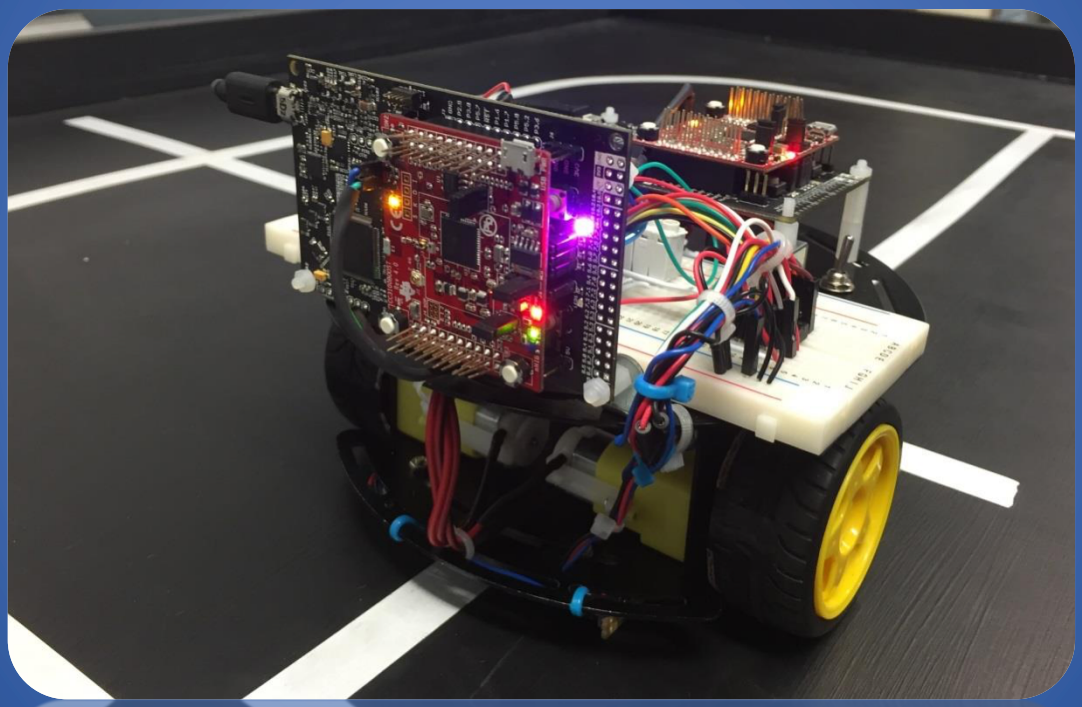
EE303

Mobile Robotics

MR. ROBOT



Group Number	02
Academic Year	2016/17
Submission Date	26/03/2017
Module Coordinator	Dr Patricia Moore

EE303: Assignment: Project Report

Student Names	Programme	ID Number
Sakthignana Sundaram Somaskandan	14346091	ECE3
Mark James Mc Hugh	14755515	ECE3
Radwan Duadu	14342606	ECE3
Ahmed Ibrahim Aly	14326366	ME3

Plagiarism Declaration:

We understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

We have read and understand the DCU Academic integrity and Plagiarism Policy. We accept the penalties that may be imposed should we engage in practice or practices that breach this policy.

We declare that this material, which we now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion, or copying. We have read and understood the Assignment Regulations set out in the module documentation. We have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study we have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

Student Names	Date	Signatures
Sakthignana Sundaram Somaskandan	26/03/2017	Sakthi Somaskandan
Mark James Mc Hugh	26/03/2017	Mark Mc Hugh
Radwan Duadu	26/03/2017	Radwan Duadu
Ahmed Ibrahim Aly	26/03/2017	Ahmed Ibrahim

Table of Contents

List of Figures

List of Tables

Acknowledgements

"Keep away from people who try to belittle your ambitions. Small people always do that, but the really great make you feel that you, too, can become great" by Mark Twain.

It is our radiant sentiment to place on record our best regards, our sense of gratitude to Dr Patricia Moore, Mr Robert Clare and the demonstrators for their careful and precious guidance which were extremely valuable for our project both theoretically and practically. Secondly, bearing in mind we would like to use this opportunity to express our deepest gratitude and special thanks to Dr Patricia Moore who in spite being extraordinarily busy with her duties, took time out to hear, guide and keep us on the correct path and allowing us to carry out our project at her esteemed organization. Forever, we will perceive this opportunity as a big milestone in our career development. We will strive to use gained skills and knowledge in the best possible way and we will continue to work for their improvement, to attain desired career objectives. Looking forward and hope to continue cooperation with all of you in the future!

1. Introduction

This report details the conceptualization and development process involved in how a team of engineers designed a microcontroller-based wheeled robot system that applies the art of IOT (Internet of Technology) technology by allowing a user to start the system through a Wi-Fi enabled portable device to autonomously navigate through the track with a pre-defined destinations acquired from the server. The report outlines how a team of engineers developed the final product including the issues, problems, and limitations encountered during the course of designing the product in a period of six weeks. The final product is then discussed in more detail with regards to the hardware and software. The aim of this project is to enable students to utilize and apply the knowledge gained from semester 1 to semester 4 of engineering and other supporting modules to interrelate sensors and actuators/motors with a microcontroller to enhance the practical experience. The name given to the final product is Mr. Robot.

In the development timeline of the prototype shown in Figure 1, a task must be completed on weekly basis with the last week assigned to incorporate an innovative feature to the mobile robot as follows:

Week 1: Build a basic model of Mr. Robot which go forward and backward using the 2WD mobile platform with the aid of electronic components such as H-Bridge, MSP432 microcontroller and DC-DC converter.

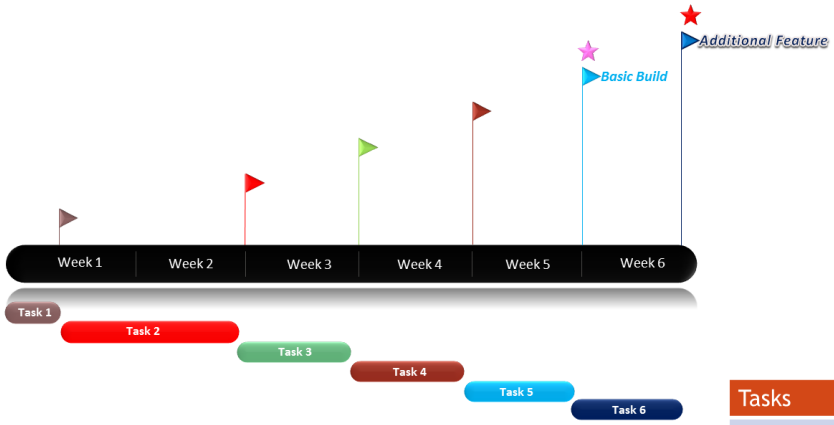
Week 2: Allow Mr. Robot to follow a white line autonomously with the aid of five line following sensors with transistor output board.

Week 3: Initiate Mr. Robot from a smartphone over WIFI with the aid of the CC3100 Booster Pack and play a melody using a piezo buzzer simultaneously.

Week 4: Let Mr. Robot communicate with the server over WIFI by transmitting current group number and position to the server in order to get to the next destination from the server.

Week 5: Parking Mr. Robot away from the white line, as near as possible but without touching the target wall with the aid of IR sensor to detect any obstacles.

Week 6: Integrating an Innovative feature to Mr. Robot that is also complex in order to achieve unique prototype. This additional feature was based on advanced IOT technology.



Tasks	
Task 1	Assembly and Basic Functionality
Task 2	White line Following
Task 3	WIFI Control with a melody
Task 4	Communication with the server
Task 5	Parking near the wall
Task 6	Innovation

Figure 1 - Project timeline

The Hardware involved in the construction of Mr. Robot is discussed in more details with regards to the components utilised, their function, structure and limitations involved. A step by step guide then explains and displays the configuration of the individual components embedded within the circuit construction of Mr. Robot. Additionally, the hardware consists of interconnected electronic/hardware components which work in conjunction with the software to produce a pre-defined visible output.

The software section discusses the programming aspect of Mr. Robot and how the software was interrelated with the hardware. It also makes it possible to grasp how each component is utilised in the most efficient manner. The software sequentially consists of a set of instructions or rules which dictate the operation of the hardware component.

The report also contains a section that discusses the team aspect of the project. This section contains information such as the meeting log used to keep track of the project's progress. While a Gantt chart was also used to show and summarise the work progress carried out during the designing and development process of the product. Project management highlights the importance of teamwork, which was one of the main themes of the project.

2. Development

2.1 Chassis Design

The chassis chosen for this project is the 2WD mobile platform turtle. The robot comes as a kit which includes two drive motors, wheels (and rear caster ball), frame and all mounting hardware. The kit features two differential drives, near zero turning radius, high-strength aluminium alloy body material, motors and a flexible rubber wheel. The top layer includes a standard servo socket and three optical sensor holders. Those three holders fit an adjustable variety of sensors. [1]

The average weight of the 2WD mobile platform is about 445 grams, with a motor weight of 45 grams according to the datasheet. Figure 2 shows the typical dimensions of the 2WD mobile platform. [2]

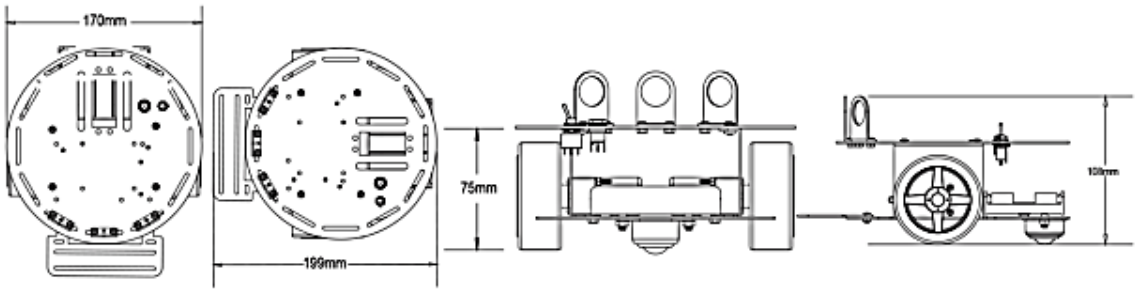


Figure 2 [2] - 2WD Mobile Platform blueprint

The main objective of this chassis is to hold the components of Mr. Robot such as the IR sensor; MSP432P401R etc. Figure 3 shows how the chassis is set up with regards to the drive motors, wheels, top and bottom layers etc.

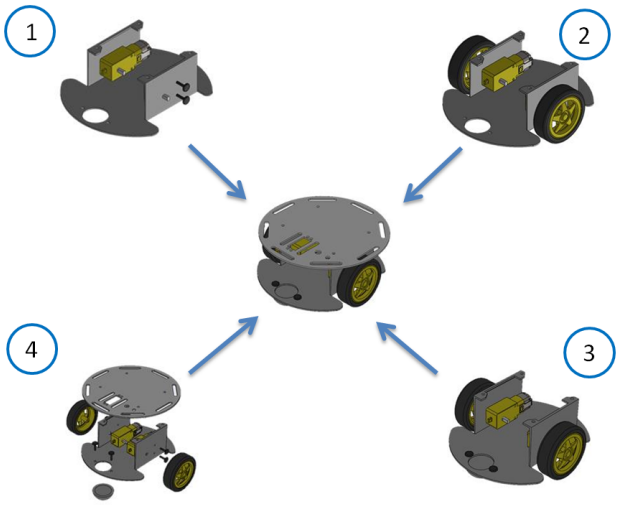


Figure 3 - Set Up Of the 2WD Mobile Platform

The motors of the 2WD Mobile Platform comes with a variety of interesting and complex features and can give different properties when connected to $3V$ or $6V$. Table 1 shows the speed, current, and torque obtained when those motors turned on at $3V$ or $6V$. [2] Note that the motors were operated at $5V$ and hence the acquired value of the feature should be in between $3V$ and $6V$.

Table 1 [2] - Motor Features at different voltage levels

Features	***Available Voltage Sources***	
	3V	***6V***
No-load Speed	$100RPM$	$200RPM$
No-load Current	$60mA$	$71mA$
Stall Current	$260mA$	$470mA$
Torque	$1.2kgcm$	$1.92kgcm$

In the development of the prototype/final product, the team had to plan efficiently where each component is placed on the chassis due to the restricted space provided on the chassis. The first and main component to be placed on the chassis is the MSP432 microcontroller. The team decided that the microcontroller should be mounted vertically in order to effectively save some space for other components and to make the final product to look aesthetically pleasing. Another component, the buzzer was placed on the top layer of the chassis. One of the main components which control the gesture of the car, the line following sensors was mounted into the bottom layer of the chassis in order to sense the white line. The breadboard which holds the DC to DC converter, motor connections (H-Bridge) was placed on the top layer of the chassis just beside the vertically mounted microcontroller. As for the WIFI chip, it was mounted on top of the vertical mounted MSP432 breadboard.

2.2 MSP432P401R Microcontroller

The microcontroller chosen to govern the operation of the final prototype is the MSP-EXP432P401R. This is the first 32-bit ARM Cortex-M4R Launchpad that originated from the well-known MSP430 family. It allows developers and engineers to develop complex performance applications that benefit from low power consumption. The MSP432 comes with a rich set of configuration of analogue, timing and communication peripherals as shown in Figure 4 providing a wide range of opportunities of applications. [3] [4]

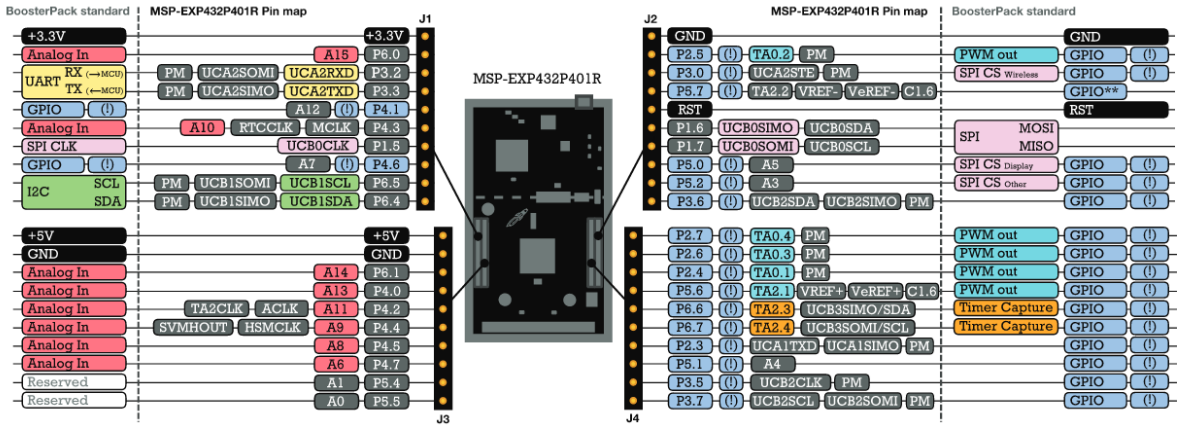


Figure 4 - MSP432P401R pin out configuration

There are precisely 40 pins within the MSP432 providing analogue, digital or PWM functions based on their specific applications. These three types of pins shall be used in the next sections. For instance, the analogue pins can be used to read the output voltage of the sensors. Whereas the PWM pins can be configured to fine control the output voltage of the MSP432 with the aid of the Pulse Width Modulation. For example, PWM pins have an important practical usefulness of controlling the speed of the DC motors driving the wheels. [3] [4]

The MSP432 microcontroller is very similar to the Arduino platform which makes it easier for Arduino developers. In fact, those microcontrollers share some of the libraries and can both compile some of the written sketches. The MSP432 comes with a user interaction feature that is based on two buttons with two LEDs. This feature can be used initially for testing purposes by compiling a *'Blink'* sketch into the microcontroller. It also consists of a UART through USB which holds the functionality of data transfer.

An extraordinary feature that the MSP432 holds is the ability to support *'multitasking'* by means of compiling more than one sketch to the board at a time which will execute consecutively in parallel without any delays affecting other tabs in the program. This feature was very useful when the car was trying to follow a white line while connected to the server to get the required path as well as playing a melody in the background.

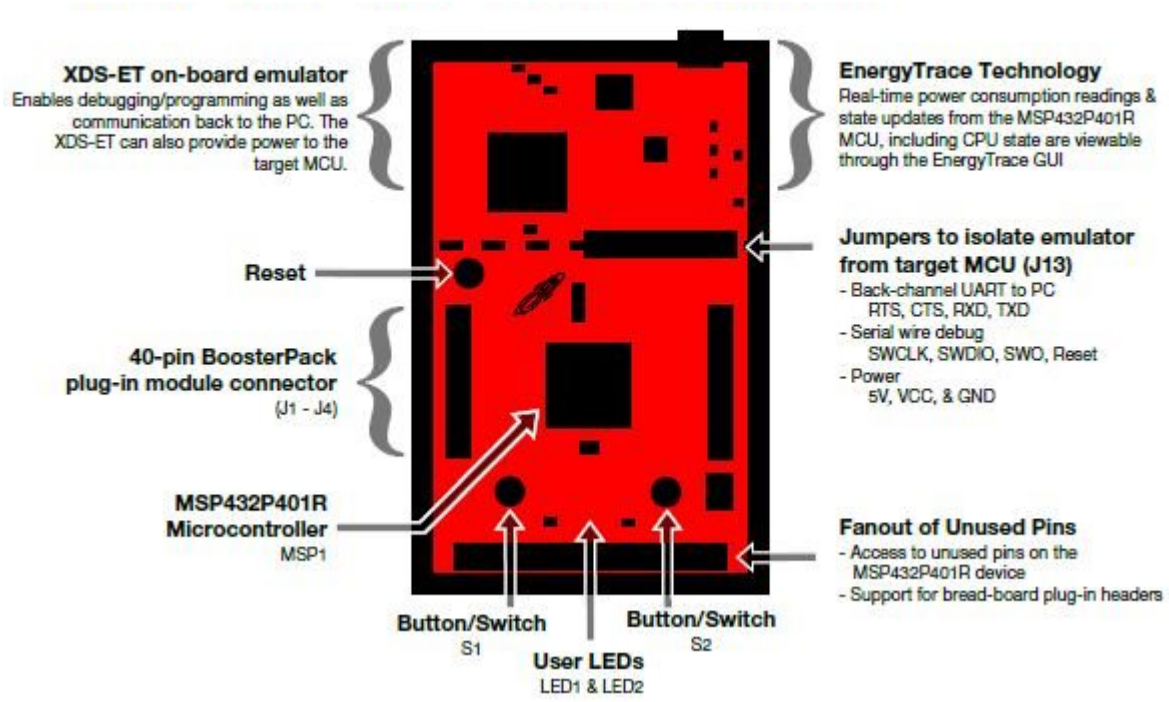


Figure 5 [5] - MSP432P401R Overview

Figure 5 shows some of the components within the MSP432. Table 2 shows some of the features of the MSP432 obtained from the datasheet. Those types of information must be considered during the development of the prototype. For instance, if a voltage greater than $4.17\ V$ was applied to any pin; this will result in the damaging of the MSP432. In order to avoid such a consequence, a resistor must be connected to any input pin to MSP432.

Table 2 [5] - Limitations of MSP432P401R

Features	***Value***	
	Min	***Max***
Supply Voltage	$1.62\ V$	$3.7\ V$
Temperature Range	$-40°C$	$85°C$
Voltage applied to any pin	$-0.3\ V$	$V_{cc} + 0.3\ V$
Absolute Max voltage to any pin	$-0.3\ V$	$4.17\ V$

2.3 DC to DC Converter

The DC-DC converter used in this project is the *'OKI-78SR Series'*. The main function of this electronic component is to convert a source of direct current (DC) from one voltage level to another [6]. Hence it acts as an electric power converter for the purpose of this project. In this project, it is used to reduce the battery voltage source from $9V$ to approximately $5V$ as it was mentioned previously that the MSP432's ports are sensitive to high levels of voltage. In addition, the DC motors are operated about $5V$. The *'OKI-78SR Series'* comes with multiple features that enables it to suit a wide range of applications. For instance, it can accept a wide input range of $7V$ to $36V$ DC. It also incorporates a protection features which includes short circuit current limit protection. The acronym of the DC-DC converter (*OKI-78SR Series*) can be further illustrated in Figure 6. [7]

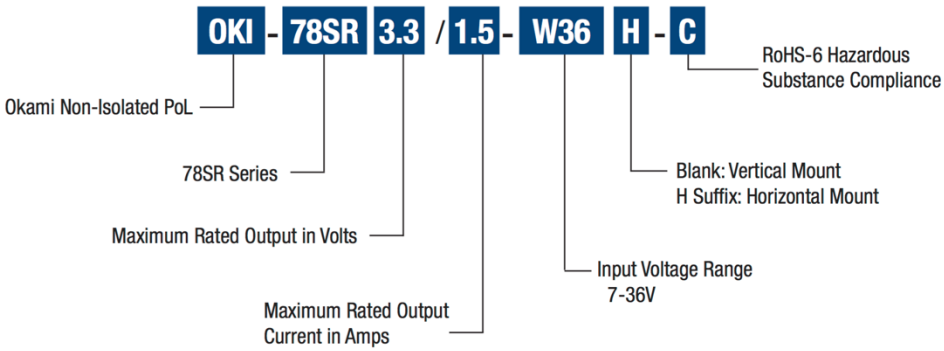


Figure 6 [7] - OKI-78SR acronym description

The *'OKI-78SR Series'* offer two nominal voltages of either 3.3 or 5 V DC with each choice providing different characteristics. The different characteristics and limitations can be obtained from the datasheet.

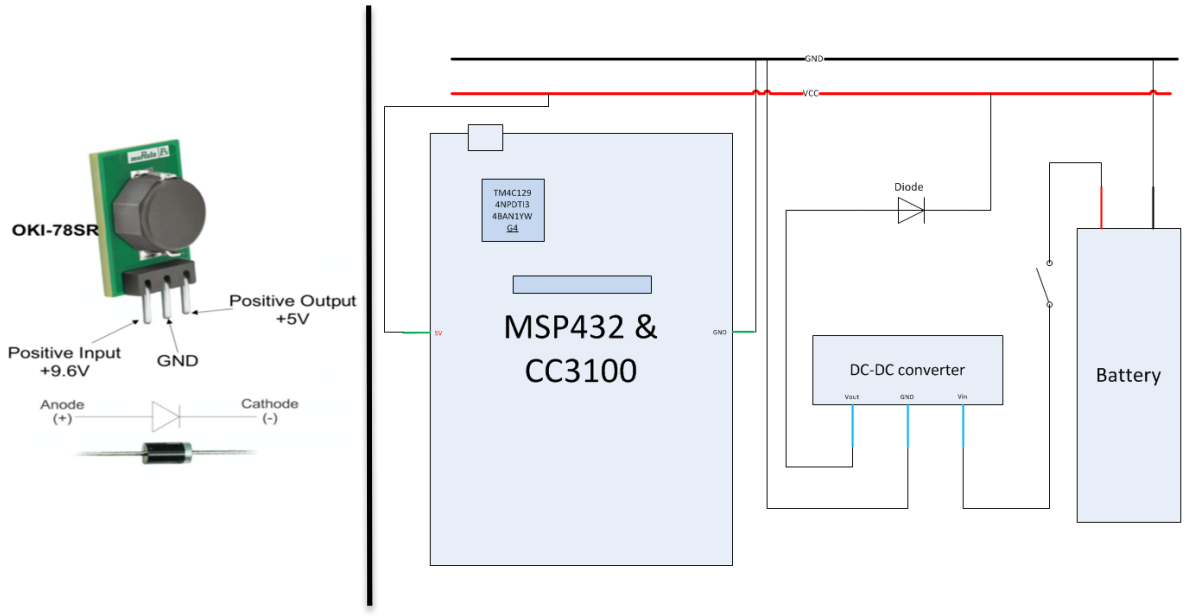


Figure 7 - DC-DC Converter Circuit

Figure 7 shows how the regulator's circuit is connected. Initially pin 1 is connected to the input voltage that is required to be reduced while pin 2 is the common (ground) pin and pin 3 is the positive output or more alternatively the output voltage. According to Figure 7, firstly the positive node of the battery is connected to a switch then into pin 1. Pin 3 is then connected to a diode. The main usage of the diode is to reduce the voltage by a desirable miniature amount which is fully suitable for the MSP432. The diode also has another feature which is protecting the microcontroller when the power is highly reduced. The DC-DC converter circuit can tested with the aid of a voltmeter by ensuring that the output voltage obtained from pin 3 of the DC-DC converter is approximately $5V$.

2.4 Motors Connections [8] [9]

To achieve the forward/backward functionality, both DC motors of the 2WD mobile platform are interfaced through the MSP432. Unfortunately, the MSP432 provides a low current control signal that cannot drive both DC motors. This was resolved with the aid of the DRV8835 dual motor driver carrier manufactured by Texas Instrument. Those carriers enable voltage to be applied across a load in either direction. Hence the wheels can be driven in both directions resulting in an efficient steering mechanism. In addition, the speed can be varied easily with the aid of such a component. Figure 8 shows a simple circuit diagram of a motor driver that is used to describe how the motor driver can be utilised to drive the motor forward or backward.

According to Figure 8, there are 4 fundamental connections on the motor driver and by varying which connections are activated, it is possible to control the flow of current which will then decide the direction of the motor rotation. Extreme care must be taken as a short circuit can be created which will result in the malfunction of the motor driver.

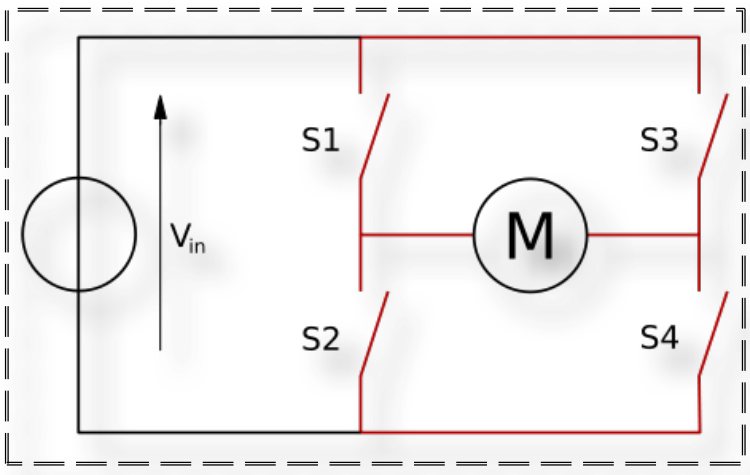


Figure 8 - Motor Driver Circuit

There are two cases that are very useful when operating the motor driver. The first one is when S_1 and S_4 are activated. This will result in the motor moving in reverse. As the current will follow the path flowing from the connection S_4 to S_1 which cause the motor to flow in reverse. The second condition is when S_2 and S_3 are activated this will cause the current to flow from connection S_2 to S_3 which will cause the motor to move forward. This is illustrated in Figure 9.

As mentioned earlier care must be taken because when changing from forward to reverse it can cause a short circuit to occur when moving from forward to reverse therefore a delay is used to give enough time for the connections to be deactivated before changing to another direction.

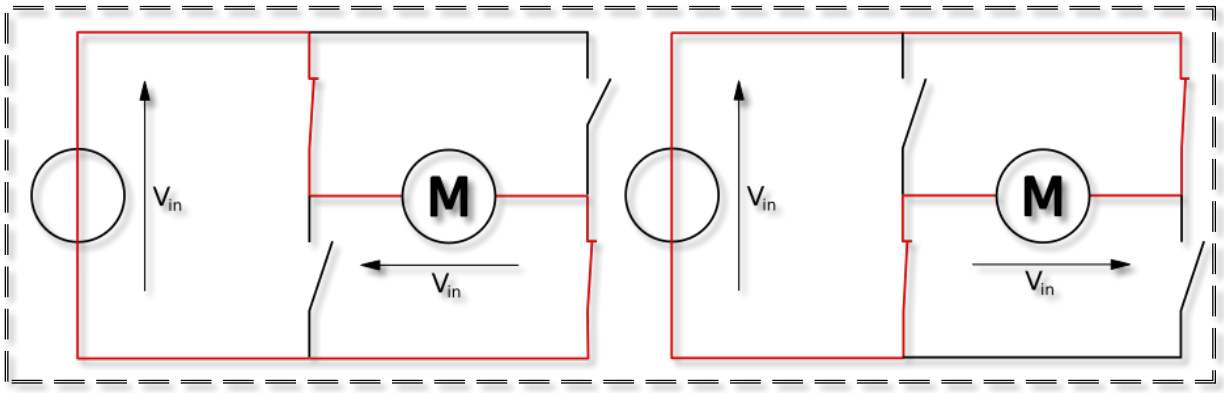


Figure 9 - Forward and Backward functionality of motor driver

Table 3 shows some of the necessary information required in the design of Mr. Robot. That information is based on what voltage can be applied into the wheels through the motor driver and output current obtained etc. Generally, this carrier is efficiently useful for the both DC motors in the case of this project as it has an operating voltage range from 0 V to 11 V and built-in protection against reverse voltage, under-voltage, over-current and over-temperature

Table 3 - Features of Motor Driver

Features	***Value***	
	Min	***Max***
Motor Supply Voltage	$0\ V$	$11\ V$
Logic Supply Voltage	$2\ V$	$7\ V$
Output Current	$1.2\ A$	$1.5\ A$
Input Voltage	$3\ V$	$3\ V$

This Texas Instrument is designed efficiently such that the motor and motor power connections are made on one side of the board and logic power and control connections are made on the other as shown in Figure 10.

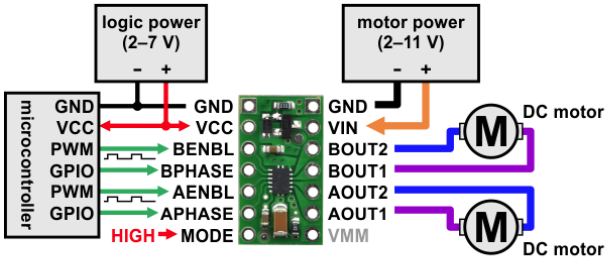


Figure 10 - Pin out Configuration of Motor Driver

The DRV8835 consists of two possible control modes which have the ability of determining the control interface. Those two possible control modes are defined as IN/IN and PHASE/ENABLE, which can be mainly either activated through the MODE pin. Setting the MODE pin high, results in setting the driver to PHASE/ENABLE mode, while the PHASE pin determines the required direction of the motor and the ENABLE pin is supplied with a PWM signal to control the motor speed.

The MSP432 sends the information in which controls the connections of the motor driver, these connections are important as they dictate the direction in which the motors will rotate. The motor driver then sends the signal to the two motors which will drive the motors in the orientation prescribed by the MSP432. It is also important to note that the MSP432 also controls the speed of rotation. It does this using a special pin called PWM (pulse with modulation pin). The digital pins in the MSP432 will have an output voltage of 0V or 5V. The PWM pins in the MSP432 make it possible to vary the output between 0V and 5V.

The PWM pins work on the basis of getting a signal and turning it on and off very quickly in a square wave formation. The percentage of how long the total signal is ON is called the duty cycle. This number is then multiplied by the maximum voltage which is 5V which will result in the average voltage level.

Figure 11 illustrates three separate waves with different duty cycles. Those duty cycles corresponds to different voltage levels as shown in Table 4.

Table 4 – Duty Cycle with corresponding Voltage Level

Duty Cycle Percentage	***Voltage Level***
$50\%\ Duty\ Cycle$	$2.5\ V$
$75\%\ Duty\ Cycle$	$3.75\ V$
$25\%\ Duty\ Cycle$	$1.25\ V$

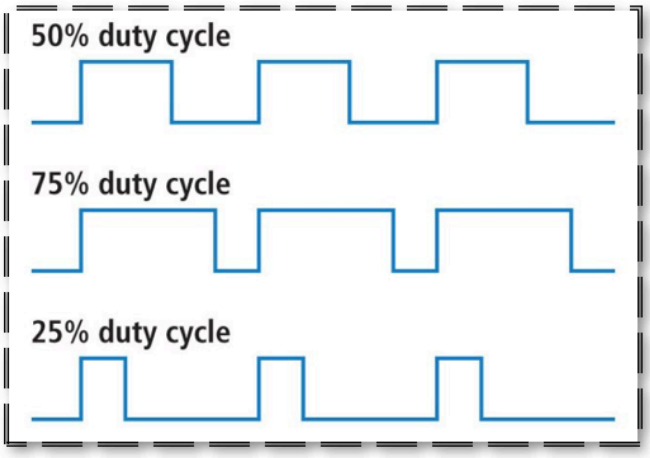


Figure 11 – Duty Cycle variations based on PWM

Hence the output voltage can be varied with the aid of PWM pins. This variation in output is important as it will influence the speed in which the motors rotate.

Figure 12 shows how the DRV8835 dual motor driver carrier is interfaced with the MPS432 and other consecutive components consumed at this stage of the project.

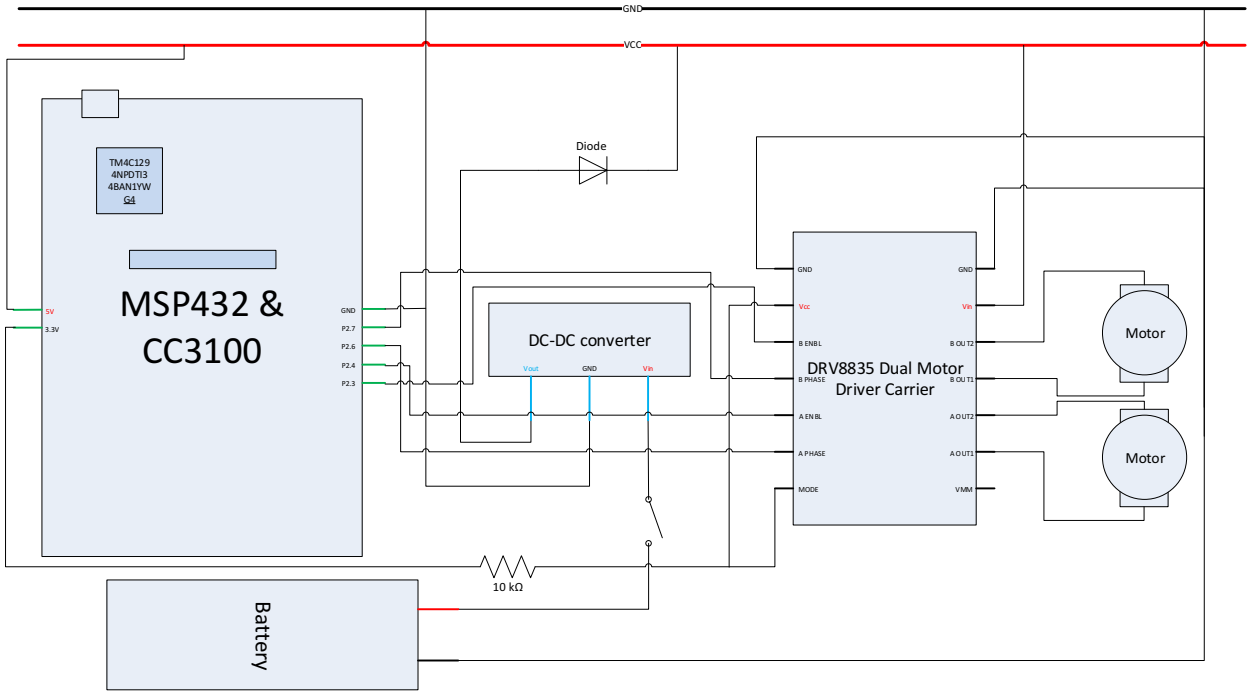


Figure 12 - DRV8835 dual motor driver carrier Circuit

2.5 Line Following Sensors [10] [11] [12]

One of the main tasks involved in the final prototype is to let the robot follow a white line with the aid of a five reflective optical sensor array. The most exceptional goal of this task is to let the robot follow the white line as the line changes curvature while letting the robot executing such a curvature. Those types of sensors are highly advanced with relation to what they can do by distinguishing between a white and black background. For example, at a white background, the sensors should give a value of 50 to 60 while in a black background; they should give output readings of 700-800.

The sensor array used are of the model 'TCRT5000L' manufactured by Vishay Semiconductors. The testing circuit of such a complex sensor can be shown in Figure 13. The TCRT5000l is an infrared (IR) reflective colour sensor that contains an IR LED and phototransistor in a single package. Those types of sensors have a set of complex applications which includes position sensor for shaft encoder, detection of reflective material, limit switch for mechanical motions in VCR and many others.

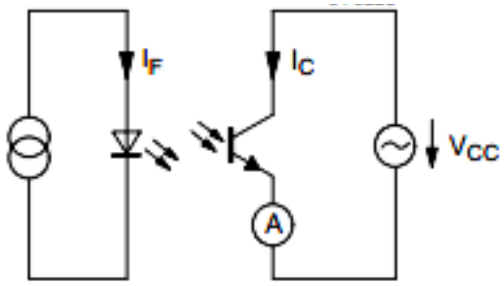


Figure 13 – Circuit of Line Follower Sensor

Table 5 holds some of the important features that TCRT5000L holds. Those features can be of enormous importance and must be considered based on the application they are used in.

Table 5 – Features of TCRT5000L sensor

Parameter	***Value***
$Total\ Power\ Dissipation$	$200\ mW$
$Operation\ Temperature\ Range$	$-25\ to\ +85\ ℃$
$Storage\ Temperature\ Range$	$-25\ to\ +100\ ℃$

Figure 14 to Figure 17 illustrates how the sensors array was utilised to follow the white line under different conditions that were met during the travelling journey of Mr. Robot.

The mobot was programmed on the basis that once the left sensors notice a black background, the car turns right so that the middle sensor is at the white line as shown below in the next set of figures. This is also the same for the case where the right sensors sense black background causing the mobot to turn left slightly.

NB: The black line is supposed to represent the white surface while the white surface is supposed to represent the black surface for Figure 14 to Figure 17.

The first case shown in Figure 14 is where the middle sensor is sensing the white background. In this situation, Mr. Robot can proceed. Note that all of the 5 sensors were used in order to achieve a high level of efficiency. The *'Left Sensor'* and *'Right Sensor'* are the most outer sensors to the left and right.

The second situation involves Mr. Robot approaching and stopping at a checkpoint as shown in Figure 15. When the checkpoint is sent from the server, Mr. Robot calculates its shortest path to the destination and stops for a certain amount of time at the checkpoint. This is sensed by all of the five sensors.

The third and last scenario involves the middle sensor sensing a black background while the outer left and right sensors are sensing a white background. In this case, the car makes a slow turn either to the left or right based on the situation shown in Figure 16 and Figure 17.

Figure 18 shows the final circuit for the line follower which is interfaced with the MSP432.

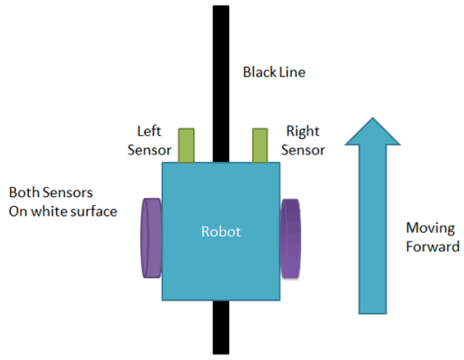


Figure 14 - Case 1

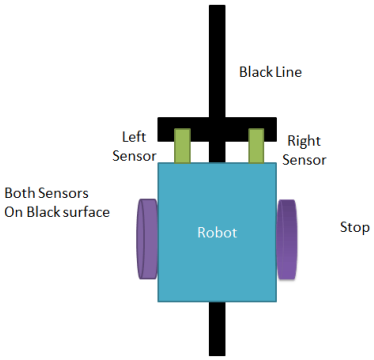


Figure 15 - Case 2

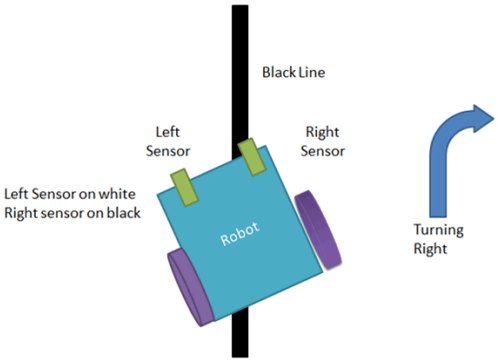


Figure 16 - Case 3

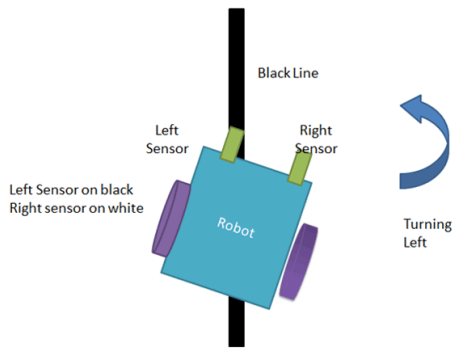


Figure 17 - Case 4

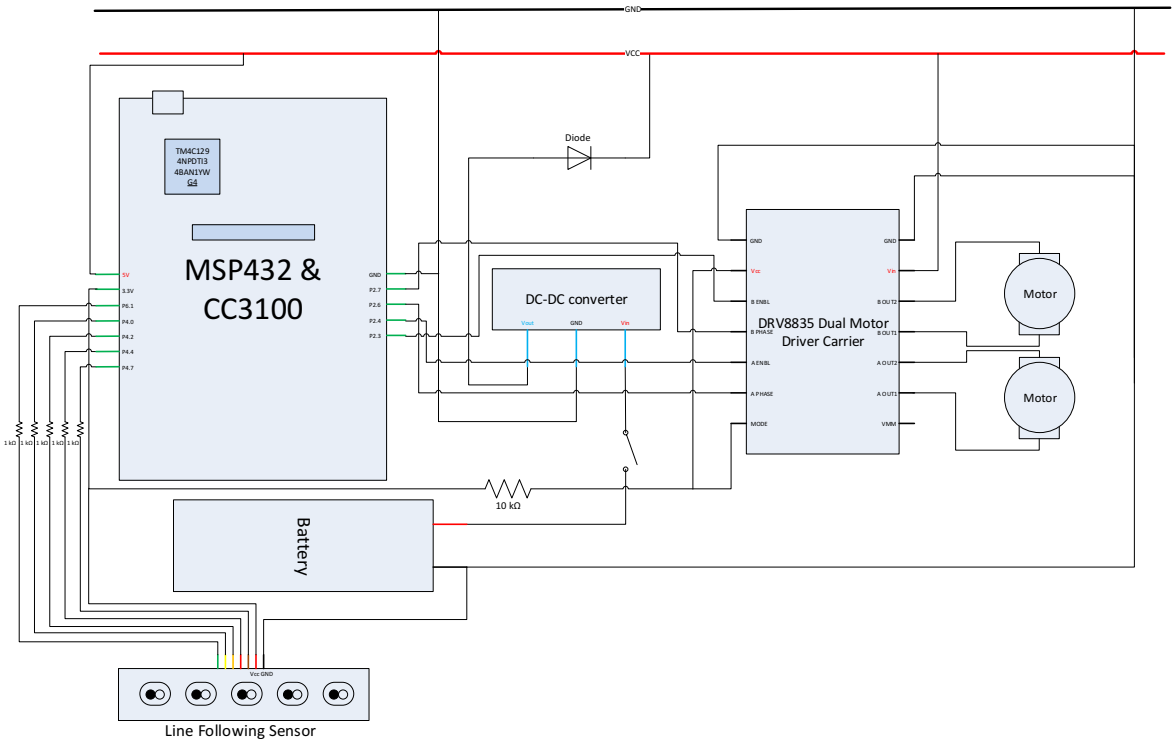


Figure 18 – Line Follower Circuit

PID (Proportional-Integral-Derivative) [10] [11] [12]

The term PID stands for Proportional-Integral-Derivative and it is a closed loop feedback system used in process control applications in various engineering divisions, i.e. machine control in mechanical and mechatronic systems, due to its high flexibility and reliability. PID aims to reduce the error between the set point (stable position) and the process variable (actual position) to zero so that the desired output can be achieved. The PID algorithm uses three constants, K_p, K_i and K_d. These three constants can be combined in such a way to obtain the desired output. Figure 19 illustrates the process of the PID controller.

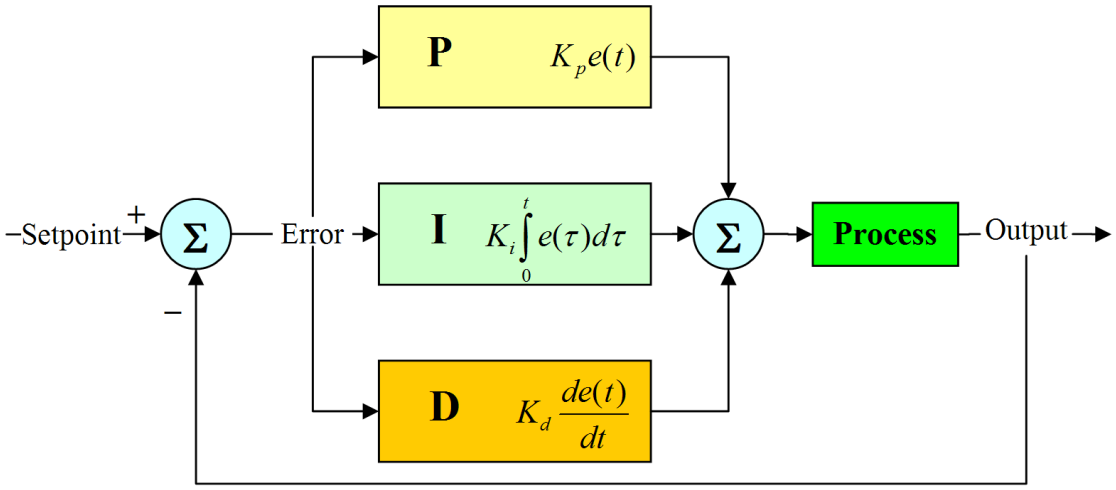


Figure 19 – PID Model

The PID controller uses three main control behaviours as the name suggests:

Proportional

Proportional or P-controller is the difference between the desired or target value and the actual measured value multiplied by a pre-determined proportional constant to get the output. The speed of the response can be adjusted by varying the proportional constant. However, as the proportional constant increases the system will try to reach the set point faster and this could cause instability in the system, which in this case turns out to be jitter along the track while following the line autonomously. The response of the system for different ranges of K_p is illustrated in Figure 20.

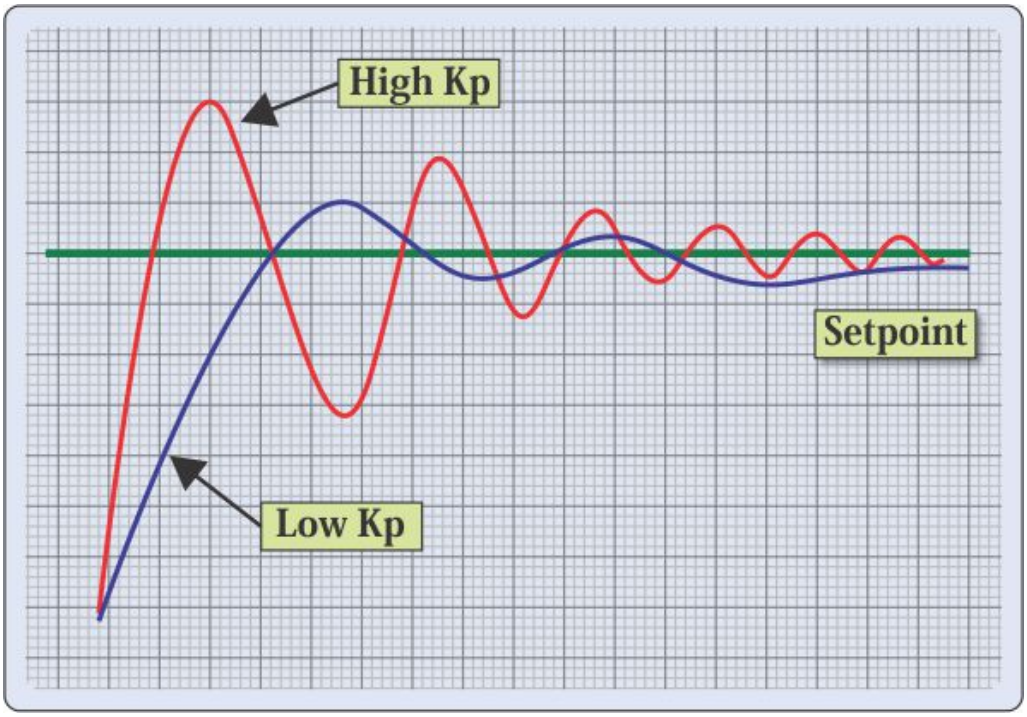


Figure 20 – Effect of Kp

The red line in the graph shows the response of the system for a high-valued K_p. As can be seen in the graph, the response is fast but the overshoot is also high resulting in instability of the system. The blue line shows the response of the system for a low-values K_p which produces a much smaller overshoot and error between the set point and the actual value resulting in a smoother and stable operation of the system. Proportional provides stable operation but always maintains the steady state error. The equation for proportional is:

$$P = K_p * error$$

and the error was calculated using the following formula:

$$error = setpoint\ value - measured\ value$$

Integral

Integral exists to eliminate the steady state error caused by the proportional (P). Integral measures the accumulated error over a period of time until the error gets to zero. The proportional (P) only creates a response for the current error, however this is not enough to dispose the error from the system completely. The system must also be able to watch the past errors and change the system output accordingly and this is where the integral comes into play. When the car starts to move, the proportional (P) of the system causes some overshoot resulting in the integral to increase to a big-valued number over time. As the integral increases the system reacts rapidly to fix the system and bring the system integral back down to zero. When the system stabilises to the point where the error is steady, the integral takes over and changes the output of the system as the integral value increases

over time depending on the error. To sum up, the longer the robot is not centred over the line, the higher the integral value becomes due to the error calculated by the microprocessor. Similar to the calculation of the proportional (P), the integral is obtained by multiplying the sum of the previous errors by a pre-determined or tested value, K_i. K_i is a constant value used to increase or decrease the impact of integral.

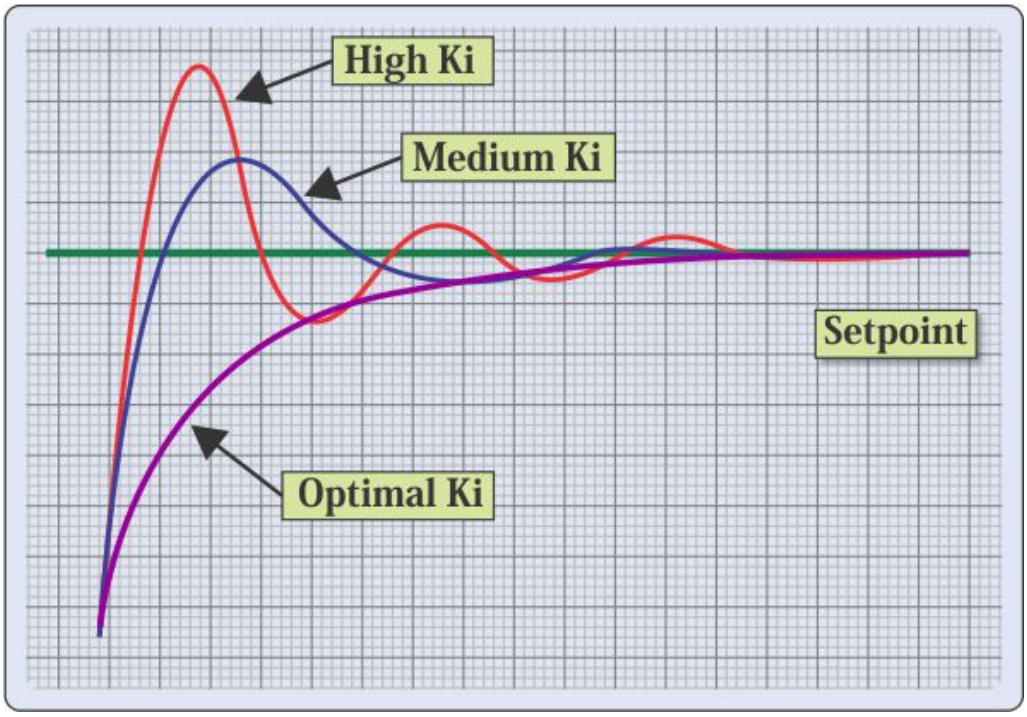


Figure 21 – Effect of Ki

As can be seen in Figure 21, a high K_i value (red line) causes a large overshoot but unlike the proportional response the integral response stabilizes to a steady state after a few oscillations due to the error reducing down. As the value of K_i decreases, the response time of the system is slow but much more stable. The equation used to obtain the integral is as follows:

$$I = K_i * \int error$$

the error was calculated the same way it was calculated for the proportional.

Derivative

The derivative measures the rate at which the robot is moving side-to-side. The faster the robot jitters, the higher the derivative value is. Basically the derivative value depends on the rate of change of error with respect to time, the derivative also aids in forecasting the future behaviour of the error so that the system response can act accordingly. The derivative raises the stability of the system by compensating for the phase lag caused by the integral. If the

system is slow, the derivative is increased to make the PID system respond faster and vice versa. the derivative can be mathematically represented as follows:

$$D = K_d * (error_n - error_{n-1})$$

The resultant formula used for the PID control system is as follows:

$$\boldsymbol{Output = P + I + D} = K_p * error + K_i * \int error + K_d * (error_n - error_{n-1})$$

The final result for a PID controller that was utilised for this project was the combination of all of the above, Proportional, Integral and Derivative and a typical response of a system is for the specified values of the three constants is illustrated in Figure 22.

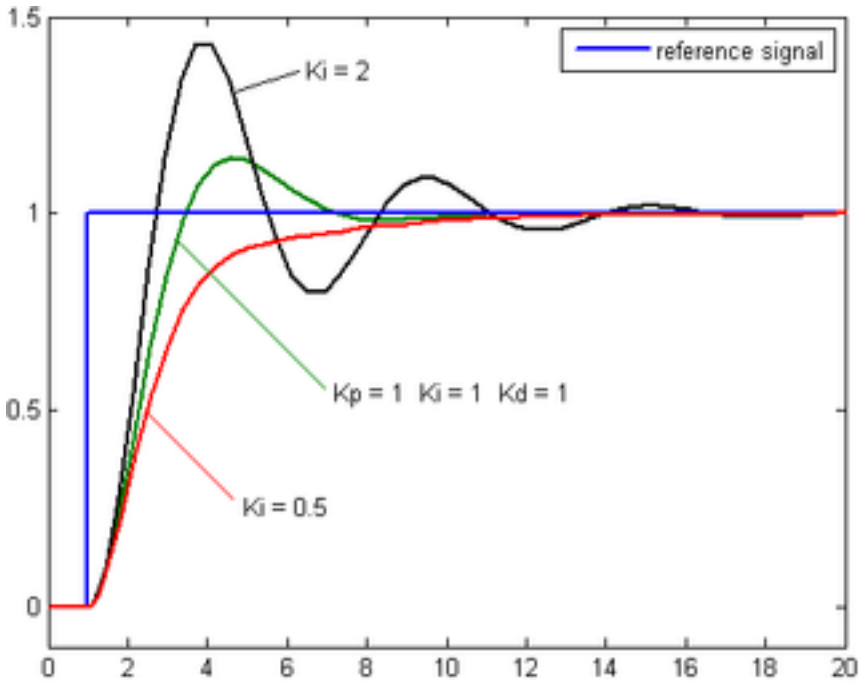


Figure 22 – Typical PID Output

In conclusion, the PID controller was used to smoothen out the turns, reduce the jitter and optimise the efficiency of the processing power that was provided and made available for this project, Texas Instruments MSP432P401R Launchpad.

The PID controller in this case was tuned for the track using the trial and error method. First of all, the K_p value was determined while maintaining the other two constants to zero so that the desired oscillation was attained for the car, then the value of K_i was adjusted to compensate for the steady state error and finally the value of K_d to smoothen everything out and to get a faster, smoother and a reliable response for the system.

2.6 WIFI CC3100 BoosterPack [13] [14]

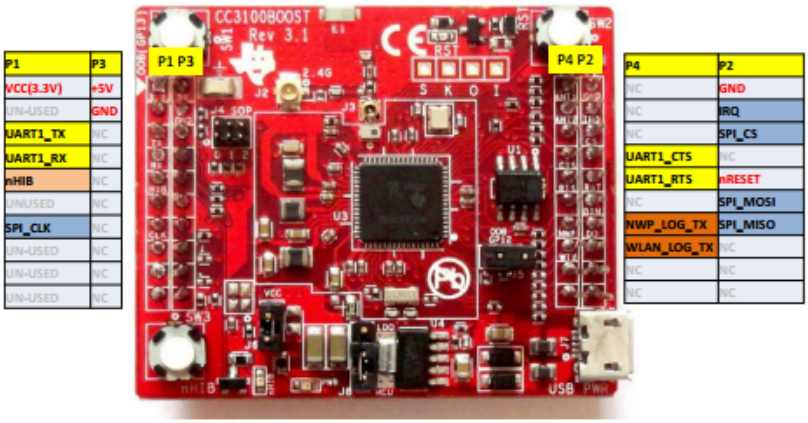


Figure 23 – Pin Out of WIFI CC3100 BoosterPack

The WIFI CC3100 Booster Pack shown in Figure 23 is a Wireless Network Processor with on-chip Wi-Fi, internet, and robust security protocols. It can be used to connect any low-cost, low-power microcontroller (MCU) to the Internet. WIFI CC3100 Booster Pack Features:

Power from on-board LDO using USB or 3.3 V from MCU Launch Pad
8 Mbit serial flash (M25PX80 from Micron)
40 MHz crystal, 32 KHz crystal and optional 32 KHz oscillator
Designed to accept power from a connected MSP432 Launch Pad

LPDS mode: This is the lowest power mode that can be exercised while continuing to maintain the application context (by means of memory retention) and the networking context (continue to maintain the Wi-Fi connection) - with some entry-exit latency overheads. WIFI CC3100 Booster Pack limitations:

1. There must be a 100K pull-down resistor on the pin19 (JTAG_TCK) for the device to reliably enter the LPDS mode. This is not present on the boards.
2. When the CC3100 device goes into hibernate state, all the digital IOs would be floating; this includes all input and output pins. While the floating inputs on the CC3100 would not cause any leakage, the outputs need to be held at valid states so that the connected Launchpad or board does not have a glitch. For example, the UART_TX line needs to be pulled high on the board using an external pull-up (100K) so that the external MCU does not get triggered by a false start bit. Similar pulls are needed on all the output pins from the device, if these cannot be provided on the MCU.

3. The WIFI board is known to affect pins. Some of the digital out pins can be affected and go into different states then prescribed in software.
4. The board requires a lot of power and if trips up it will restart causing the previous connection to be terminated.

The Wi-Fi shield was mounted onto the MSP432 by aligning $3.3V$ pin on the MSP432 with the arrow mark on the Wi-Fi shield as illustrated in the Figure 24.

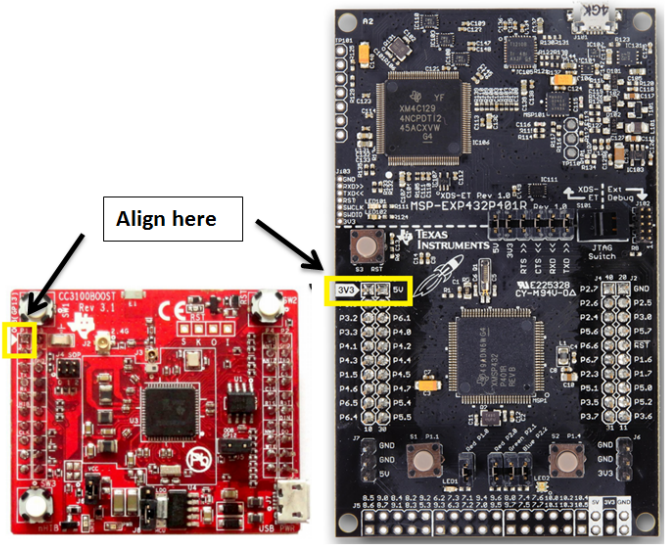


Figure 24 – Mounting of Wi-Fi to MSP432

2.7 Sharp Distance Sensor [15] [16]

A sensor is an object whose purpose is to detect events or changes in its environment and send the information to a computer or microcontroller that is embedded in the system, which then tells the actuator (output devices) to provide the corresponding output. It is also a device that converts real world data (Analogue) into digital data that a computer can understand and operate on.

A Distance sensor also referred to as a proximity sensor is a sensor able to detect the presence of nearby objects without any physical contact. A proximity sensor often emits an electromagnetic field or a beam of electromagnetic radiation and looks for changes in its field of view. The distance sensor used in this case is an IR emitting sensor which comprises an emitter and a receiver to send out a beam of IR ray and receive the reflection of the sent-out ray. The object being sensed is often referred to as the sensor's target.

Figure 25 – Distance range of sharp sensor

Table 6 – Relationship between voltage and distance

Voltage (V)	Distance (cm)
2.3	10
1.4	20
0.96	30
0.75	40
0.62	50

As shown in Table 6 the distance sensor will output different ranges of voltages depending on the proximity of the sensor to an object.

The technical information which was obtained from the datasheet is presented in Table 7. Figure 26 shows the wiring of the distance sensor into the system.

Table 7 – Features of Sharp Distance Sensor

Range	10 – 80 cms
Typical response time :	39 ms
Average Current Consumption:	30 mA
Detection Area Diameter	12 cm
Operating Supply Voltage	4.5 to 5.5

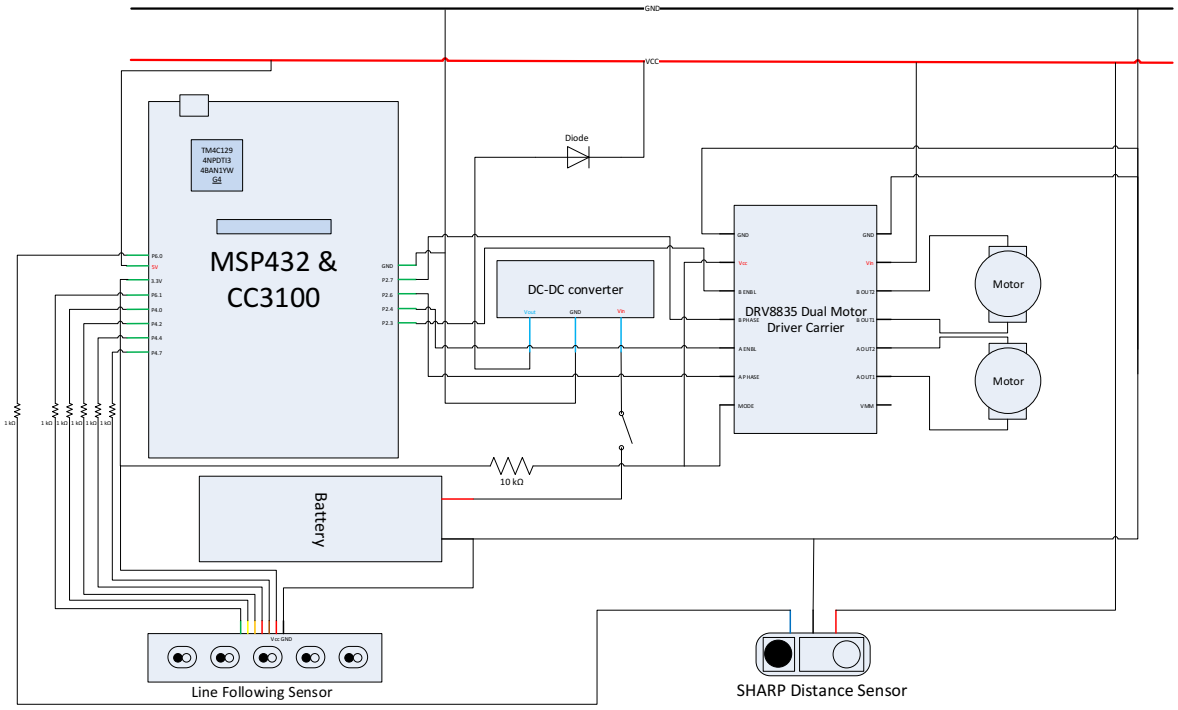


Figure 26 – Distance Sensor interface with the PS432

2.8 Additional Feature and Innovation [17] [18] [19]

Master Slave Communication

The car uses a second board with a Wi-Fi shield attached to send location data to an online dashboard and to update a twitter feed. Key metrics of the journey are sent, allowing key journey metrics to be updated and viewed in real time world-wide. The two boards are connected in such a way that multiple variable values can be sent from the master board to the slave board.

1. The master board is the board connected to and receives instruction from the competition server. It also interfaces with the various sensors and drivers to control the car.
2. The slave board is connected to the internet and receives position information from the master board and publishes it to the internet. It also interfaces with the accelerometer.

Communication between the two boards is achieved using each boards on-board Hardware UART module with the RX pin of one board connected to the TX pin of the other as can be seen in the Figure 27.

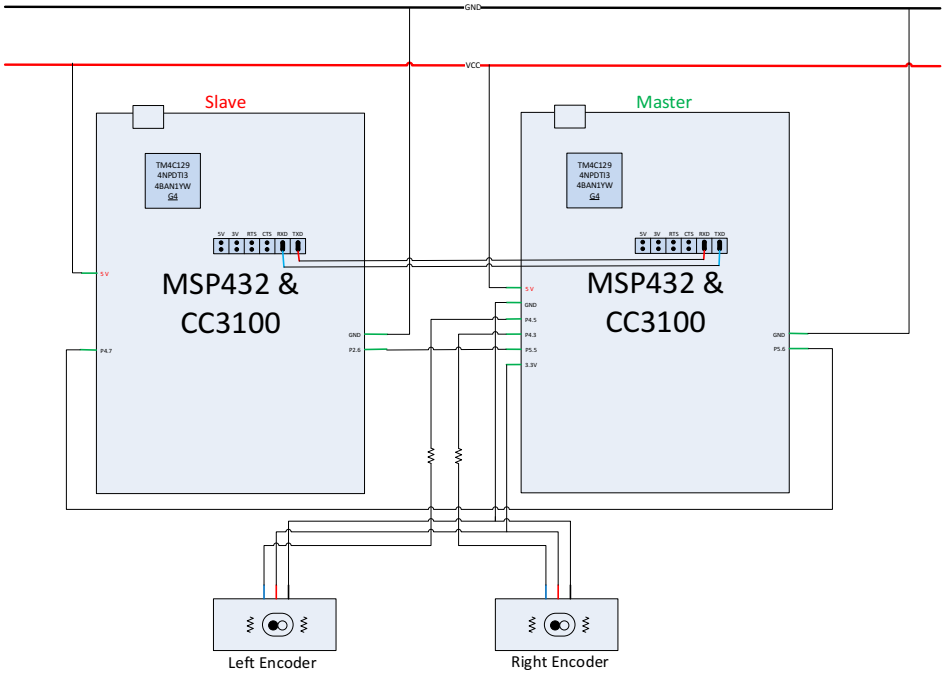


Figure 27 – Connection between Master and Slave

I/O pins on each board are then used to control and synchronise the communication of data from one board to the other on a Master-Slave fashion. The reading and writing of data as well as the initialisation of the connection between the boards is achieved using the Serial library. Using the functions in the serial library, the master and the slave can communicate with each other in the serial instead of displaying the information on the monitor for the user to view. Therefore, there was a trade-off to allow the two boards to communicate with each other. For instance, one board can write something to the serial while the other can read it from the serial and store it or use it as a flag to initiate something in the program. the communication is a two-way communication, both the master and the client can read and write to the serial. As a result of two-way communication, the serial monitor was disabled. The communication between the two boards is illustrated and discussed in detail below

Call and Response

The slave for the master to signal that data is to be sent and published to the internet. To initiate the communication the master sets its flag high and waits for the slave device to set its flag high in acknowledgement before sending the data. The complete flow control for a complete transmission cycle is shown in Figure 28.

Figure 28 – Communication between master and slave

Internet Dashboard

The internet dashboard shown in Figure 29 is built in Freeboard.io and can be logged into and viewed anywhere in the world. The dashboard has LEDs to represent each location, when the car visited a location an LED would light up on the dashboard and all the others would stay off. A graph also showed all the points visited thus far along the journey and another LED was used to show if the car had to stop at any point along its journey due to an obstruction in its path which would cause a delay in the arrival of the robot at the destination.

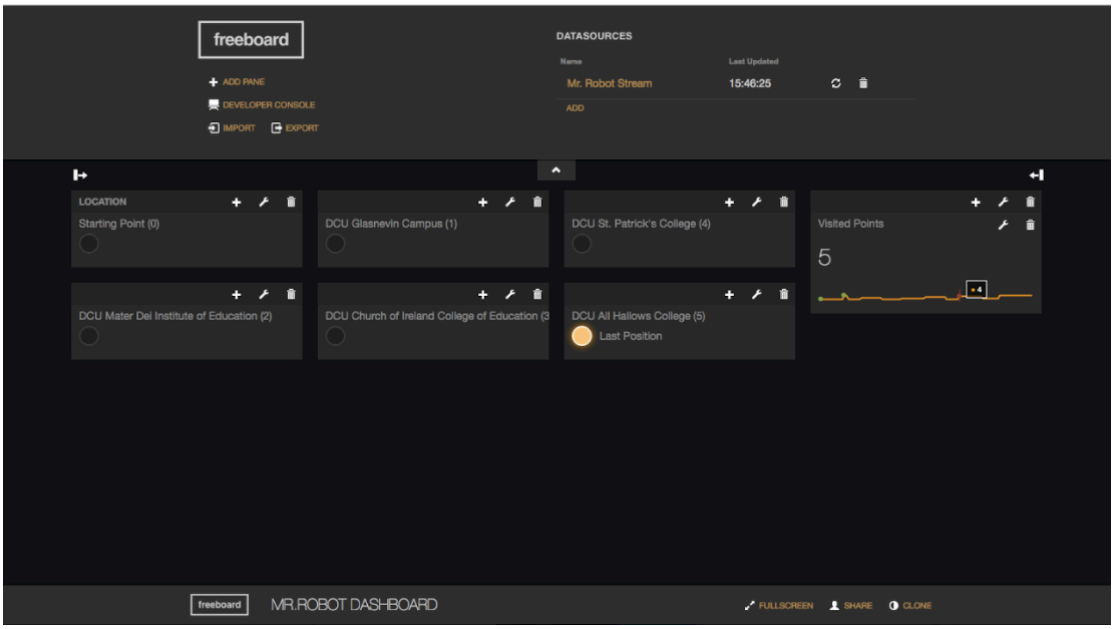


Figure 29 - Internet Dashboard

Freeboard is designed to allow fast data visualisation though dashboards which can be rapidly configured through drag and drop menus. Graphs, gauges charts, maps and user created widgets and HTML scripts can be plugged into the dashboard. Each dashboard widget can subscribe to any one of several IoT streaming APIs and for Mr. Robot's specific Freeboard application PubNub was used. For a selected dashboard indicator, a data source is chosen/subscribed to. The Freeboard dropdown menu automatically parses the PubNub data to retrieve the specific data source to be presented.

PubNub

PubNub is a data streaming network which provides an API allowing developers to build secure real-time IoT and mobile applications [1]. In PubNub there are publishers and subscribers as shown in Figure 30. In this context Mr. Robot acted as the publisher and Freeboard as the subscriber.

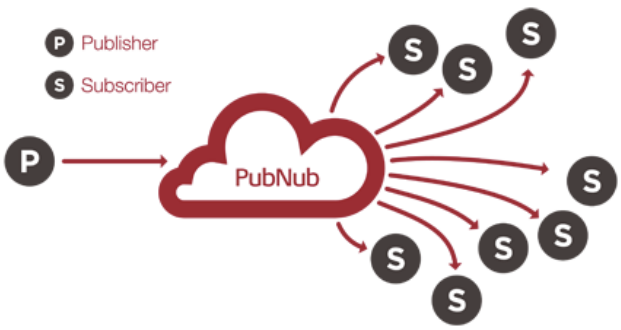


Figure 30 – PubNub

"Blocks" of code can be configured and run in PubNub cloud to set the publishing event handler type as well as operate on and process the data before processing. The setup of a subscribable channel can be performed on PubNub website after setting up an account as follows:

1. Generate a new Keyset and enable PubNub *"Blocks"* to allow operation on any data. The keyset comprises of a Publisher key, subscriber key and a secret key to enable secure transmission of the data.

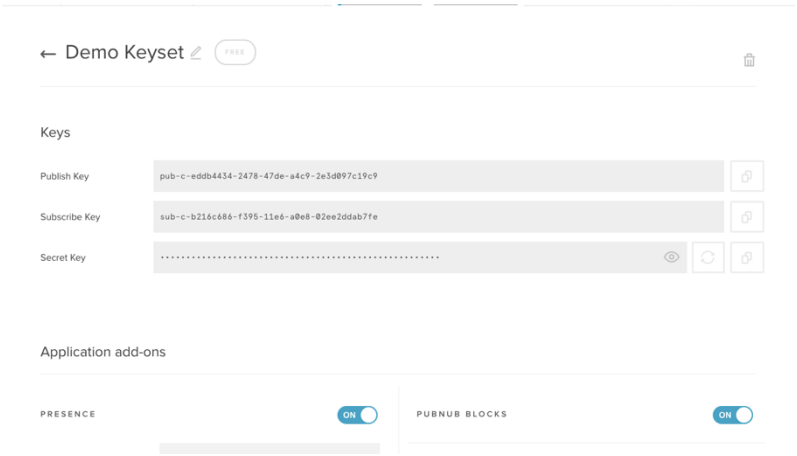


Figure 31 – Generating a keyset

2. Create a new block, configure the Block and the event handler. Then add the block to the keyset and run the code as shown in the figure below:

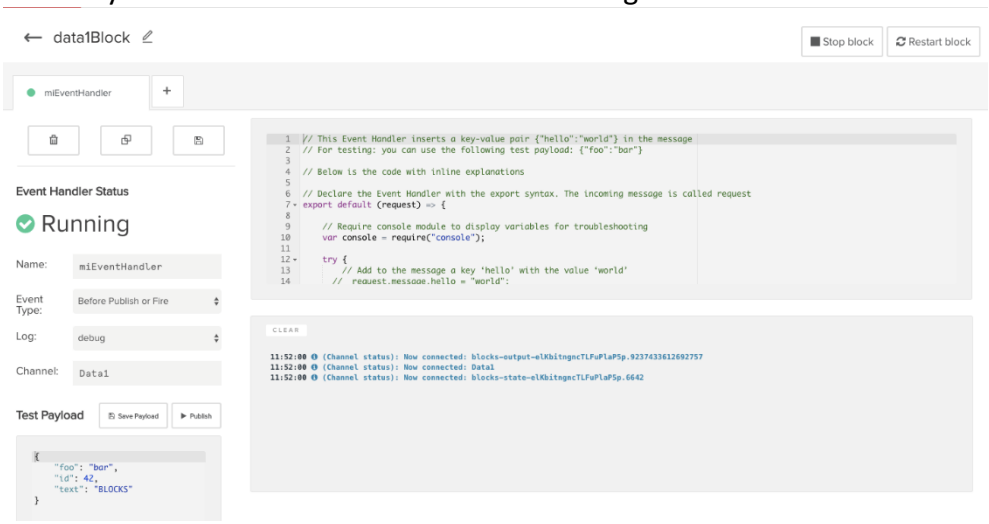


Figure 32 – Creating a new block

The data stream sent to the PubNub cloud is sent as an aJson String which is just a string formatted in a specific way. Multiple data sources can be inserted into the string. The master board uses the aJson library to setup the string for publishing and then uses the PubNub library to send it to the cloud over Wi-Fi.

Twitter

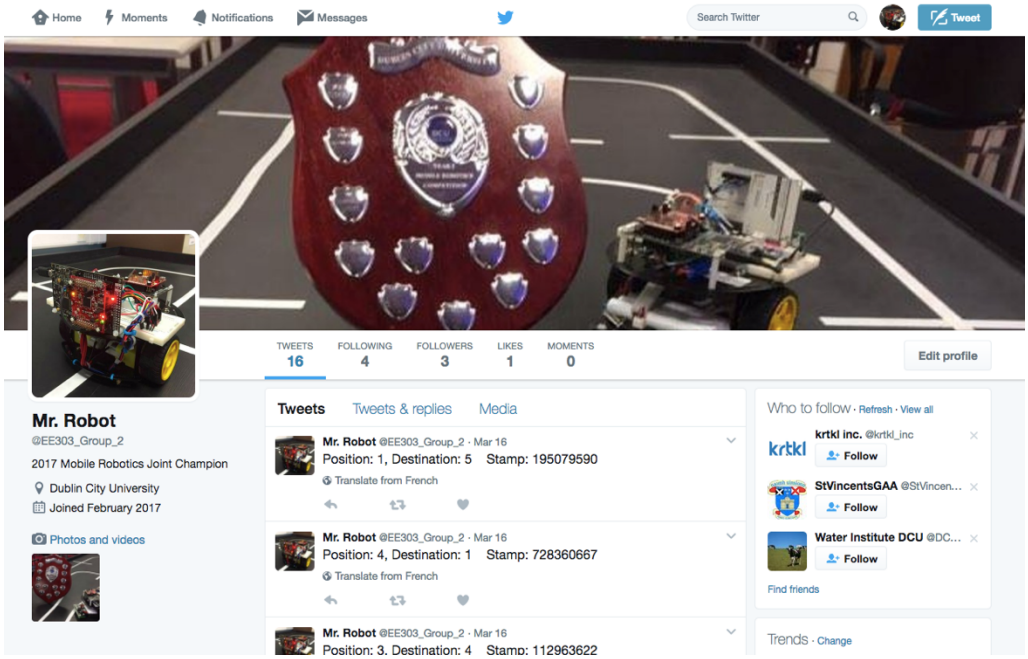


Figure 33 – Twitter Page

Figure 33 shows the Mr. Robot Twitter page. Twitter allows for automated tweets to be enabled through the Twitter.dev website. Duplicate tweets within a specific period of time are blocked as it is understood as rapid tweeting. Publishing of tweets is also quite slow in comparison to the speed achieved in visualising the data in Freeboard. The exploration of the specific reason for this was not possible due to time constrainst.

Initially upon arrival at position 0 and having received its next destination Mr. Robot tweeted that it was at the Taxi Stand, referring to the starting point. At each of the next destinations, the Mr. Robot would tweet the current position and the next position (destination). Figure 34 shows the tweets that were sent on the last run (Round 3) during the competition.

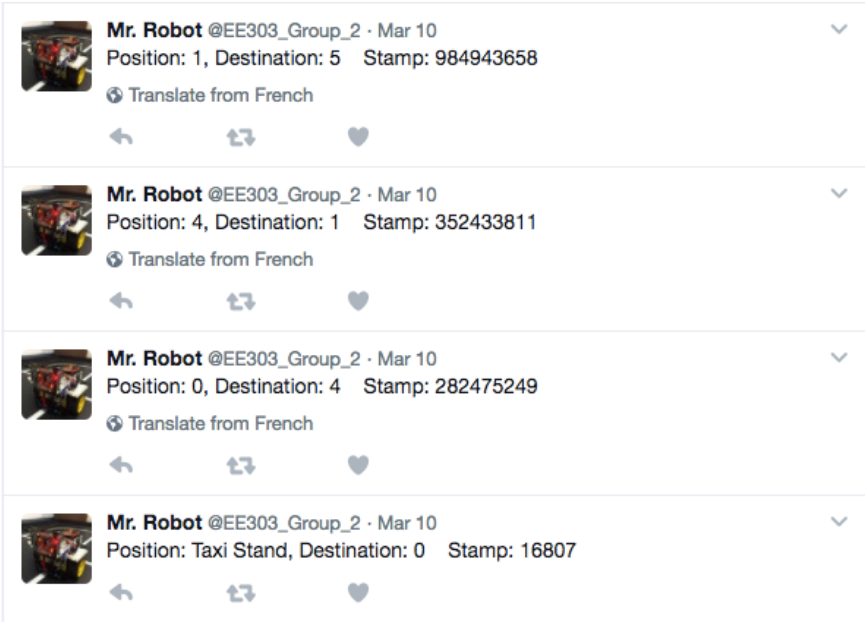


Figure 34 – Mr. Robot Tweets Current position and next destination

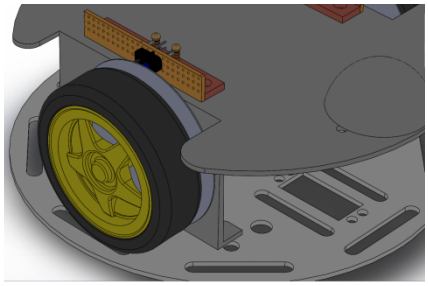
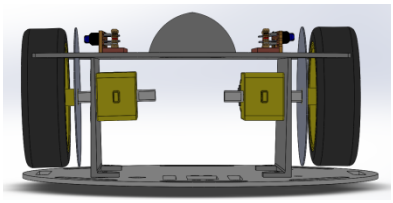
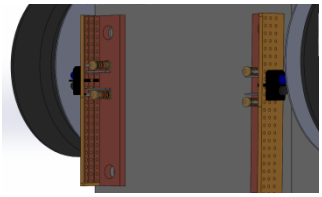
The stamp and number at the end of each tweet was used to try and stop twitter from blocking duplicate tweets within a specified period of time. The stamp number was simply a randomly generated number.

Temboo

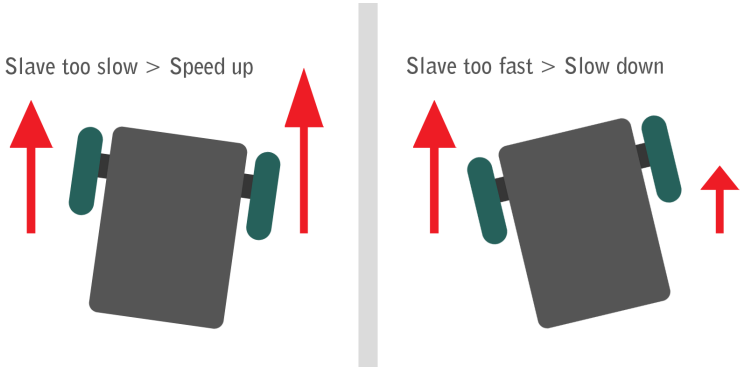
Temboo was the streaming network used to send the tweets. Temboo is similar to PubNub with regard to how it was used for this project but for more scaled and complicated applications it is much more basic and restrictive than PubNub. Temboo is also much easier to use, providing out of the box functionality and did not require configuring of the cloud network as PubNub did

Encoders

Encoders are electromechanical devices that are used to control the motors of a vehicle, in this the mobile robot, and make the wheels turn at the same rate. This was achieved by simply mounting a self-made (soldered with the right resistors in place) line following sensor at the bottom of the chassis as shown in the Table below.

Isometric View	Front View	Bottom View
		

A circular disc was attached to the wheel facing the inside of the robot and the sensor so that it can be programmed to count the number of encoder ticks within a specific timeframe and compare them with the other wheel. If the number of ticks on both wheels are the same, then that means the robot is going in a straight line; if not, some modification has to be done to the speed of one wheel to compensate the overshoot of the other. The speed of the wheel that needs to be modified is determined by the code depending on the algorithm. For instance, if the slave motor was slowing down when compared with the master motor, the slave motor will have to be sped up to match the speed of the master motor and if the slave motor was speeding more than the master motor, then the slave motor will have to slow down and wait for the master motor to catch up. Matching the speeds of the two motors was one of the major difficulties encountered during this project and hence the self-made encoders were implemented. The functionality of self-made rotary encoders is graphically illustrated in the figure below:



3. Team Dynamics and Individual Contributions

Once, the kit was provided, the team decided to meet up and start working on the project on weekly basis. The team believed that there should be a meeting log kept and after every meeting updated it, that this would keep the team on schedule. During the group's first meeting, each person's skillset is evaluated and what is their contribution to the team. The work was then divided fairly so that each team member had about 25% of the workload. A summary of the delegation of tasks is shown in 8. The team at least had one meeting in weekly basis, preferably on Monday along with the labs on Tuesday, Wednesday, and Thursday.

A Google Drive account for the project was created online. In this account, every work that a member had done was placed there; as the team members had separate files had to be accessed by each team member to make modifications. Also, there was an online meeting log made so that if a member missed a meeting, he knew when the next meeting is taking place and what tasks had to be completed by then. At each meeting, an agenda with individual goals for the project was set out. This gave team members set deadlines that we had to complete as other tasks could not be started without them.

A Gantt chart shown in Table 9 was also created so that the team had a timeline to stick to that the team knew what and when every task should be done. With regards to the team structure, at the start of each meeting, each member gave a review on what was done and discusses issues with the rest of the team. Also, the project timeline can be shown in Figure 1.

In the case of team issues, for the duration of the semester, there was no major team issue that has arisen. The reason for such a statement is due to good planning especially the meeting log table causing every member to contribute equally. In addition, it was agreed amongst the team members that more frequent meetings would have resulted in a better organisation and control of the project.

With regards to the team structure, it is believed that team was an efficient team and the team can be seen as a jigsaw. This was agreed by all team members for two main reasons. Firstly, the team consists of members with different modules based on their course of study. Secondly, every member had their own strengths on specific engineering regions. Those two reasons caused the team to just fill like a jigsaw.

Aside, from the work load among the team, the team used a useful strategy which in turn was useful. This strategy was based on not electing a team leader for the project but allow each member of the team to lead the other members in the region that the member is most

comfortable to work in while having the role of a team leader such that every member within the team got the chance to lead the team in a way or another.

3.1 Work load Distribution

Table 8 shows the delegations of tasks among team members. It can be clearly shown that the project was completed across a period of 5 weeks with the given task to each member of the team on weekly basis.

Table 8 - Delegation of Tasks

Week No.	Agenda
1	Collected the kit, establish a project timeline, complete assembly of the chassis, commence hardware and software phase. Start to research for the next task. ***Tasks For First Week*** Complete the forward/backward functionality.
2	Completed White line following functionality and started to research on WIFI. ***Tasks For Second Week*** 1. Sakthignana - Research on PID method. 2. Mark - implemented PID method for white line following. 3. Radwan - Wired Line Follower Array and start basic programing. 4. Ahmed - Wired Line Follower Array and Start Basic Programing.
3	Start Mr. Robot from smartphone over Wi-Fi while playing a melody. Further optimised PID functionality. ***Tasks For Third Week*** 1. Sakthignana - Compile basic code for Wi-Fi. 2. Mark - Improve and Test PID functionality. 3. Radwan - Research on Innovative Features. 4. Ahmed – Experimentation on MPU6050 for Innovative Feature.
4	Communicate with server by transmitting group number and position to the server while obtaining the next checkpoint. Documentation of the report. ***Tasks For Fourth Week*** 1. Sakthignana - Permutation Code for possible checkpoints. 2. Mark - Testing and developing of Permutation code. 3. Radwan - Initiated documentation phase (Final Report). 4. Ahmed - Initiated documentation phase (Final Report).
5	Enhanced the functionality of Mr. Robot. Parking Mr. Robot away from the white line an near as possible without touching the obstacle (target wall). ***Tasks For Fifth Week*** 1. Sakthignana - Parking Mr. Robot away from white line without touching wall. 2. Mark - Parking Mr. Robot away from white line without touching wall. 3. Radwan – Integrating an LCD display into Mr. Robot for innovative feature. 4. Ahmed - Integrating MPU6050 into Mr. Robot for innovative feature.

6	Additional Feature and Report. ***Tasks For Sixth Week*** 1. Sakthignana – Improve full functionality of Mr. Robot and Report. 2. Mark - Improve full functionality of Mr. Robot and Report. 3. Radwan - Improve full functionality of Mr. Robot and Report. 4. Ahmed - Improve full functionality of Mr. Robot and Report.

3.2 Gantt chart and Project Timeline

Table 9 shows the Gantt chart displaying the tasks to be completed, with their initial and due date.

Table 9 – Gantt chart

TASK LIST

TASKS	START DATE	DUE DATE	% COMPLETE	DONE
Collect and Return Kit	31/01/2017	16/03/2017	**100%**	●
Assemble Robot Kit	31/01/2017	31/01/2017	**100%**	●
Forward and Backward Funcitonality	31/01/2017	01/02/2017	**100%**	●
White Line Following	01/02/2017	09/02/2017	**100%**	●
WIFI start up/communication and Piezo Buzzer	09/02/2017	23/02/2017	**100%**	●
Parking near the wall	23/02/2017	02/03/2016	**100%**	●
Additional Feature	20/02/2017	09/032017	**100%**	●
Competition	10/03/2017	10/03/2017	**100%**	●
Report	20/02/2017	26/03/2017	**100%**	●

4. Conclusions and Recommendations

Overall, the team felt that the module was well organised over the course of the semester. The lecture content was directly related to the required tasks and provided a good foundation for the group to base their ideas on. It was also found to be very interesting putting all of the theory taught from semester 1 to semester 4 into practice. It highlighted the importance of careful planning and how every fine detail used in the design stage of Mr. Robot can greatly impact the system's ability to achieve the required specifications.

A few strategies have been acquired from this project which can be of enormous importance in the future. One strategy was based on "step-by-step analysis", meaning that for every task, whether it is software or hardware related, the most reasonable and measurable approach is to do a step by step analysis of the problem given at hand. This will in general result in a better and efficient system. Another strategy which is engineering related is the testing and calibration of the components. In this mini-project, calibration was one of the most important aspects of this project, as most of the sensors were needed to be tested and calibrated to work up to the required standard. Therefore, it is important to always understand the various aspects of the hardware that is being dealt with while questioning if it is calibrated and most importantly how it interfaces with different components.

Demonstrators were always present in lab sessions which was a great help to teams having difficulties. They provided assistance to problems encountered, while leaving it up to each team to figure out a solution amongst themselves.

As a conclusion, the aims of this project have been successfully which was mainly based the design of a microcontroller-based wheeled robot system that applies the art of IOT (Internet of Technology) technology by allowing a user to start the system through a Wi-Fi enabled portable device to autonomously navigate through the track with a pre-defined destination acquired from the server. A complex, yet interesting additional feature was then integrated into the device resulting in an advanced IOT technology. Possible advice for next year's students based on the team's experience:

- Elect a team leader!
- Listen, think and evaluate as a team!
- Take advantage of each team members set of skills!
- Work coordinately.
- Communicate with each other.
- Plan things out.
- Enjoy it!

Possible recommendation includes the addition of extra demonstrators, extending the lab hours, and provide more interesting components for the innovative feature.

5. Software

The code that was used to operate the car can be seen in the code section of the report. The code was divided into three subdivisions and each subdivision was put into a tab so that the they can be used to multi-task between the two boards. The MSP432 board allows multi-tasking which is achieved by treating each tab as a thread and RTOS (Real-Time Operating System) lets each tab to be run for a certain amount of time as data is read in which in reality looks like the microprocessor is dividing the processing power of the board between all the tabs running in the program. The global variables must be declared in the first tab for the variables to be accessible by other tabs or threads in the program. The subdivisions are:

1. Drive
2. Master
3. Slave

The Drive tab and the Master tab were put into the master MSP432 board which performs the major tasks such as talking to the server, driving to the specified checkpoint and signalling the slave the current position; the Slave tab was put into the slave MSP432 board which basically waited and listened to the master MSP432 board for instructions such as the current position and when to publish to PubNub and Twitter. The two boards, the threads within each board and the functions in each thread were organised to optimise the processing power and the efficiency of the MSP432 board. The organised hierarchy is illustrated in Table 10.

Table 10 – Functions

Master MSP432		***Slave MSP432***
sync_Master	***DriveTab***	***sync_Slave***
$setup()$	$setup()$	$setup()$
$loop()$	$loop()$	$loop()$
$updateGateway()$	$PID()$	$readMaster()$
$establishContact()$	$readError()$	$publishToCloud()$
$ReadFromServer()$	$forward()$	$tweet()$
$SendToServer()$	$backward()$	
$printWifiStatus()$	$ccw()$	
$startWifi()$	$cw()$	
	$stopRobot()$	
	$nextpos()$	
	$chkRot()$	
	$miDelay()$	
	$moveToNextpoint()$	
	$moveToJunc\_CCW\_turn()$	
	$moveToJunc\_CW\_turn()$	
	$passOverLine()$	

	$zeroFunc()$	
	$oneFunc()$	
	$twoFunc()$	
	$threeFunc()$	
	$fourFunc()$	
	$Towall()$	

Drive

The main purpose of this tab is to allow the mobile robot to drive autonomously to the checkpoint specified by the server. Above all the functions, the variables used in the Drive tab were declared so they can be accessed with ease within the Drive tab. Since the Drive tab is not the first tab in the master MSP432 board, the variables declared outside of a function cannot be viewed or accessed by any other tabs in the program. The variables declared contain information such as:

1. The output and input pin numbers for the components in the circuit
2. The speed of the car
3. The initial PID constants
4. The direction the car is initially facing in

The *setup()* function of the Drive tab simply initiates the communication between the board and the computer through the serial monitor using the following command with a certain baud rate, which in this case is 115200:

```
Serial.begin(115200);
```

The *loop()* function awaits for a flag to be turned off (*false*). Once this flag is turned off, the drive is enabled to move the car from the current checkpoint to the next specified checkpoint whole also turning the flag back on so that the *loop()* will not run more than once with the same current and next checkpoint numbers.

```
if(!hold){ hold=true;  nextpos(pos, nextPos); }
```

The *PID()* function basically implements the Proportional-Integral-Derivative concept discussed earlier in the report. It was practically achieved by purely getting the error through the sensors and computing the rate of change of error and the sum of error. Consequently these values were used to get the PID value by substituting the numbers in relevant variables in the formula mentioned in the PID part in the development section of the report. After checking if the PID value computed is valid, the wheel speeds are adjusted to suit the situation, i.e. turn in the path etc. Along with controlling the motor speeds to aid

in tackling a curve in the path, the distance sensor is also working to identify any obstacle in the path, and if an obstacle was detected the mobile robot would stop and wait until the obstacle was cleared. The code for PID is as follows:

```
void PID(){

 // stop if obstacle in the way
    if(analogRead(sensorpin)>650){
          blocked=255;
          // stop
          analogWrite(motor_right_PWM,0);
          analogWrite(motor_left_PWM, 0);

          // wait for path to clear
          while(analogRead(sensorpin)>650){}
          blocked=0;
    }

// Update errors
    prevErr=err;
    err=readError();
    intErr+=err;
    diffErr=err-prevErr;

// calculate pid value
    pidVal = int(Kp*err +Kd*diffErr +Ki*intErr);

// ensure value within allowable range
    if( pidVal>  speed )  pidVal =  speed;
    if( pidVal< -speed )  pidVal = -speed;

// turn by subtracting from wheel speed
    if(pidVal>0){
      analogWrite(motor_right_PWM, speed*offsetR);
      analogWrite(motor_left_PWM, (speed - abs(pidVal))*offsetL);
    }
    else{
      analogWrite(motor_right_PWM, (speed - abs(pidVal)) *offsetR );
      analogWrite(motor_left_PWM, speed*offsetL);
    }
}
```

The error was calculated using another function called *readError()* which essentially reads in the sensor values and converts them to digital values, i.e. 1 or 0, using a threshold value. Next depending on the position of the 5 sensors on the white line, the error value is determined and returned to the function which called it.

The following few functions: *forward()*, *backward()*, *ccw()*, *cw()* and *stopRobot()* utilise the functionality of the H-bridge in setting the direction and the speed of motion of the wheels. The concept of H-bridge and how they work is discussed in detail in the development section of the report. in practise, by setting the phase pins of the H-bridge to *HIGH* the wheels can be made to rotate in counter-clockwise direction and by setting the enable pins of the H-bridge using the PWM out pins of the MSP432, the speed of the robot can be

adjusted. For instance, the following code makes the car move in a counter-clockwise direction:

```
digitalWrite(motor_left_phase, HIGH);
digitalWrite(motor_right_phase, HIGH);
```

And the code to set the speed of the car is as follows:

```
int speed=215;
analogWrite(motor_left_PWM, speed);
analogWrite(motor_right_PWM, speed);
```

The speed of the car must only be set using PWM out pins of the MSP432 board the output voltage can be varied significantly in PWM out pins when compared with digital output pins where the only two possible outputs are *HIGH* or *LOW.*

The *nextpos()* function directs the code to the right place in the program so that the robot moves and takes the expected path. It takes in two parameters, the current position and the next position. The current checkpoint number is used to determine the function to go to and the next checkpoint number is passed into that function as a parameter so that this function knows which piece of code to execute exactly to obtain the instructed and desired result. This was achieved using a consecutive *if* statements and once the function has been done executing, the program comes out of the *if* loop and stops for 500 milliseconds to update the relevant global variables so that all the tabs in the program knows that the new checkpoint has been reached.

```
void nextpos(int currentpos, int nxtpos){

  if(currentpos == 0) zeroFunc(nxtpos);

  else if(currentpos == 1){ oneFunc(nxtpos); }

  else if(currentpos == 2){ twoFunc(nxtpos);  }

  else if(currentpos == 3){ threeFunc(nxtpos);  }

  else if(currentpos == 4){ fourFunc(nxtpos); }

  stopRobot();

// updatae global variables before returning
  destination=true;
  pos = nextPos;
  miDelay(500);

}
```

The *chkRot()* function exists to ensure that the robot is looking towards the right direction before progressing to execute the PID controller and the checkpoints code. This function takes in an integer which represents the direction the robot should be pointing towards

before proceeding to move to the next checkpoint. The function only executes when the robot is not looking towards the right direction. If the robot is not pointing towards the right direction, the function tells the robot to reverse until it detects the previous checkpoint and then do an 180° turn. The turn or rotation was accomplished by using a *while* loop and checking if the middle sensor is back on the white line. Once the robot is back on the white line the robot stops for 350 milliseconds before continuing the route. The function is shown below:

```
//    rotation function
//      ahead      -> used to reverse back over line if needed. ie. will be
behind the line at the start so no reverse
//      north =  1 -> robot facing 5
//      north = -1 -> robot facing away from 5/south
void chkRot(int set){
   /*
    * turn only if facing in direction opposite
    * to that of the desired direction
   */
   if(set*north==-1){
        readError();
        backward(); while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==1 &&
a[4]==1)){ backward(); readError();}   // drive back to the line if ahead
of line
        stopRobot();
        ccw(); miDelay(350);
        while(readError() ==5 ){}
        stopRobot();
    }

}
```

One problem encountered with multitasking was using delays in the program, hardware delays couldn't be used because it stopped the microprocessor from executing any tab for the specified time. Due to this reason, a custom delay method must be implemented which implements software delay. This was done by simply using the built-in *millis()* method and a *while* loop to ensure that the time specified as a parameter has been passed. The implementation of the custom software delay method is as follows:

```
// implements delay without using hardware delay
void miDelay(unsigned long dLay){
    unsigned long currentMs = millis();
    unsigned long prevms = currentMs;
    while(currentMs < prevms + dLay) currentMs = millis();
}
```

The whole purpose of using and implementing methods is to avoid rewriting code and for this reason, the following few methods were implemented:

The *moveToNextpoint()* method was implemented so that every time the robot wants to just simply follow the line using the PID controller until the checkpoint was reached, this function could be called. These kinds of functions reduce the number of lines in the code drastically. The function is shown below:

```
// moves the car to the next checkpoint
void moveToNextpoint(){
    forward();
    readError();
    while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==1 && a[4]==1)) PID();
}
```

The *movetoJunc_CCW_turn() and the movetoJunc_CW_turn()* were designed specifically for the junction that exists on the way from 0 to 1 or 2 to 1 on the track. When the robot is instructed to go from 0 to 1 or 4 to 1, the *movetoJunc_CCW_turn()* is executed because the robot must turn counter-clockwise at the junction and if the robot is approaching the checkpoint #1 from the other side of the track, *movetoJunc_CW_turn()* is executed because of the orientation of the car at the junction. These two functions simply contain commands to call on different functions in the same thread such as PID controller with custom software delay method and a change of speeds to make the robot jitter free during the turn. The *while* loops in the function exist to check if the car is back on the track by constantly checking the middle sensor for a reading corresponding to the white line.

The functionality of the *passOverLine()* method is to update the error of the PID control system once the robot crosses a checkpoint because once the robot crosses a checkpoint the PID controller loses the error so the error has to be reinstated. The *passOverLine()* method is shown below:

```
void passOverLine(){
  miDelay(200);  readError(); // ensures line passed -> update sensor
values
}
```

The methods *zeroFunc(), oneFunc(), twoFunc(), threefunc()* and *fourFunc()* follow the same concept as *movetoJunc_CCW_turn()* and *movetoJunc_CW_turn()* since they all call upon different functions to complete the task at hand. However, the *Func* methods comprise of consecutive *if* statements that execute depending on the next checkpoint number obtained from the server. At the destination of a checkpoint, the variable *north* is updated to eother 1 (north or facing the wall) or -1 (south or facing away from the wall) depending on the direction the robot is pointing towards so that when the next destination is received from the server, the program knows which the robot is currently facing towards and which direction it should be facing before leaving the current checkpoint. Even though a range of different functions were created to aid in the motion of the car towards a checkpoint, there

are certain situations where the situation is unique and have to be treated independently as can be seen in the full code presented in this report.

At last, the function to allow the robot to leave the track and move towards the wall the *Towall()* function was created which gradually slows down as the robot gets closer to the wall. The distance between the wall and the robot is determined using the IR distance sensor mounted at the front of the car. When the distance between the robot and the wall becomes less than or equal to an inch, the robot stops and informs the server that the destination has been reached.

Master and Slave

Upon switching on the cars power supply each board enters its setup where among other things a UART connection between the boards is established. Each board initially sets up its Wi-Fi connection, starts a serial connection with a baud rate of 115200 and configures the pins used to control the UART. When a serial connection is established between the two boards using UART, the *Serial.available()* function will return a number greater than 1 on either board. To get the connection to synchronise the master repeatedly sends the *'A'* character to the serial, the piece of code that carries this functionality is shown below:

```
void establishContact() {
  while (Serial.available() <= 0) {
      Serial.println('A');
      digitalWrite(LED, HIGH);
      delay(150);
      digitalWrite(LED, LOW);
      delay(150);
  }
  Serial.flush();
}
```

Upon completing the setup as described above, the slave device enters a loop reading from the serial until an 'A' is read in the slave MSP432 board.

```
do {
    Serial.readBytes(buffer, 1);
} while (buffer[0] != 'A');
```

When the character is read serially by the slave an acknowledgement is automatically sent and the *Serial.available()* returns a number greater than 1 on the master side of the communication.

The master MSP432 implemented the following functions in order to achieve the IoT (Internet of Things) innovative functionality:

```
void updateGateway(){

      if(pos==13 || nextPos==10) return;
```

```
      // used for a loop timeout so wont hang
      unsigned long timer = millis();

      // signal to slave
      digitalWrite(flagMaster, HIGH);

      while(digitalRead(flagSlave)==LOW && millis()-
timer<6000){digitalWrite(LEDb, HIGH);delay(50);digitalWrite(LEDb,
LOW);delay(50);}

      // write data
      Serial.write(nextPos);
      Serial.write(nextPos);
      Serial.write(blocked);

      digitalWrite(flagMaster,LOW);

      // Wait forslave flag to go LOW and signal end of transmission
      while(digitalRead(flagSlave)==HIGH && millis()-
timer<6000){digitalWrite(LEDb, HIGH);delay(50);digitalWrite(LEDb,
LOW);delay(50);}
}
```

For the send of payload portion of the transmission each variable is written to the serial from the master side using a separate *Serial.write()* statement. Each call writes a byte representing the value held in each respective variable to the serial. In doing this the *Serial.readBytes(buffer, 3)* function call can be utilized to perform both the reading and parsing of the data on the slave side. The call to this function with the arguments given looks for three bytes to read from the serial and conveniently places each byte into a separate entry of the array buffer. To ensure a race condition does not occur resulting in infinite looping of either the master or slave side a timeout is used on both sides so that the total time allowed for any transmission cycle does is 6 seconds.

```
void readMaster() {

  // execute read if signaled
  if (digitalRead(flagMaster) == HIGH) {

    // used for a loop timeout so wont hang
    unsigned long timer = millis();

    digitalWrite(flagSlave, HIGH);

    while (digitalRead(flagMaster) == HIGH && millis() - timer < 6000) {
      digitalWrite(LEDb, HIGH);
      delay(50);
      digitalWrite(LEDb, LOW);
      delay(50);
    }

    //read from master
    Serial.readBytes(buffer, 3);
    analogValues[0] = buffer[0];
    analogValues[1] = buffer[1];
```

```
    analogValues[2] = buffer[2];  // blocked indicator

    digitalWrite(flagSlave, LOW);


    // update the freeboard dashboard
    for (int i = 3; i < 9; i++) {

      if (analogValues[0] == i - 3)
        analogValues[i] = 255;
      else
        analogValues[i] = 0;
    }

    if (analogValues[0] != 13 && analogValues[0] != 255) {
      twt = "Position: ";
      twt +=  tmp;
      twt += ", Destination: ";
      twt += String(analogValues[0], DEC);
      twt += "    Stamp: ";
      twt += String(random(rand())); // random seed
      tweet(twt);
      tmp = String(analogValues[0], DEC);
    }
  }
}
```

Once a transmission cycle has been completed the values in an array *analogValues[]* are set according with regard to the current position of the car and whether it has been obstructed along the way which caused a delay. These values are then used when updating the online IoT dashboard in Freeboard as described fully in the additional feature part in the development section of this report. A String to be sent as a tweet is also constructed and the function to send the tweet is also called upon to post the tweet upon the Twitter account of the robot.

Master – Server functionality

The *sync_master* thread is the first tab in the master MSP432, therefore the global variables are declared in this tab so that they are public to all the tabs in the board. The major variables that are declared here are the flags that are used to initiate or stop another thread, the variables that are acting as flags are as follows:

```
boolean hold=false;
boolean destination=false;
boolean start=true; // set this false after moving off
boolean send = false;

int blocked=0;
```

The *hold* Boolean variable is used to start and stop the *DriveTab* so that the robot doesn't repeat the same current and next position route. The *destination* Boolean variable is used to

halt the communication between the server and the Wi-Fi shield that is mounted on the car. The *start* Boolean variable is used to acquire a slightly different functionality to the functionality that is achieved during the rest of the run. This variable is set to *false* once the robot reaches the first destination which is checkpoint 0 so that the robot can start to function normally as it was originally meant to. This was only done as a counter measure for a problem that was encountered during the development of Mr. Robot. The *send* Boolean variable is used to initiate the master-slave communication which in turn results in the communication between Mr. Robot and the internet (Twitter and Dashboard). The functionality of the flags were really important in this project due to their ability of starting and ending a process which was very much useful in this case since there were a couple of tabs or threads (three to be precise) running in parallel.

The server functionality was achieved through just two functions in the *sync_master* tab where one sends the group number and the current position of the car on the track to the server and the other function receives and stores the response from the server in a local variable so that the response can be parsed through and extracted of any relevant information that are needed for the operation or the run of the robot such as the next checkpoint number. The function that sends information to the server is as follows:

```
//Send message to Server
  void SendToServer(int pos){
      String messageToSend = "POST
/mobilerobotics/api/position/tag/?group=2&pos=";  // append string to send
to position to server
      messageToSend += String(pos);
      messageToSend += " HTTP/1.1";
      client.println(messageToSend);
      client.println("Host: 192.168.1.2");
      client.println();
  }
```

The group number and the current position of the robot was sent to the server by simply created a *String* variable with the following sentence:

```
"POST /mobilerobotics/api/position/tag/?group=2&pos="
```

where the group number was entered straight after the word *group* in the sentence and the current position which varies every time the robot is trying to communicate with the server due to the motion of the car along the track instructed by the server the previous it was in contact. Since the current position number always changes, it was added to the end of the sentence after the definition of the variable like as follows:

```
messageToSend += String(pos);
messageToSend += " HTTP/1.1";
```

And the sentence was ended with specifying the version of HTML that was used because the that's how the server knows when to stop reading in from the client. The host number and the string was sent to the server using in the following format:

```
client.println(messageToSend);
client.println("Host: 192.168.1.2");
client.println();
```

The response from the server was read and stored in a local variable in the function shown below:

```
//Read reply from Server
  int ReadFromServer(){
      String tmp="";
      char buffer[255] = {0}; while (!client.available()) {}; //Wait for
connection to be available...
      if (client.available()) {
        client.read((uint8_t*)buffer, client.available());
      }
      String finalMessage = buffer;
      tmp += String(finalMessage.charAt(178));
      if(tmp.equals("T")){ return -1; }
      int pos = tmp.toInt();
      return pos;
  }
```

The first few lines in the function is declaring variable to store the response in and waiting for the connection to be established. Once the server and the client are connected, the client reads in the response and stores it in a buffer so that only the relevant information is extracted and stored. This is done because the server sends a lot of information to ensure the message has been reached to the client safely without being lost or corrupted and the only information Mr. Robot needs is the destination checkpoint number. When the last destination has been reached and this position number is sent to the server, the server returns the word *"Target"* in place of the next destination number to let the robot know that this is the final destination and terminate the communication with the server. Therefore, when the response contains *T* the position returned by the function is -1 which is invalid representing the termination of connection with the server.

Two Wi-Fi shields were utilised in this project, one connected to the server and the other connected to the DCU-guest-WiFi to access the internet for publishing to online dashboard and tweeting. The code that was used to connect to the Wi-Fi involved three main libraries:

1. SPI
2. WiFi
3. WiFiClient

And the functions in these libraries were utilised to check if the Wi-Fi shield was connected to not etc.

NB: The flow diagrams for each tab/thread can be viewed in the Appendix section of this report.

6. Problems Encountered

1. One of the main problems encountered was that the motors were not rotating at the same speed, however this problem was solved by installing encoders to match the speed of one wheel with the other. This was done by measuring the rotation of the two wheels and then offsetting the wheels, this offset was then used to control the level of power supplied to each wheel resulting in the wheels rotating at the same speeds.
2. A problem encountered in the early stages was that the buzzer was unbearably loud which was solved by placing a resistor in series with the buzzer to reduce the voltage intake of the buzzer.
3. When the buzzer was triggered, the Wi-Fi capability started to malfunction, i.e. the buttons on the webpage suddenly turned unresponsive, In order to avoid the malfunction of the buttons on the web page, another tab was allocated for the code that implements the buzzer feature (melody). Flags were then used to enable this tab to execute when requested by the user through the web page (multitasking).
4. A problem faced with the server was not knowing when to stop reading information sent by the server, i.e. knowing what the last checkpoint is: It was found that after the last checkpoint in the defined path, the server sends "Target" as the next checkpoint, which confirms the end of route. This string was utilised to inform the mobile robot to stop sending any more information to the server. This was achieved using an *'if'* statement to check if the index number which was evaluated earlier contains the character *'T'*.
5. Another one of the main problems that the team was faced with was getting the car to slow done as it approached an obstacle before it finally stopped, the solution to this problem was to get as close as possible to an obstacle; the best approach was to slowly decrement the speed of the car as it approached an obstacle. This was done by changing the speed then going into a *while* loop which will make the car run at the speed prescribed until the distance sensor is greater than a predefined threshold value. Then changing the motor speed again and moving to a second *while* loop. This was done three times to give a smooth flow as the car approached an obstacle and finally when the sensor was greater than 900 it would stop the car leaving barely any space between the car and an obstacle resulting in the ultimate stop of the car.
6. The last problem encountered was with the innovative feature of Mr. Robot. The problem was that the Wi-Fi shield of the slave would update the Twitter account really slow which sometimes hindered the functionality of the board with PubNub and Freeboard; this problem was overtook by simply reducing the load on the slave and the amount of tweets per run.

7. Code

Master.ino

```
// -- globals ------------------------------------
// current position and next position
    int pos=4;
    int nextPos=0;

    int i = 0;

// default values
    boolean hold=false;
    boolean destination=false;
    boolean wait=true;
    boolean start=true; // set this false after moving off
    boolean send = false;

    int blocked=0;

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

// libraries
    #ifndef __CC3200R1M1RGC__
    #include <SPI.h>
    #endif
    #include <WiFi.h>
    #include <WiFiClient.h>

// network credentials
    char ssid[] = "NETGEAR72";
    char password[] = "littlecello367";
    int keyIndex = 0;
    uint16_t port = 80;     // port number of the server
    IPAddress server(192, 168, 1, 2);   // IP Address of the server
    WiFiClient client;

// used to sync UART com to slave
    const int flagMaster = 37; // P5.6 - P4.7   on slave
    const int flagSlave = A0; // P5.5 - P2.6 -  on slave

// LED definitions
    #define LED RED_LED
    #define gLED GREEN_LED
    #define LEDb BLUE_LED

void setup() {
    // set up the flag pins
    pinMode(flagSlave, OUTPUT);
    digitalWrite(flagSlave, LOW);


    digitalWrite(LED, HIGH);
    startWiFi();
    digitalWrite(LED, LOW);

    Serial.begin(115200);
```

```
    // allow contact with other MSP432
    establishContact();

   delay(500);
}

void loop() {
    if(i == 0){
      updateGateway();
      i++;
    }

    if(WiFi.status() != WL_CONNECTED){
      digitalWrite(RED_LED, HIGH);       // turn the LED off (LOW is the
voltage level)
    }



    if(nextPos != -1){
         if(destination){
               SendToServer(pos);
               nextPos = ReadFromServer();
               updateGateway();
               miDelay(500);
               hold=false;
               destination=false;
        }
    }
    if(blocked == 255)  updateGateway();
}

void updateGateway(){

      if(pos==13 || nextPos==10) return;

      // used for a loop timeout so wont hang
      unsigned long timer = millis();

      //signal to slave
      digitalWrite(flagMaster, HIGH);

      while(digitalRead(flagSlave)==LOW            &&            millis()-
timer<6000){digitalWrite(LEDb,         HIGH);delay(50);digitalWrite(LEDb,
LOW);delay(50);}

      // write data
      Serial.write(nextPos);
      Serial.write(nextPos);
      Serial.write(blocked);

      digitalWrite(flagMaster,LOW);

      // Wait forslave flag to go LOW and signal end of transmission
      while(digitalRead(flagSlave)==HIGH           &&            millis()-
timer<6000){digitalWrite(LEDb,         HIGH);delay(50);digitalWrite(LEDb,
LOW);delay(50);}

}
```

```
void establishContact() {
  while (Serial.available() <= 0) {
      Serial.println('A');
      digitalWrite(LED, HIGH);
      delay(150);
      digitalWrite(LED, LOW);
      delay(150);
  }
  Serial.flush();
}

//Read reply from Server
  int ReadFromServer(){
      String tmp="";
      char buffer[255] = {0}; while (!client.available()) {}; //Wait for
connection to be available...
      if (client.available()) {
        client.read((uint8_t*)buffer, client.available());
      }
      String finalMessage = buffer;
      tmp += String(finalMessage.charAt(178));
      if(tmp.equals("T")){ return -1; }
      int pos = tmp.toInt();
      return pos;
  }

//Send message to Server
  void SendToServer(int pos){
      String              messageToSend              =              "POST
/mobilerobotics/api/position/tag/?group=2&pos=";   //  append  string  to
send to position to server
      messageToSend += String(pos);
      messageToSend += " HTTP/1.1";
      client.println(messageToSend);
      client.println("Host: 192.168.1.2");
      client.println();
  }

// prints status of wifi sheild
void printWifiStatus() {
      IPAddress ip = WiFi.localIP();
      long rssi = WiFi.RSSI();
}


void startWiFi(){
      pinMode(RED_LED, OUTPUT);

      WiFi.begin(ssid, password);

      // waiting to connect
      while ( WiFi.status() != WL_CONNECTED) { delay(300);}

      // waiting fo IP
      while (WiFi.localIP() == INADDR_NONE) {  delay(300);}

      uint8_t tries = 0;
      while (client.connect(server, port) == false) {
        if (tries++ > 100) { while(1); }
        delay(100);
```

```
        }
}
```

Slave.ino

```
#include <SPI.h>
#include <WiFi.h>
#include <PubNub.h>
#include <aJSON.h>
#include <Temboo.h>
#include "TembooAccount.h"

String twt;

// used to sync UART com to master
const int flagSlave = 39; // P2.6
const int flagMaster = A6;   // P4.7


#define LED RED_LED
#define LEDg GREEN_LED
#define LEDb BLUE_LED

//wifi credentials
char ssid[] = "DCU-guest-WiFi";
char password[] = "";
String tmp = "Taxi Stand";

static int keyIndex = 0;

// pubnub credentials
const static char pubkey[] = "pub-c-eddb4434-2478-47de-a4c9-2e3d097c19c9";
const static char subkey[] = "sub-c-b216c686-f395-11e6-a0e8-02ee2ddab7fe";
const static char channel[] = "Data1"; // applicaion specific channell

#define NUM_CHANNELS 9

int analogValues[NUM_CHANNELS];
char buffer[3];
boolean setupComplete = false;
unsigned long wdt;
int ct = 0;

void setup()
{
  // set up the flag pins
  pinMode(flagMaster, OUTPUT);
  //pinMode(flagSlave,INPUT);
  digitalWrite(flagMaster, LOW);

  // spin lock if no sheild present
  if (WiFi.status() == WL_NO_SHIELD)
    while (true) {
      digitalWrite(LED, HIGH);
      delay(150);
      digitalWrite(LED, LOW);
      delay(150);
```

```
    }

  digitalWrite(LED, HIGH);

  // attempt to connect to Wifi network-> red LED off when connected
  digitalWrite(LED, HIGH);
  digitalWrite(LEDg, LOW);
  int status;
  do {
    status = WiFi.begin(ssid);
    //status = WiFi.begin(ssid, password);
  } while (status != WL_CONNECTED);
  digitalWrite(LED, LOW);

  digitalWrite(LEDb, HIGH);
  Serial.begin(115200);
  char buffer[1];
  do {
    Serial.readBytes(buffer, 1);
  } while (buffer[0] != 'A');

  for (int i = 0; i < 5; i++)
    Serial.print('A');
  Serial.flush();
  digitalWrite(LEDb, LOW);

  PubNub.begin(pubkey, subkey);

  wdt = millis();

  analogValues[0] = 0;
  analogValues[1] = 0;
  analogValues[2] = 0;  // blocked indicator

  analogValues[3] = 255;

  for (int i = 4; i < 9; i++)
    analogValues[i] = 0;

  publishToCloud();
}

void loop() {

  readMaster();

  if (millis() - wdt > 500) {
    publishToCloud();
    wdt = millis();
  }

  digitalWrite(LED, HIGH);
  delay(50);
  digitalWrite(LED, LOW);
  delay(50);
}

void readMaster() {

  // execute read if signaled
```

```
  if (digitalRead(flagMaster) == HIGH) {

    // used for a loop timeout so wont hang
    unsigned long timer = millis();

    digitalWrite(flagSlave, HIGH);

    while (digitalRead(flagMaster) == HIGH && millis() - timer < 6000) {
      digitalWrite(LEDb, HIGH);
      delay(50);
      digitalWrite(LEDb, LOW);
      delay(50);
    }

    //read from master
    Serial.readBytes(buffer, 3);
    analogValues[0] = buffer[0];
    analogValues[1] = buffer[1];
    analogValues[2] = buffer[2];  // blocked indicator

    digitalWrite(flagSlave, LOW);

    // update the freeboard dashboard
    for (int i = 3; i < 9; i++) {

      if (analogValues[0] == i - 3)
        analogValues[i] = 255;
      else
        analogValues[i] = 0;
    }

    if (analogValues[0] != 13 && analogValues[0] != 255) {
      twt = "Position: ";
      twt +=  tmp;
      twt += ", Destination: ";
      twt += String(analogValues[0], DEC);
      twt += "    Stamp: ";
      twt += String(random(rand())); // random seed
      tweet(twt);
      tmp = String(analogValues[0], DEC);
    }
  }
}

// format sensor values into aJson object and publish to cloud
void publishToCloud() {

  WiFiClient *client;

  // formatted ajson object to be sent
  aJsonObject *msg = aJson.createObject();

  // create sender object and add to message object
  aJsonObject *sender = aJson.createObject();
  aJson.addStringToObject(sender, "name", "Arduino");
  aJson.addItemToObject(msg, "sender", sender);

  //ajson object which holds values read from previous function call
  aJsonObject *analog = aJson.createIntArray(analogValues, NUM_CHANNELS);
```

```
  // add reading to message object
  aJson.addItemToObject(msg, "analog", analog);

  // convert json object to charr array which pubnub api will understand
  char *msgStr = aJson.print(msg);

  // delete formatted json object
  aJson.deleteItem(msg);

  // trim buffer to size
  msgStr = (char *) realloc(msgStr, strlen(msgStr) + 1);

  // publish message to pubnub to cloud over wifi connection
  client = PubNub.publish(channel, msgStr);

  // free up memory assigned to msgStr
  free(msgStr);

  // flash red LED if client disconnects
  if (!client) {
    digitalWrite(LED, HIGH);
    delay(1250);
    digitalWrite(LED, LOW);
    delay(250);
  }
  client->stop();
}

void tweet(String tweet) {

  WiFiClient tweetClient;

  //Serial.println("Tweeting");
  TembooChoreo StatusesUpdateChoreo(tweetClient);

  // Invoke the Temboo client
  StatusesUpdateChoreo.begin();

  // Set Temboo account credentials
  StatusesUpdateChoreo.setAccountName(TEMBOO_ACCOUNT);
  StatusesUpdateChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  StatusesUpdateChoreo.setAppKey(TEMBOO_APP_KEY);

  // Set Choreo inputs

  StatusesUpdateChoreo.addInput("StatusUpdate", tweet);
  String ConsumerKeyValue = "YejB2vm1QXJYxwNg5pS9WEF70";
  StatusesUpdateChoreo.addInput("ConsumerKey", ConsumerKeyValue);
  String          AccessTokenValue          =          "836348481337950209-
1AX4CbEPEhUNFghfpgdA93GgHqu18Vy";
  StatusesUpdateChoreo.addInput("AccessToken", AccessTokenValue);
  String                        ConsumerSecretValue                        =
"7OpIYZktDTRXurmDlKeEEAm9rNjX4FOGWlPHccdC7atMV9wLTc";
  StatusesUpdateChoreo.addInput("ConsumerSecret", ConsumerSecretValue);
  String                      AccessTokenSecretValue                       =
"3lVjlUZopIpDk7VovQjNABZ8erkdPA9RQnmBGjf1wOP4m";
  StatusesUpdateChoreo.addInput("AccessTokenSecret",
AccessTokenSecretValue);

  // Identify the Choreo to run
```

```
  StatusesUpdateChoreo.setChoreo("/Library/Twitter/Tweets/StatusesUpdate");

  // Run the Choreo; when results are available, print them to Serial
  // Boolean input to tell library to use HTTPS
  StatusesUpdateChoreo.run();

  StatusesUpdateChoreo.close();
  tweetClient.stop();
}
```

DriveTab.ino

```
// Motor Pin Declarations
    const int motor_left_PWM = 34;     // P2.6
    const int motor_right_PWM = 38;    // P2.5
    const int motor_left_phase = 40;   // P2.7
    const int motor_right_phase = 39; // P2.6

// holds sensor values
    int a[5] = {0,0,0,0,0};

//Analog pins            P4.7, P4.4, P4.2, P4.0, P6.1
    int AnalogPin[5] = {A6, A9, A11, A13, A14};

//                                                                  PID
variables
    float Kp = 35;          // 50 -> speed=255
    float Ki =  0;
    float Kd = 10;          // 30
    int pidVal = 0;

// cross track error
    float err = 0;
    float prevErr = 0;

// integral and differential error
    float intErr = 0;
    float diffErr = 0;

// motor speed
    int speed=215;
    int prevSpeed;
    double offsetL=1;
    double offsetR=0.95;

// sensor threshold
    int thres= 100; // tested at 100 -> maybe included varying threshold

// indicates when turning, helpes bring car back to line if it deviates
    boolean hardTurn = false;

// ir sensor pin
    int sensorpin=2;
    int val=0;

// north =  1 -> robot facing 5/north
// north = -1 -> robot facing away from 5/south
```

```
    int north= -1; // starting direction



void setup()
{
 // put your setup code here, to run once:
  Serial.begin(115200);
}



void loop(){
      if(!hold){ hold=true;  nextpos(pos, nextPos); }
}


void PID(){

 // stop if obstice in the way
    if(analogRead(sensorpin)>650){

          blocked=255;
          // stop
          analogWrite(motor_right_PWM,0);
          analogWrite(motor_left_PWM, 0);

          // wait for path to clear
          while(analogRead(sensorpin)>650){}
          blocked=0;

    }

// Update errors
    prevErr=err;
    err=readError();
    intErr+=err;
    diffErr=err-prevErr;

// calculate pid value
    pidVal = int(Kp*err +Kd*diffErr +Ki*intErr);

// ensure value within allowable range
    if( pidVal>  speed )  pidVal =  speed;
    if( pidVal< -speed )  pidVal = -speed;

// turn by subtracting from wheel speed
    if(pidVal>0){
      analogWrite(motor_right_PWM, speed*offsetR);
      analogWrite(motor_left_PWM, (speed - abs(pidVal))*offsetL);
    }
    else{
      analogWrite(motor_right_PWM, (speed - abs(pidVal)) *offsetR );
      analogWrite(motor_left_PWM, speed*offsetL);
    }
}

float readError(){

// temp variable
```

```
   float tErr=0;

// Read and set sensor values to 0 ar 1
   for (int i=0; i<5; i++){
      int tmp=analogRead(AnalogPin[i]);
      if(tmp > thres) a[i]=0;
      else a[i]=1;
    }

// deduce error
    if(a[0]==0 && a[1]==0 && a[2]==0 && a[3]==0 && a[4]==1) // 00001
        {tErr=4; hardTurn = true;}
     else
    if(a[0]==0 && a[1]==0 && a[2]==0 && a[3]==1 && a[4]==1) // 00011
        {tErr=3; hardTurn = true;}
     else
    if(a[0]==0 && a[1]==0 && a[2]==0 && a[3]==1 && a[4]==0) // 00010
        {tErr=2; hardTurn = true;}
     else
    if(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==1 && a[4]==0) // 00110
        {tErr=1; hardTurn = false;}
     else
    if(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0 && a[4]==0) // 00100
        {tErr=0; hardTurn = false;}
     else
    if(a[0]==0 && a[1]==1 && a[2]==1 && a[3]==0 && a[4]==0) // 01100
        {tErr=-1; hardTurn = false;}
     else
    if(a[0]==0 && a[1]==1 && a[2]==0 && a[3]==0 && a[4]==0) // 01000
        {tErr=-2; hardTurn = true;}
     else
    if(a[0]==1 && a[1]==1 && a[2]==0 && a[3]==0 && a[4]==0) // 11000
        {tErr=-3; hardTurn = true;}
     else
    if(a[0]==1 && a[1]==0 && a[2]==0 && a[3]==0 && a[4]==0) // 10000
        {tErr=-4; hardTurn = true;}
     else
     // crossing line
    if(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==1 && a[4]==1) // 11111
        {tErr=0; hardTurn = false;}
     else
    if(a[0]==0 && a[1]==0 && a[2]==0 && a[3]==0 && a[4]==0){
       if(hardTurn==true)
          tErr=5*prevErr/abs(prevErr); // max error when deviated from
line
       else
          tErr=5; // used when rotating
    }
     else
    if(hardTurn==true)
        tErr=  5*prevErr/abs(prevErr); // max error when deviated from
line

    return tErr;
}

// Motor control methods
void forward(){
     digitalWrite(motor_left_phase, HIGH);
     digitalWrite(motor_right_phase, LOW);
```

```
      analogWrite(motor_left_PWM, speed);
      analogWrite(motor_right_PWM, speed);
}

void backward(){
      digitalWrite(motor_left_phase, LOW);
      digitalWrite(motor_right_phase, HIGH);
      analogWrite(motor_left_PWM, speed);
      analogWrite(motor_right_PWM, speed);
}

void ccw(){
      digitalWrite(motor_left_phase, HIGH);
      digitalWrite(motor_right_phase, HIGH);
      analogWrite(motor_left_PWM, speed);
      analogWrite(motor_right_PWM, speed);
}

void cw(){
      digitalWrite(motor_left_phase, LOW);
      digitalWrite(motor_right_phase, LOW);
      analogWrite(motor_left_PWM, speed);
      analogWrite(motor_right_PWM, speed);
}

void stopRobot(){
      digitalWrite(motor_left_PWM, LOW);
      digitalWrite(motor_right_PWM, LOW);
}

void nextpos(int currentpos, int nxtpos){

  if(currentpos == 0) zeroFunc(nxtpos);

  else if(currentpos == 1){ oneFunc(nxtpos); }

  else if(currentpos == 2){ twoFunc(nxtpos);  }

  else if(currentpos == 3){ threeFunc(nxtpos);  }

  else if(currentpos == 4){ fourFunc(nxtpos); }

  else if(currentpos == 5){
    // yet to be implemented
  }

  stopRobot();

// updatae global variables before returning
  destination=true;
  pos = nextPos;
  miDelay(500);

}

// rotation function
//      ahead      -> used to reverse back over line if needed. ie. will
be behind the line at the start so no reverse
//      north =  1 -> robot facing 5
//      north = -1 -> robot facing away from 5/south
```

```
void chkRot(int set){
   /*
    * turn only if facing in direction opposite
    * to that of the desired direction
   */
   if(set*north==-1){
        readError();
        backward(); while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==1 &&
a[4]==1)){ backward(); readError();}   // drive back to the line if ahead
of line
        stopRobot();
        ccw(); miDelay(350);
        while(readError() ==5 ){}
        stopRobot();
    }
}

// implements delay without using hardware delay
void miDelay(unsigned long dLay){
    unsigned long currentMs = millis();
    unsigned long prevms = currentMs;
    while(currentMs < prevms + dLay) currentMs = millis();
}

void moveToNextpoint(){
    forward();
    readError();
    while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==1 && a[4]==1)) PID();
}

// coded for junction beside 1
void moveToJunc_CCW_turn(){
   forward();
   while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==0 && a[4]==0)){ PID();
}// brings to junction
   miDelay(200);
   stopRobot();
   ccw();
   analogWrite(motor_left_PWM, 0.625*speed);
   analogWrite(motor_right_PWM, 0.625*speed);
   miDelay(250);
   while(!(a[0]==0  &&  a[1]==0  &&  a[2]==1  &&  a[3]==0  &&  a[4]==0)){
readError();}
}

// coded for junction beside 1
void moveToJunc_CW_turn(){
   forward();
   while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==1 && a[4]==1)) PID();
// brings to junction
   miDelay(200);
   stopRobot();
   cw();
   analogWrite(motor_left_PWM, 0.625*speed);
   analogWrite(motor_right_PWM, 0.625*speed);
   miDelay(250);
   while(!(a[0]==0  &&  a[1]==0  &&  a[2]==1  &&  a[3]==0  &&  a[4]==0)){
readError();}
   forward();
}
```

```
void passOverLine(){
  miDelay(200);   readError(); // ensures line passed -> update sensor
values
}

void zeroFunc(int nxtpos){

/*
 * start used in chkRot as from start position will be behind line.
 * otherwise driving to the line we can usually assume sensors will be
 * beyond line.
 */
    if(nxtpos == 0){}
    else
    if(nxtpos == 2){  chkRot(-1);         // want facing south
                      moveToNextpoint(); // brings us to 2
                      north=1;           // update direction

    }
    else
    if(nxtpos == 3){  chkRot(-1);         // want facing south
                      moveToNextpoint(); // brings us to 2
                      passOverLine();
                      moveToNextpoint(); // brings us to 3
                      north=1;
    }
    else
    // this is a bit flakey with the increased speed = 185
    if(nxtpos == 1){  chkRot(-1);           // want facing south
                      moveToJunc_CCW_turn();
                      moveToNextpoint();    // brings us to 1
                      north=1;              // update direction
    }
     else
    if(nxtpos == 4){
                      chkRot(1);          // want facing north
                      moveToNextpoint(); // brings us to 4
                      north=1;           // update direction
    }
    else
    if(nxtpos == 5){  chkRot(-1);           // want facing south
                      moveToJunc_CCW_turn();
                      Kp = 15;
                      moveToNextpoint();    // brings us to 1
                      north=1;              // update direction
                      Towall();             // Towards the wall
                      stopRobot();
    }

}

void oneFunc(int nxtpos){

    if(nxtpos == 1){}
    else
    if(nxtpos == 2){ chkRot(-1);
                     moveToNextpoint(); // brings us to junction
                     miDelay(150);
                     stopRobot();       //turn at junction
```

```
                    ccw();
                    analogWrite(motor_left_PWM, 0.625*speed);
                    analogWrite(motor_right_PWM, 0.625*speed);
                    miDelay(100);
                    while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0 &&
a[4]==0)){ readError();}
                    moveToNextpoint(); // brings us to 2
                    north=1;
    }
    else
    if(nxtpos == 0){ chkRot(-1);
                    moveToNextpoint(); // brings us to junction
                    miDelay(150);
                    stopRobot();
                    cw();
                    analogWrite(motor_left_PWM, 0.625*speed);
                    analogWrite(motor_right_PWM, 0.625*speed);
                    miDelay(100);
                    while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0 &&
a[4]==0)){ readError();}
                    moveToNextpoint(); // brings us to 0
                    north=1;
    }
    else
    if(nxtpos == 4){ if(north==-1){
                        backward();
                        miDelay(300);
                        cw();
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(200);
                        readError();
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0
&& a[4]==0)){readError();}
                        stopRobot();
                        miDelay(50);
                        moveToNextpoint();
                        north=-1;
                    }
                    else if(north == 1){
                        moveToNextpoint();
                        miDelay(100);
                        ccw();
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(200);
                        readError();
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0
&& a[4]==0)){readError();}
                        stopRobot();
                        miDelay(50);
                        moveToNextpoint();
                        north=-1;
                    }
    }
    else
    if(nxtpos == 3){ if(north==-1){
                        backward();
                        miDelay(300);
                        ccw();
```

```
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(200);
                        readError();
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0
&& a[4]==0)){readError();}
                        stopRobot();
                        miDelay(50);
                        moveToNextpoint();
                        north=-1;
                    }
                    else if(north == 1){
                        moveToNextpoint();
                        miDelay(100);
                        cw();
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(200);
                        readError();
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0
&& a[4]==0)){readError();}
                        stopRobot();
                        miDelay(50);
                        moveToNextpoint();
                        north=-1;
                    }

    }
    else
    if(nxtpos == 5){  chkRot(1);
                      ccw();
                      miDelay(70);  // this is probably very dependent on
the speed=210
                      stopRobot();
                      forward();
                      miDelay(150);
                      stopRobot();
                      //readError();
                      //PID();
                      //stopRobot();
                     Towall();         // Towards the wall
    }
}

void twoFunc(int nxtpos){

    if(nxtpos == 1){       // sensors don't really allign at the junction
                      chkRot(-1);
                      moveToJunc_CW_turn();
                      moveToNextpoint();    // brings us to 1
                      north=1;              // update direction
    }
    else
    if(nxtpos == 2){}
    else
    if(nxtpos == 0){
                      chkRot(-1);
                      moveToNextpoint();    // brings us to 0
                      north=1;              // update direction
    }
```

```
    else
    if(nxtpos == 4){
                        chkRot(-1);
                        moveToNextpoint(); // brings us to 0
                        passOverLine();    // updating error
                        moveToNextpoint(); // brings us to 4
                        north=1;           // update direction
    }
    else
    if(nxtpos == 3){
                        chkRot(1);
                        moveToNextpoint(); // brings us to 3
                        north=1;           // update direction
    }
    else
    if(nxtpos == 5){  chkRot(-1);
                        moveToJunc_CW_turn();
                        miDelay(50);
                        Kp = 15;
                        moveToNextpoint();    // brings us to 1
                        north=1;              // update direction
                        Towall();             // Towards the wall
    }
}

void threeFunc(int nxtpos){

    if(nxtpos == 1){
                        chkRot(1);
                        forward();
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==1 &&
a[4]==1)) PID(); // brings to junction
                        analogWrite(motor_left_PWM, speed);
                        analogWrite(motor_right_PWM, speed);
                        miDelay(250);
                        ccw();
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(100);
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0 &&
a[4]==0)){ readError();}
                        moveToNextpoint();
                        stopRobot();
                         north=-1;
    }
    else
    if(nxtpos == 2){
                        chkRot(-1);
                        moveToNextpoint(); // brings us to 2
                        north=-1;          // update direction
    }
    else
    if(nxtpos == 0){
                        chkRot(-1);
                        moveToNextpoint(); // brings us to 2
                        passOverLine();    // updating error
                        moveToNextpoint(); // brings us to 0
                        north=1;
    }
    else
```

```
    if(nxtpos == 4){
                        chkRot(1);
                        moveToNextpoint(); // brings us to junction
                        passOverLine();    // updating error
                        moveToNextpoint(); // brings us to 4
                        north=-1;
    }
    else
    if(nxtpos == 3){}
    else
    if(nxtpos == 5){
                        chkRot(-1);
                        moveToNextpoint(); // brings us to junction
                        moveToJunc_CW_turn();
                        moveToNextpoint();    // brings us to 1
                        moveToNextpoint(); // brings us to 1
                        Towall();          // Towards the wall
    }
}


void fourFunc(int nxtpos){

  if(nxtpos == 0){
                        chkRot(-1);
                        moveToNextpoint();      // brings us to 0
                        north=-1;               // update direction
  }
  else
  if(nxtpos == 1){
                        chkRot(1);
                        forward();
                        while(!(a[0]==1 && a[1]==1 && a[2]==1 && a[3]==0 &&
a[4]==0)) PID(); // brings to junction
                        analogWrite(motor_left_PWM, speed);
                        analogWrite(motor_right_PWM, speed);
                        miDelay(250);
                        cw();
                        analogWrite(motor_left_PWM, 0.625*speed);
                        analogWrite(motor_right_PWM, 0.625*speed);
                        miDelay(100);
                        while(!(a[0]==0 && a[1]==0 && a[2]==1 && a[3]==0 &&
a[4]==0)){ readError();}
                        moveToNextpoint();
                        stopRobot();
                        north=-1;
                        // starts south ends south
  }
  else
  if(nxtpos == 2){
                        chkRot(-1);
                        moveToNextpoint();      // brings us to 0
                        passOverLine();         // updating error
                        moveToNextpoint();      // brings us to 2
                        north=1;
  }
  else
  if(nxtpos == 3){
                        chkRot(1);
                        moveToNextpoint();      // brings us to 3
```

```
                        north=-1;
  }
  else
  if(nxtpos == 4){}
  else
  if(nxtpos == 5){
                    chkRot(-1);
                      moveToNextpoint(); // brings us to junction
                      passOverLine();
                      moveToJunc_CCW_turn();
                      moveToNextpoint();    // brings us to 1
                      moveToNextpoint(); // brings us to 1
                      Towall();          // Towards the wall
  }
}

void Towall(){
    analogWrite(motor_left_PWM, speed*offsetL);
    analogWrite(motor_right_PWM, speed*offsetR);
    while(analogRead(sensorpin)<100){ miDelay(40);}

    analogWrite(motor_left_PWM, 0.75*speed*offsetL);
    analogWrite(motor_right_PWM, 0.75*speed*offsetR);
    while(analogRead(sensorpin)<250){ miDelay(40);}

    analogWrite(motor_left_PWM, 0.5*speed*offsetL);
    analogWrite(motor_right_PWM, 0.5*speed*offsetR);
    while(analogRead(sensorpin)<500){ miDelay(40);}

    analogWrite(motor_left_PWM, 0.15*speed*offsetL);
    analogWrite(motor_right_PWM, 0.15*speed*offsetR);
    while(analogRead(sensorpin)<875){ miDelay(40);}

    stopRobot();
}
```

TembooAccount.h

```
#define TEMBOO_ACCOUNT "ee303mrrobot"  // your Temboo account name
#define TEMBOO_APP_KEY_NAME "myFirstApp"  // your Temboo app key name
#define   TEMBOO_APP_KEY    "b2hQnDqcqi9NvPG6iFHCtABWhvRUQ0yG"    //   your
Temboo app key
```

8. Bibliography

[1] P. Manual, “https://www.dfrobot.com,” [Online]. Available: https://www.dfrobot.com/wiki/index.php/2WD_Mobile_Platform_for_Arduino_(SKU:ROB0005). [Accessed 20 02 2017].

[2] T. -. 2. M. Platform, “http://hanjindata.lgnas.com/,” [Online]. Available: http://hanjindata.lgnas.com:10000/myweb/P0085/P0085.pdf.

[3] T. I. MSP432, “http://www.ti.com/,” [Online]. Available: http://www.ti.com/tool/msp-exp432p401r. [Accessed 15 02 2017].

[4] M. M. B. Easy, “http://www.mouser.ie/,” [Online]. Available: http://www.mouser.ie/new/Texas-Instruments/ti-msp432-microcontrollers/.

[5] T. I. M. Datasheet, “Texas Instruments,” [Online]. Available: http://www.ti.com/lit/ug/slau597c/slau597c.pdf.

[6] anonymous, “https://en.wikipedia.org,” [Online]. Available: https://en.wikipedia.org/wiki/DC-to-DC_converter.

[7] M. P. Solutions. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/13d7/0900766b813d7b66.pdf.

[8] Pololu, “Pololu,” [Online]. Available: https://www.pololu.com/product/2135.

[9] T. I. H.-B. Datasheet. [Online]. Available: https://www.pololu.com/file/0J570/drv8835.pdf.

[10] Vishay, “http://www.robotshop.com,” [Online]. Available: http://www.robotshop.com/media/files/PDF/vishay-TCRT5000L-infrared-reflector-specs.pdf.

[11] batchloaf, “https://robosumo.wordpress.com,” [Online]. Available: https://robosumo.wordpress.com/2017/02/15/getting-started-with-the-tcrt5000-ir-sensor-and-a-switch-input/.

[12] CircuitDigest, “https://circuitdigest.com,” [Online]. Available: https://circuitdigest.com/microcontroller-projects/line-follower-robot-using-arduino.

[13 Annonmous, “google Docs,” [Online]. Available:

] https://docs.google.com/document/d/1Q98DwYL-a2PifLR8oZfz5VTut_BV1MSLEdd_-fFHtUo/edit?hl=en_US.

[14] T. Agarwal. [Online]. Available: https://www.elprocus.com/the-working-of-a-pid-controller/.

[15] [Online]. Available: http://www.pcbheaven.com/wikipages/PID_Theory/.

[16] T. I. CC3100. [Online]. Available: http://www.ti.com/lit/ug/swru371b/swru371b.pdf.

[17] T. I. Threads. [Online]. Available: http://processors.wiki.ti.com/index.php/CC31xx_&_CC32xx.

[18] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Sensor.

[19] W. P. Sensor. [Online]. Available: https://en.wikipedia.org/wiki/Proximity_sensor.

[20] Pubnub. [Online]. Available: https://www.pubnub.com.

[21] CODEPATH. [Online]. Available: https://guides.codepath.com/android/Real-time-Messaging.

[22] Obviate. [Online]. Available: https://obviate.io/2015/08/12/so-many-iot-software-platforms-where-to-begin/.

9. Appendix

Competition day flier

Mr. Robot Analytics Stream

Join in a live data stream on your platform of choice,

- Twitter
- Freeboard

Twitter

A location stream will be updated to twitter using Temboo. Mr. Robot has his own twitter account, please give him a follow and keep up to date on the competition progress from anywhere in the world. Please note twitter is slow to process tweets and ocassionally blocks tweets.

Twitter Username: @EE303_Group_2

Freeboard

Using a subscription stream sent to Pubnub an online dashboard will keep you informed of some key metrics of the car as he makes his journey. To join in and view the dashboard on any device from anywhere in the world in real time,

1. Navigate to

   freeboard.io

2. Login

   Username: **racket**
   Password: **mrcarbot**

3. Click on

   Mr Robots dashboard

Some Extra Features

- The two MSP boards communicate using Serially using UART through the TX and RX header pins. Two IO pins between the boards are used to synchronis the communication in a Master/Slave, Call/Response customized protocol.

- Mr.Robot uses IR sensors boards mounted underneath the car to set an initial low speed offset which is then used throughout the run improve the cars straightline tracking.

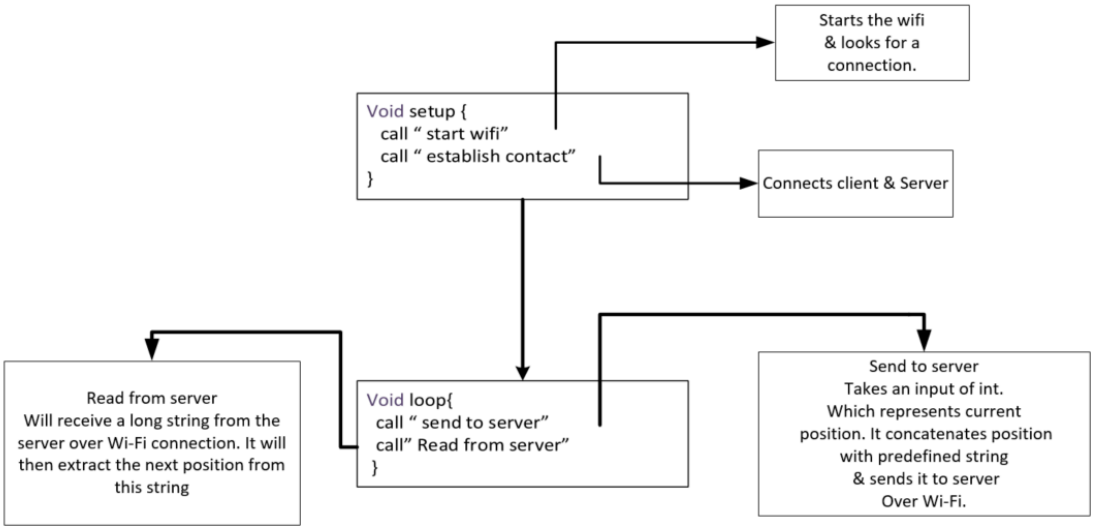
Master/Slave Protocol

- Master and slave Flags initially low.

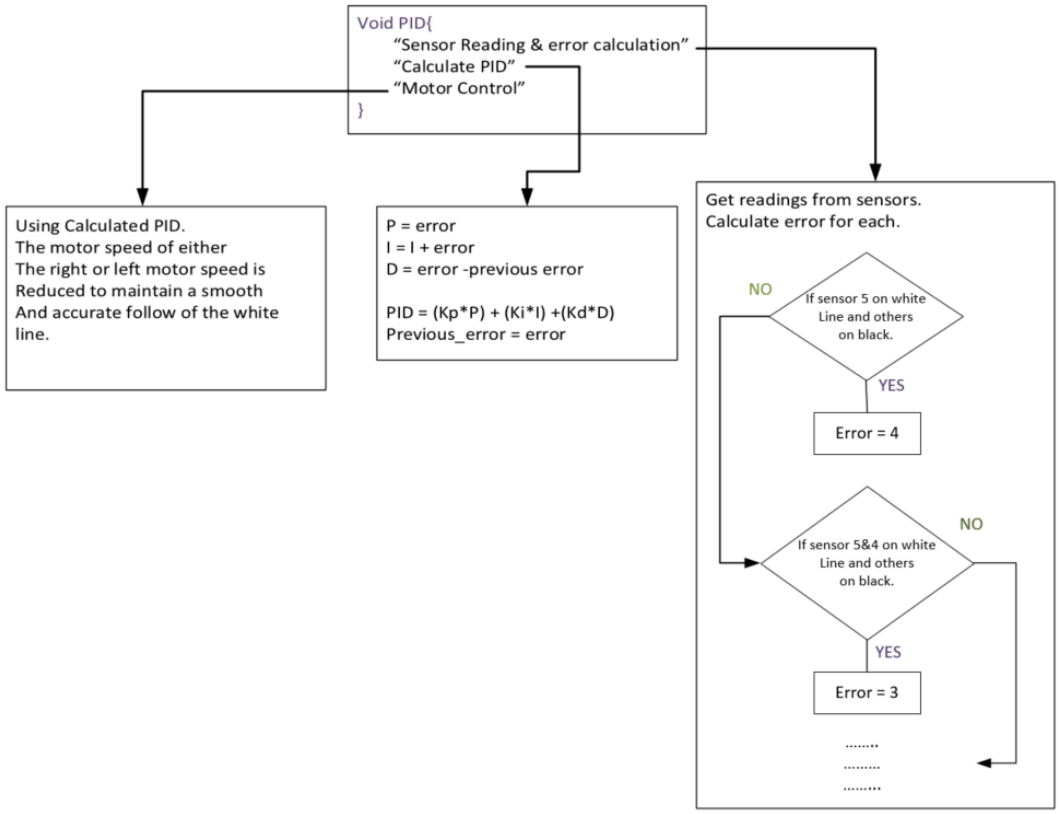
Master	**Slave**
	Wait
masterFlag High	
Wait	
	Read masterFlag High
	slaveFlag High
	Wait
Read slaveFlag High	
sendPayload	
masterFlag Low	
Wait	
	Read masterFlag Low
	read payload
	slaveFlag Low
	Wait
Read slaveFlag Low	

Flow diagrams

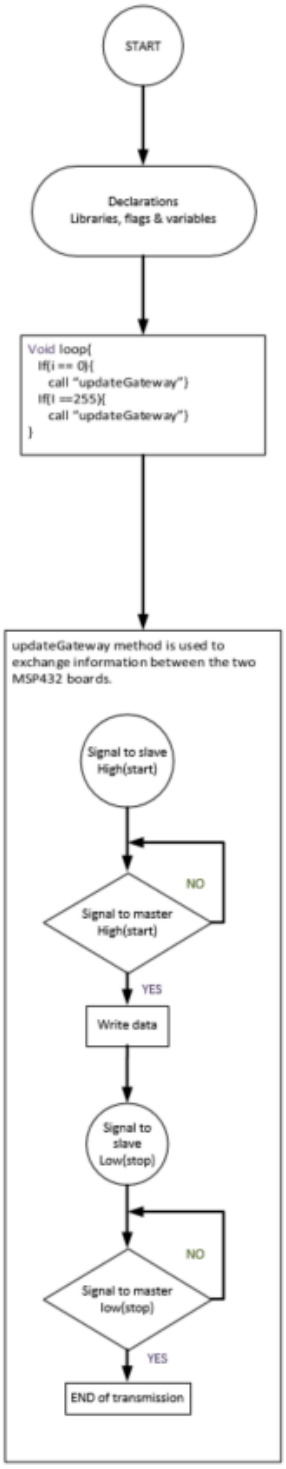
Wi-Fi Connection



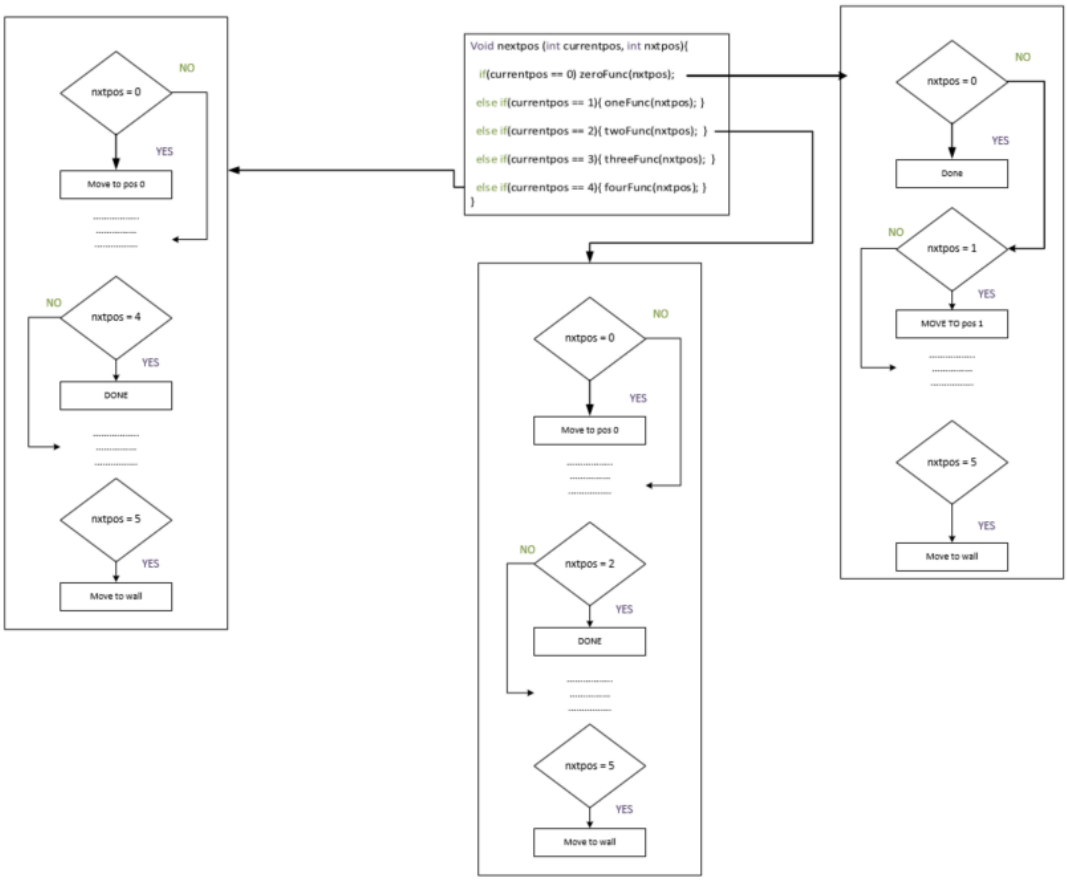
PID



Master-Slave Connection



Drive


Void nextpos (int currentpos, int nxtpos){
if(currentpos == 0) zeroFunc(nxtpos);
else if(currentpos == 1){ oneFunc(nxtpos); }
else if(currentpos == 2){ twoFunc(nxtpos); }
else if(currentpos == 3){ threeFunc(nxtpos); }
else if(currentpos == 4){ fourFunc(nxtpos); }
}
nxtpos = 0
NO
YES
Move to pos 0
nxtpos = 4
DONE
nxtpos = 5
Move to wall
nxtpos = 0
Done
nxtpos = 1
MOVE TO pos 1
nxtpos = 5
Move to wall
nxtpos = 0
Move to pos 0
nxtpos = 2
DONE
nxtpos = 5
Move to wall