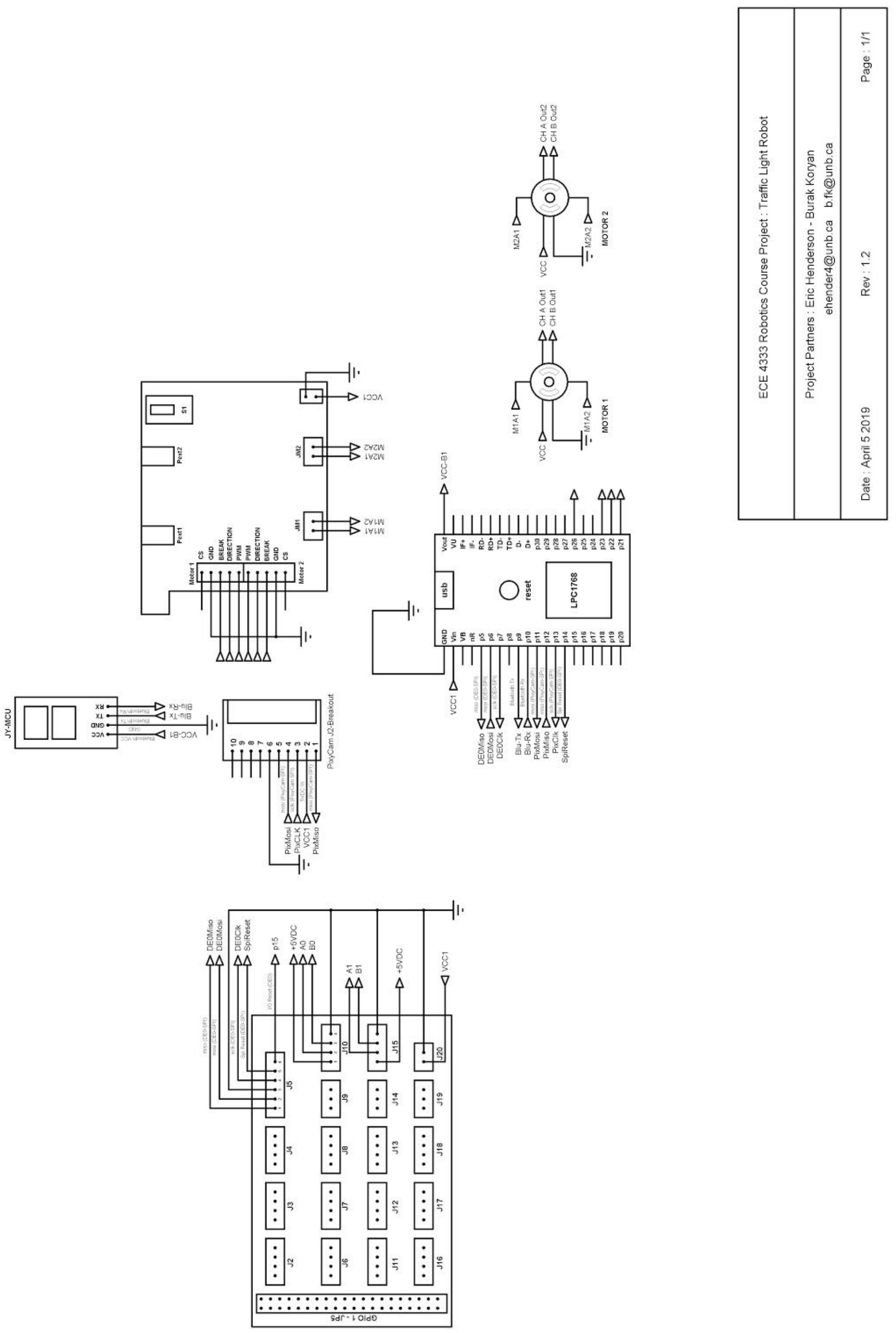
Figure B-5: Overall speed (m/s) and angular speed of robot (rad/s) in u-turn command

Appendix C: Component specifications

PixyCam Specifications[1]		MBED Specifications[2]		Altera DE0 Specifications[3]		UNB H-Bridge Board Specifications[4]	
Processor	NXP LPC4330 204 MHz Dual Core	**Processor**	ARM Cortex-M3 core	**Processor**	Altera Cyclone® III 3C16 FPGA device	**H-Bridge**	LMD 18200
Image Sensor	Omnivision OV9715 ¼”,1280 x 800	**Analog Peripherals**	12-bit ADC with 8 channels and 10-bit DAC	**Analog Peripherals**	3 Pushbuttons, 10 toggle switches,10 green LEDs,VGA DAC	**Bus Transceiver**	74LV245, 3-state
Lens field of view	75 degrees horizontal,47 degrees vertical	**Power Modes**	Sleep Deep-Sleep Power-down Deep power-down	**General I/0**	346 Pins	**Power Input**	26VDC Battery
Lens type	Standard M12	**Crystal Oscillator**	1 MHz to 25 MHz	**Oscillator**	50 MHz		
Power Consumption	140 mA	**Power Consumption**	< 500 mA	**Power Consumption**	-	**Power Consumption**	-
Power Input	Power input: USB input (5V) or unregulated input (6V to 10V)	**Power Input**	5VDC (USB Powered)	**Power Input**	7.5VDC Wall-mount power supply		
Ram & Flash	264K bytes,1M bytes	**Ram & Flash**	512 kB Flash,64 kB data memory	**Ram & Flash**	8MByte SDRAM 4MByte Flash		
Available data outputs	UART,SPI, I^2C,USB	**Available data outputs**	Ethernet, USB, UART, CAN, SPI, I^2C	**Available Data outputs**	RS-232 Transceiver,PS/2 Mouse/Keyboard Two 40-pin Headers		
Dimensions	2.1" x 2.0" x 1.4	**Dimensions**	5.5 cm x 3.0 cm	**Dimensions**	13 cm x 10 cm	**Dimensions**	11 cm x 5.5 cm
Weight	27 grams	**Weight**	-	**Weight**	-	**Weight**	-

Table C-1 : Robot component specifications

Appendix D : Project Schematic



Appendix E : PixyCam calibration Matlab code

```
%{
ECE 4333 - Robotics project PixyCam calibration data
Created by : Eric Henderson | Burak Koryan
Date : April 6 2019

Variable Descriptions:
x & y : Coordinates of recognized color blcok
w & h : actual width and height of the color block
r : distance of the color block to PixyCam
MeansW : An array of mean of widths for all measurements from 55cm to 10cm
MeansH : An array of mean of widths for all measurements from 55cm to 10cm
HW : 20x1 matrix (it is the gamma in the given formula)
fpinV : pseudo inverse of Gamma
%}

clc;
clear;
w = 8e-2;       % width of color blocks. 4cm x 2 blocks = 8cm width
h = 4e-2;       % height of color blocks 4 cm

%% distance r = @55 cm
r = 55e-2;
x = [111 176 168 163 163 163 163 163 163 160];
y = [132 135 136 135 135 136 135 135 135 135];
width = [35 37 39 43 37 38 37 36 38 37];
height = [19 19 19 19 19 20 19 18 19 19];
meanx55 = mean(x);
meany55 = mean(y);
meanW55 = mean(width);
meanH55 = mean(height);

%% distance r = @50cm
r = 50e-2;
x = [150 150 150 150 150 150 150 151 151 150];
y = [132 132 132 131 132 131 131 131 131 132];
width = [43 43 43 41 41 42 43 44 44 42];
height = [21 22 21 21 22 21 21 21 21 22];
meanx50 = mean(x);
meany50 = mean(y);
meanW50 = mean(width);
meanH50 = mean(height);

%% distance r =  @45 cm
```

```
r = 45e-2;
x = [166 165 164 165 164 165 165 166 165 165];
y = [129 129 129 129 129 129 129 129 129 129];
width = [46 46 47 45 48 45 45 46 45 44];
height = [23 23 23 23 22 23 23 23 22 23];
meanx45 = mean(x);
meany45 = mean(y);
meanW45 = mean(width);
meanH45 = mean(height);

%% distance r = @40 cm
r = 40e-2;
x = [155 156 155 155 155 156 154 156 155 155];
y = [76 76 76 76 77 76 76 76 76 76];
width = [51 54 52 53 53 54 51 52 52 52];
height = [25 25 25 25 26 25 24 25 25 25];
meanx40 = mean(x);
meany40 = mean(y);
meanW40 = mean(width);
meanH40 = mean(height);

%% distance r = @35 cm
r = 35e-2;
x = [ 177 176 178 180 179 177 177 176 177 178];
y = [ 120 120 119 120 119 120 119 120 120 120];
width = [57 55 59 54 51 58 58 57 58 53];
height =[29 30 29 30 29 30 29 29 30 30];
meanx35 = mean(x);
meany35 = mean(y);
meanW35 = mean(width);
meanH35 = mean(height);

%% distance r = @30 cm
r = 30e-2;
x = [167 170 169 170 171 171 169 170 171 169];
y = [114 114 113 113 113 113 113 113 113 112];
width = [70 69 71 67 67 69 70 68 68 67];
height = [35 34 35 35 34 35 35 35 34 35];
meanx30 = mean(x);
meany30 = mean(y);
meanW30 = mean(width);
meanH30 = mean(height);

%% distance r = @25 cm
r = 25e-2;
x = [201 200 200 201 200 200 200 200 201 200];
y = [130 130 129 129 131 131 131 130 131 130];
```

```
width = [74 75 76 76 76 76 76 76 76 78];
height = [43 43 43 43 42 42 43 43 42 43];
meanx25 = mean(x);
meany25 = mean(y);
meanW25 = mean(width);
meanH25 = mean(height);

%% distance = @20 cm
r = 20e-2;
x = [202 201 202 202 203 203 203 203 203 203];
y = [87 87 86 86 87 84 87 87 86 86];
width = [98 99 93 93 93 94 95 93 92 92];
height = [50 51 48 48 51 44 51 50 48 48];
meanx20 = mean(x);
meany20 = mean(y);
meanW20 = mean(width);
meanH20 = mean(height);

%% distance = @15 cm
r = 15e-2;
x = [ 215 216 216 215 216 216 216 215 215 216];
y = [ 83 83 83 83 83 83 83 83 83 83];
width = [127 128 126 127 128 127 126 127 128 127];
height = [67 67 67 67 67 67 67 67 67 67];
meanx15 = mean(x);
meany15 = mean(y);
meanW15 = mean(width);
meanH15 = mean(height);

%% distance = @10 cm
r = 10e-2;
x = [188 188 188 188 187 188 188 187 187 189];
y = [48 48 48 48 48 48 48 48 48 48];
width = [186 186 186 188 186 185 186 187 185 186];
height = [91 90 90 90 90 90 91 91 90 91];

meanx10 = mean(x);
meany10 = mean(y);
meanW10 = mean(width);
meanH10 = mean(height);


%% calculation of f
range = [55 50 45 40 35 30 25 20 15 10].*10e-2;
MeansW = [meanW55 meanW50 meanW45 meanW40 meanW35 meanW30 meanW25 meanW20 meanW15
meanW10];
MeansH = [meanH55 meanH50 meanH45 meanH40 meanH35 meanH30 meanH25 meanH20 meanH15
meanH10];
HW = zeros(20,1);
HW =
[h./meanH55,w./meanW55,h./meanH50,w./meanW50,h./meanH45,w./meanW45,h./meanH40,w./me
anW40,h./meanH35,w./meanW35,h./meanH30,w./meanW30,h./meanH25,w./meanW25,h./meanH20,
w./meanW20,h./meanH15,w./meanW15,h./meanH10,w./meanW10];
r = [55e-2 55e-2 50e-2 50e-2 45e-2 45e-2 40e-2 40e-2 35e-2 35e-2 30e-2 30e-2 25e-2
25e-2 20e-2 20e-2 15e-2 15e-2 10e-2 10e-2];
f =(1./(transpose(HW).*HW)).*(transpose(HW).*r);
fPinV = r*pinv(HW);

%% Plot measured means of width-height and range data with best-fit lines

polFit = polyfit(MeansW,range,3);
polVal = polyval(polFit,MeansW);
plot(MeansW,range);hold on;
plot(MeansW,polVal);hold off;
ylabel('range(m)')
xlabel('psi(pixel)')
title('Plot of range(m) versus psi(pixel)')
disp(polFit)

figure;
polFit = polyfit(MeansH,range,3);
polVal = polyval(polFit,MeansH);
plot(MeansH,range);hold on;
plot(MeansH,polVal);hold off;
ylabel('range(m)')
xlabel('lambda(pixel)')
title('Plot of range(m) versus lambda(pixel)')
disp(polFit)
```

Appendix F: TrafficLightRobot.cpp

```
#include "mbed.h"
#include "rtos.h"
#include "Pixy.h"

#define SLEEP_TIME                  500 // (msec)
#define PRINT_AFTER_N_LOOPS         20
#define PIXY_MAX_SIGNATURE          7

// PIXYCAM COLORS:
// 1 - BLUE
// 2 - RED
// 3 - YELLOW
// 4 - GREEN

#define SLEEP_TIME                  500 // (msec)
#define PRINT_AFTER_N_LOOPS         20

Serial pc(USBTX, USBRX); // tx, rx
Serial bluetooth(p9, p10); // TX on pin 9 and RX on pin 10
SPI DE0(p5, p6, p7); // mosi, miso, sclk (BRK)

void PeriodicInnerLoopISR(void);
void PeriodicOuterLoopISR(void);
void WatchdogISR(void const *n);
void InnerLoopThread(void const *argument);
void OuterLoopThread(void const *argument);
void WatchdogThread(void const *argument);

// Processes and threads
//int32_t SignalInnerLoop, SignalOuterLoop;
osThreadId PeriodicInnerId;
osThreadDef(InnerLoopThread, osPriorityRealtime, 1024); // Declare InnerLoopThread as
a thread/process
osThreadId PeriodicOuterId;
osThreadDef(OuterLoopThread, osPriorityHigh, 1024); // Declare OuterLoopThread as a
thread/process
osThreadId WatchdogId;
osThreadDef(WatchdogThread, osPriorityHigh, 1024); // Declare WatchdogThread as
thread/process

osTimerDef(Wdtimer, WatchdogISR);
osTimerId OneShot = osTimerCreate(osTimer(Wdtimer), osTimerOnce, (void *)0);

Ticker PeriodicInnerInt; // Declare a timer interrupt: PeriodicInnerInt
Ticker PeriodicOuterInt;
PwmOut PwmLeft(p23); // PWM output for the left motor
PwmOut PwmRight(p26);// PWM output for the right motor
DigitalOut IoReset(p15);    // Resets all spi prepherials on the DE0
DigitalOut directionLeft(p22),directionRight(p25); // motor direction pins
DigitalOut brakeLeft(p21),brakeRight(p24); // motor brake pins
DigitalOut led3(LED3), led2(LED2), led4(LED4);
DigitalOut SpiReset(p14); // DE0 SPI reset

int outerLoopFlag = 0;

float T = 1000; // PWM period (us)
float h = 0.005; // periodic interrupt period (s)
float PWMdutyCycle;

float L = 0.306; // axis length of the robot (used in kinematic calculations)
float vRight, vLeft, vLeft_m , vRight_m ; // motor velocities (setpoint and measured)

int ID, counter = 0;; // ID returned from DE0 SPI read
int16_t dPosition1,dPosition2; // change in position of the motors from last read
(DE0)
unsigned short dTime1,dTime2,Time1,Time2; // change in time since last read (DE0)
float Position1,Position2; //
float rampVal1,rampVal2,dRamp;
float omegaL_m,omegaR_m, omegaL,omegaR; // angular velocity of motors (setpoint and
measured)
float omega_m = 0; // overall robot angular velocity
float Position, Theta, PrevThetaError; // Pos and Angle of robot and error from last
loop
float speedSetpoint,storedSpeedSetpoint,angularSetpoint,thetaSetpoint; // setpoints
for control
float eLeft=0,eRight=0; // inner loop (individual motor) errors
float xiLeft=0, xiRight=0, xiAngle = 0, kp, ki, kd; // control gains and accumulators
float uLeft, uRight, outputLeft, outputRight; // PWM outputs
int instructionComplete = 1;
uint16_t x,y,width,height, blocksR;
float distToBlock = 0, distSetpoint = 0.20, headingErrorPrev = 0, xiPos, posError;
float accumulatedDist = 0, xiTheta = 0, dTheta = 0, stabilityFlag = 0, vRobot;

Mutex r_mutex; // declaration of mutex

SPI spiR(p11, p12, p13); // Create Pixy SPI port (mosi, miso, sclk)
PixySPI pixyR(&spiR, &bluetooth); // Select SPI interface for PixyCam

char sig[6]; // global cc signature string

// Uses code from TPixy.h, adapted from code written by swilkins8
void blockName(int idx){ // stores CC name in global char sig[6]
    int i, j;
    char d;
    bool flag;
    i++;
    for (i = 12, j = 0, flag = false; i >= 0; i -= 3) {
        d = (pixyR.blocks[idx].signature >> i) & 0x07;
        if (d > 0 && !flag) {
            flag = true;
        }
        if (flag) {
            sig[j++] = d + '0';
        }
    } // end for loop
    sig[j] = '\0';
} // end blockName()


void newcommand(float angle){
    bluetooth.printf("\n\rNew Command Observed: %f\n\r",angle);

    r_mutex.lock();
    storedSpeedSetpoint = speedSetpoint;
    instructionComplete = 0;
    angularSetpoint = 0;
    speedSetpoint = 0;
//    r_mutex.unlock();
    Thread::wait(5000); // stops robot and waits briefly before executing command
//    r_mutex.lock();
    thetaSetpoint = thetaSetpoint + angle; // adds new command to theta setpoint
    xiPos = 0;
    r_mutex.unlock();
    sig[0] = '\0';
} // end newcommand()

void controlAdjust(void){
    //float distToCenter = 999; // will only give new command if this is updated
    float angle;
    if(sig[0] == '1' and sig[1] == '2'){ // Blue and Red
//        distToCenter = centerSignature();
        angle = 90; // turns robot 90 degrees (left turn)
    }
    else if(sig[0] == '1' and sig[1] == '3'){ // Blue and Yellow
//        distToCenter = centerSignature();
        angle = -90; // turns robot -90 degrees (right turn)
    }
    else if(sig[0] == '1' and sig[1] == '4'){ // Blue and Green
//        distToCenter = centerSignature();
        angle = 180; // turns the robot 180 degrees (left turn around)
    }
    //if((distToCenter < 5 and distToBlock <= distSetpoint and instructionComplete ==
1)){
    if((distToBlock <= distSetpoint and instructionComplete == 1)){
        newcommand(angle); // only executes command when the robot is close enough
                           // and when it is centered on the "traffic light"
                           // AND when the prev instruction is complete
    }

}// end controlAdjust()

void PixyCamRead(void)
{
    static int i = 0;
    int j;

    blocksR = pixyR.getBlocks(); // number of blocks
    //bluetooth.printf("getting blocks..\n\r");
    if (blocksR) {
        //bluetooth.printf("blocks here..\n\r");
        if (i % 50 == 0 ){
            //bluetooth.printf("\n\rDetected blocksR %d:\n\r", blocksR);
            //bluetooth.printf(buf);

            for (j = 0; j < blocksR; j++) {
                blockName(j); // gets the name of the currently observed sig
                //bluetooth.printf(" blockR %d: \n\r", j);
                //bluetooth.printf(buf);
                //pixyR.printBlock(pixyR.blocks[j]);
                if((sig[0] == '1') and (sig[1] == '2' or sig[1] == '3' or sig[1] ==
'4')){
                    r_mutex.lock();
                    x = pixyR.blocks[j].x;
                    y = pixyR.blocks[j].y;
                    width = pixyR.blocks[j].width;
                    height = pixyR.blocks[j].height;
                    distToBlock = ((((8.0/width)+(4.0/height))*257.2969)/2.0)/100.0;
// using f from calibration (which is in cm)

// also converts to meters
                    accumulatedDist = 0;
                    r_mutex.unlock();
                    //bluetooth.printf("\n\r [%d,%d] position of CC %s is %f
away\n\r",x,y,sig,distToBlock);
                    controlAdjust(); // adjusts controls based on observed sig
                    osTimerStart(OneShot, 2000); // Start or restart the watchdog
timer interrupt and set to  2000ms.
                    led4 = 0;
                    break;
                }
            }// end for
        }// end if
    }// end if

} // end PixyCamRead()

int main(){
    brakeRight = 0;
    brakeLeft = 0;

    PwmLeft.period_us(T); // PWM period to 2.5 ms
    PwmLeft.pulsewidth_us(0); // Initialize to zero
    PwmRight.period_us(T); // PWM period to 2.5 ms
    PwmRight.pulsewidth_us(0); // Initialize to zero

    DE0.format(16,1);     // 16-bit data,mode: 1 (https://os.mbed.com/handbook/SPI)
(BRK)
    DE0.frequency(1000000);  // SPI bit rate: 5 MHz (BRK)
    IoReset = 0;
    IoReset = 1;
    wait_us(5);
    IoReset = 0;
    SpiReset = 0;
    SpiReset = 1;
    wait_us(5);
    SpiReset = 0;

    ID = DE0.write(0x8002);
    bluetooth.printf("\n\rDevice ID: %x",ID);

    // Initialization of Globals
    Position1 = 0;Position2 = 0; Time1 = 0; Time2 = 0; //ii = 0;
    rampVal1 = 0; rampVal2 = 0; xiLeft = 0; xiRight = 0, xiPos = 0;

    Position = 0; Theta = 0; // Resetting position of robot

    omega_m = 0; speedSetpoint = 0; angularSetpoint = 0; thetaSetpoint = 0;
    PrevThetaError = 0;

    char newcontrol = 0;
    float tempkp, tempki, tempkd, tempdRamp;
    float tempSpeedSetpoint, tempAngularSetpoint;

    int debugFlag = 0;
    bluetooth.printf("\r\n\t\t\tECE 4333 Robotics Course Project - Traffic Light
Robot");
    bluetooth.printf("\r\n\t\t\t\t Eric Henderson - Burak Koryan\n\r");
    bluetooth.printf("\r\nEnter desired mode of operation\r\n<0> for traffic or <1>
for manual");
    bluetooth.printf("\r\n(for debugging via print statements, <d> and then as
above)\n\r");
    while(true){
        if(bluetooth.readable()){
            outerLoopFlag = bluetooth.getc();
            //bluetooth.printf("%c\n\r",outerLoopFlag);
            if(outerLoopFlag == '1'){
                bluetooth.printf("\n\rPress any keys listed below to control the
robot:\n\r");
                bluetooth.printf("\n\r\t\t\t(w)Forward\n\r \t(a)Turn left \t\t\t
(d)Turn right \n\r \t\t\t(s)Reverse\n\r");
//                bluetooth.printf("... but a flag is implemented to take out outer
loop control\n\r");
                break;
            }
            else if(outerLoopFlag == '0'){
                break;
            }
            else if(outerLoopFlag == 'd'){debugFlag = 1;bluetooth.printf("...debug
mode active...\n\r");}
            else{bluetooth.printf("Invalid command, try again\n\r");}
        }
    }

    WatchdogId = osThreadCreate(osThread(WatchdogThread), NULL);
    PeriodicInnerId = osThreadCreate(osThread(InnerLoopThread), NULL);
    // Specify address of the PeriodicInnerInt ISR as PiControllerISR, specify the
interval
    // in seconds between interrupts, and start interrupt generation:
    PeriodicInnerInt.attach(&PeriodicInnerLoopISR, h);

    // if in manual mode, don't enable the outer loop controller
    if(outerLoopFlag == '0'){
        PeriodicOuterId = osThreadCreate(osThread(OuterLoopThread), NULL);
        // Specify address of the PeriodicOuterInt ISR as PiControllerISR, specify
the interval
        // in seconds between interrupts, and start interrupt generation:
        PeriodicOuterInt.attach(&PeriodicOuterLoopISR, h*10);
    }

    pixyR.init();


    bluetooth.printf("\r\nPress space for a new setpoint, enter for new
controls\n\r");
    while(true){
        PixyCamRead(); // reading from the sensor (the PixyCam)
        if(bluetooth.readable()){
            newcontrol = bluetooth.getc();
            if(newcontrol == ' '){ // get new setpoints
                bluetooth.printf("\r\nEnter the desired forward velocity in m/s: ");
                bluetooth.scanf("%f", &tempSpeedSetpoint);
                bluetooth.printf("%f", tempSpeedSetpoint); // echo
                bluetooth.printf("\r\nEnter the desired angular velocity in rad/s:
");

                bluetooth.scanf("%f", &tempAngularSetpoint);
                bluetooth.printf("%f\n\r", tempAngularSetpoint); // echo

                // Mutex setting variables
                r_mutex.lock();
                speedSetpoint = tempSpeedSetpoint;
                angularSetpoint = tempAngularSetpoint;
                r_mutex.unlock();
            }
            if(newcontrol == '\r'){ // get new controls
                bluetooth.printf("\n\rProportional control: ");
                bluetooth.scanf("%f",&tempkp);
                bluetooth.printf("%f",tempkp);
                bluetooth.printf("\n\rIntegral control: ");
                bluetooth.scanf("%f",&tempki);
                bluetooth.printf("%f",tempki);
                bluetooth.printf("\n\rDerivative control: ");
                bluetooth.scanf("%f",&tempkd);
                bluetooth.printf("%f",tempkd);
                bluetooth.printf("\n\rRamping rate: ");
                bluetooth.scanf("%f",&tempdRamp);
                bluetooth.printf("%f\n\r",tempdRamp);
                r_mutex.lock();
                rampVal1 = 0;
                rampVal2 = 0;
                kp = tempkp;
                ki = tempki;
                kd = tempkd;
                dRamp = tempdRamp;
                r_mutex.unlock();
            }
            if(outerLoopFlag == '1'){ // only do this if in manual mode
                if(newcontrol == 'w'){
                    //bluetooth.printf("\n\r W Pressed : Speeding up\n\r");
                    r_mutex.lock();
                    speedSetpoint = speedSetpoint + 0.05;
                    osTimerStart(OneShot, 5000); // Start or restart the watchdog
timer interrupt and set to  2000ms.
                    angularSetpoint = 0;
                    led4 = 0;
                    r_mutex.unlock();
                }
                if(newcontrol == 's'){
                    //bluetooth.printf("\n\r S Pressed : Slowing down\n\r");
                    r_mutex.lock();
                    speedSetpoint = speedSetpoint - 0.05;
                    osTimerStart(OneShot, 5000);
                    led4 = 0;
                    r_mutex.unlock();
                }
                if(newcontrol == 'a'){
                    //bluetooth.printf("\n\r A Pressed : Turning left\n\r");
                    r_mutex.lock();
                    angularSetpoint = angularSetpoint + 0.05;
                    osTimerStart(OneShot, 5000);
                    led4 = 0;
                    r_mutex.unlock();
                }
                if(newcontrol == 'd'){
                    //bluetooth.printf("\n\r D Pressed : Turning right\n\r");
                    r_mutex.lock();
                    angularSetpoint = angularSetpoint - 0.05;
                    osTimerStart(OneShot, 5000);
                    led4 = 0;
                    r_mutex.unlock();
                }
            }
        }// end new user input if()


        counter = counter + 1;
        if((counter >= 10) and (debugFlag == 1)){ // only print every 10th loop
            //r_mutex.lock();
            bluetooth.printf("%f, %f, ",omegaL,omegaR);
            bluetooth.printf("%f, %f, ",omegaL_m,omegaR_m);
            //bluetooth.printf("%f, %f  | ",eLeft,eRight);
            //bluetooth.printf("%f, %f  | ",eLeft,eRight);
            bluetooth.printf("%d, %f, ",x,distToBlock);
            //bluetooth.printf("%f, %f, %f, %f,
%f\n",rampVal,Theta,e,output/T*100,xi);
            bluetooth.printf("%f, %f, ",thetaSetpoint, Theta);
            bluetooth.printf("%f, %f, ", speedSetpoint, angularSetpoint);
            bluetooth.printf("%f, %f    ",omega_m, angularSetpoint);
            bluetooth.printf("%f, %f\n\r",vLeft_m, vRight_m);
            //r_mutex.unlock();
            counter = 0;
            bluetooth.printf("\n\r");
        }// end print thing

        Thread::wait(25);

    }// end while(True)
} //end of main

void trafficLightMode(void){
    float ThetaError, dTerm;

    ThetaError = thetaSetpoint - Theta; // error in orientation
    dTerm = (ThetaError - PrevThetaError)/(h*10); // derivative term for controller

    if((abs(ThetaError)) < 5){stabilityFlag = stabilityFlag + 1;} // error is less
than 5
    else{stabilityFlag = 0;}

    if(stabilityFlag >= 100){ // Instruction is only complete when 100 time steps
                            // read that the error is less than 5 degrees
        speedSetpoint = storedSpeedSetpoint;
        instructionComplete = 1;
        distToBlock = 0; // clearing previous so it doesn't keep moving
        x = 0;          // same as above
        osTimerStart(OneShot, 2000);
        bluetooth.printf(".....Finished Command.....\n\r");
    }

    r_mutex.lock();
    angularSetpoint = (ThetaError*(kp/1) + xiTheta*(ki/15) + dTerm*kd/100)/100.0; //
new angular setpoint
    if(abs(angularSetpoint) > 2){angularSetpoint =
(angularSetpoint/abs(angularSetpoint))*2;} // maintains direction
    else{xiTheta = xiTheta + ThetaError*(h*10);}
    r_mutex.unlock();

    PrevThetaError = ThetaError;
}

void approachMark(void){
    float headingError, dTermHeading;

    // Measuring the distance between the setpoint and the robot
    // (accumulatedDist away from the observed traffic light)
    posError = (distToBlock - 0.01) - accumulatedDist - distSetpoint;
    // Only track the movement in theta with the heading error if an actual
    // mark is observed- this prevents the robot from drifting off without seeing it
    if(blocksR){
        // calculating error in "heading" or the angle made with the distance to the
object and the
        // x position in pixel space
        headingError = -atan2((x-160),distToBlock)*(180.0/3.1415); // 160 is the
center x coordinate
        // ** in reviewing this code before submission, I now see that this angle
would not be correct
        //    as the x-160 term is in pixels, instead of meters as the distToBlock
is... changing this would
        //    improve the oscillation seen in the approach significantly
        dTermHeading = (headingError - headingErrorPrev)/(h*10);
        headingErrorPrev = headingError;
    }
    else{headingError = 0; dTermHeading = 0;}

    thetaSetpoint = thetaSetpoint + dTheta;

    r_mutex.lock();

    speedSetpoint = (posError*(kp/5) + (ki/15)*xiPos)/5.0;
    angularSetpoint = (headingError*kp + dTermHeading*kd/50)/200.0;

    if(abs(speedSetpoint) > 0.5){speedSetpoint =
(speedSetpoint/abs(speedSetpoint))*0.5;} // maintains direction
    else{xiPos = xiPos + posError*(h*10);} // preventing windup
    if(abs(angularSetpoint) > 2){angularSetpoint =
(angularSetpoint/abs(angularSetpoint))*2;} // maintains direction

    r_mutex.unlock();

    headingErrorPrev = headingError;
}

// ******** Outer Loop Periodic Timer Interrupt Thread ********
void OuterLoopThread(void const *argument){
    float omega_m_new;
    while(true){
        osSignalWait(0x1, osWaitForever); // Go to sleep until signal,
SignalOuterLoop, is received.

        r_mutex.lock();
        omega_m_new = (vRight_m - vLeft_m)/L; // calcs for overall angular velocity
of robot
        dTheta = (h*10)*((omega_m+omega_m_new)/2)*(180.0/3.1415);
        Theta = Theta + dTheta; // accumulates angular movement
        omega_m = omega_m_new;

        vRobot = (vRight_m + vLeft_m)/2; // calcs for overall forward velocity of
robot
        accumulatedDist = accumulatedDist + vRobot*(h*10);
        r_mutex.unlock();
        // if an instruction has been set, do what it tells you
        if((instructionComplete == 0)){trafficLightMode();}
        // ... if not, approach a "traffic light"
        else{approachMark();}
        led2 = !led2;
    }
}

// ******** Inner Loop Periodic Timer Interrupt Thread ********
void InnerLoopThread(void const *argument) {
    while (true) {
        osSignalWait(0x1, osWaitForever); // Go to sleep until signal,
SignalInnerLoop, is received.
        //led3= !led3; // Alive status - led3 toggles each time InnerLoopThread is
signaled.

//        if(outerLoopFlag == '0'){outerLoop();} // only do

        // individual motor velocity calculations
        vLeft = -(speedSetpoint - angularSetpoint*L/2); // left motor is turned
around
        vRight = speedSetpoint + angularSetpoint*L/2;

        // motor velocities to motor angular velocities
        omegaL = vLeft*(1.0/0.05)*(90.0/44.0)*18.75;
        omegaR = vRight*(1.0/0.05)*(90.0/44.0)*18.75;

        if(speedSetpoint == 0 and angularSetpoint == 0){brakeLeft = 1;brakeRight =
1;}
        else{brakeLeft = 0;brakeRight = 0;}

        // Reading Encoder Feedback
        SpiReset = 0;
        SpiReset = 1;
        wait_us(5);
        SpiReset = 0;

        ID = DE0.write(0x8004);

        dPosition1 = DE0.write(0xFF); // send dummy value and read position1
        dTime1 = DE0.write(0xFF);     // send dummy value and read time1
        dPosition2 = DE0.write(0xFF); // send dummy value and read position2
        dTime2 = DE0.write(0xFF);     // send dummy value and read time2

        // angular speed = (change in pos/ change in time) * Scaling
        omegaL_m = ((float(dPosition1)*2.0*3.1415)/64.0)/(float(dTime1)*10.24e-6);
        omegaR_m = ((float(dPosition2)*2.0*3.1415)/64.0)/(float(dTime2)*10.24e-6);


        // velocity of wheel = omega of motor scaled thru gears, to wheel and *
radius
        vLeft_m = -(omegaL_m*(44.0/90.0)*0.05*(1.0/18.75)); // negative b/c motor is
turned around
```

```
        vRight_m = omegaR_m*(44.0/90.0)*0.05*(1.0/18.75);

        // Ramping up
        rampVal1 = rampVal1 + dRamp;
        if(rampVal1 > omegaL){rampVal1 = omegaL;}
        rampVal2 = rampVal2 + dRamp;
        if(rampVal2 > omegaR){rampVal2 = omegaR;}


        // Setting PWM Output
        eLeft = rampVal1 - omegaL_m;   // speed error left wheel
        eRight = rampVal2 - omegaR_m; // speed error right wheel

        uLeft = kp*eLeft + ki*xiLeft;
        uRight = kp*eRight + ki*xiRight;

        outputLeft = abs(uLeft);
        outputRight = abs(uRight);

        if(outputLeft > 0.5*T){outputLeft = 0.5*T;} // 50% is the max duty cycle
        else{xiLeft = xiLeft + eLeft*h;}

        if(outputRight > 0.5*T){outputRight = 0.5*T;} // 50% is the max duty cycle
        else{xiRight = xiRight + eRight*h;}

        if(uLeft > 0){directionLeft = 0;}
        else{directionLeft = 1;}

        if(uRight > 0){directionRight = 0;}
        else{directionRight = 1;}

        // setting PWM outputs
        PwmLeft.pulsewidth_us(outputLeft);
        PwmRight.pulsewidth_us(outputRight);
        if(counter > 0 and counter <= 5){
            led3= !led3;
        }
    }// end while(True)
}// end InnerLoopThread()

void WatchdogThread(void const *argument){
    while(true) {
        osSignalWait(0x1, osWaitForever); // Go to sleep until a signal,
SignalWatchdog, is received
        speedSetpoint = 0; // stops robot
        angularSetpoint = 0;
        led4 = 1; // led4 is activated when the watchdog timer times out
    }
}

// ******** Inner Loop Periodic Timer Interrupt Handler ********
void PeriodicInnerLoopISR(void) {
    osSignalSet(PeriodicInnerId,0x1); // Send signal to the thread with ID,
PeriodicInnerId
```

```
}

// ******** Outer Loop Periodic Timer Interrupt Handler ********
void PeriodicOuterLoopISR(void) {
    osSignalSet(PeriodicOuterId,0x1); // Send signal to the thread with ID,
PeriodicOuterId
}

// ******** Watchdog Interrupt Handler ********
void WatchdogISR(void const *n)
{
    osSignalSet(WatchdogId,0x1); // Send signal to thread with ID, WatchdogId, i.e.,
WatchdogThread
}
```

References:

[1] *Moore-Penrose pseudoinverse*. [Online]. Available: https://www.mathworks.com/help/matlab/ref/pinv.html.

[2] *Lecture 5 : Sensing the Environment*. [Online]. Available: https://lms.unb.ca/d2l/le/content/148798/viewContent/1515831/View.

[3] *USB Bluetooth dongle figure*. [Online]. Available: https://www.amazon.ca/StarTech-com-USBBT1EDR4-160-Feet-Wireless-Bluetooth/dp/B00FCK307I.

[4] *JY-MCU Bluetooth Wireless Serial Port Module figure* . [Online]. Available: https://www.dx.com/p/jy-mcu-arduino-bluetooth-wireless-serial-port-module-2011902#.XK9N_jBKhhE.

[5] *LPC 1768 Datasheet*. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/LPC1769_68_67_66_65_64_63.pdf.