

**Embedded Systems Project 2020-21**

**Final Report**

**Title: Heat analysis and analysis of the Procedures undertaken throughout**

**Group Number: 1**

Group members:	ID Number	I confirm that this is the group's own work.
Ahmed Omara	10344777	<input checked="" type="checkbox"/>
Ibrahim Al Brashdi	10513452	<input checked="" type="checkbox"/>
Jiangsheng Wang	10471806	<input checked="" type="checkbox"/>
Mohammad Ali	10510145	<input checked="" type="checkbox"/>
Xavier Millican	10363193	<input checked="" type="checkbox"/>
Zhengyi Zhao	10447097	<input checked="" type="checkbox"/>

**Tutor: Dr Emad Alsusa**

**Date: 12/11/2020**

## **Contents**

1. Executive Summary .....	1
2. Introduction .....	2
3. Final System Component Summary.....	2
4. Team Organisation and Planning.....	7
5. Heats Analysis .....	10
6. References.....	12

## 1. Executive Summary

The hardware used for the buggy include 13 reflectance sensors for tracking the line, a sonar sensor used for distance measurements used for signalling the turn at the wall, a servo motor used to adjust the angle at which the sonar is pointing, two quadrature encodes (one for each wheel) for speed measurement, two motors for the motion of the buggy, one castor wheel as to balance the back of the buggy, a two layer chassis for carrying all the components, an esp32 microcontroller for interfacing the code with the components of the buggy and finally a power bank to supply the power to the buggy while on the track. These were all connected and used for the project. The software included three classes and two supplied header files. The three classes were the servo class, the sonar class, and the encoder class. The two given header files were Relfectance.h and MotorDriver.h. There were three PID controllers used for the system, which were one controller for the line sensors to apply a correction value to the set point in the outer loop depending on the position of the sensors and two controllers (one for each motor) in the inner loop to adjust the speed of the motors accordingly, so as to make controlled speed increases or decreases on the ramp or when deviations in speed occur.

The main objectives of the project included MATLAB simulation, such that the buggy could run in a simulated world to move forwards, turn left and right, and go up a ramp while following a line. Another main objective was transferring this into C++ to code the buggy. There were issues in the team at some points, where there was miscommunication or late submissions of the different sections of the work. Ultimately these issues were overcome by the project manager and the other team members in the WhatsApp group and over Zoom to make sure all the work was being done and that all was running well. At the start of semester wo, a Gantt chart was made for project planning and time management and throughout each task the task was split between the available team members such that everyone had something that they had to complete toward the project.

In preparation for the heats, the buggy was simulated in MATLAB so as to see if the concept for running the buggy was operational or not. After the buggy showed successful results in the MATLAB simulation it was then coded using Arduino, from which the classes described above were coded, and the inner and outer loops were written with the relevant controllers. Between the final testing slot and the heats certain changes were made, including the method of detecting error from the line sensors and the set speed of the wheels was increased. The main successes of the puzzlebot included the servo motor which allows variations of the angle at which the sonar is pointing, the encoders on the input shaft for increased accuracy and big wheels for increased torque. The drawbacks of the puzzlebot included that the wheels were made out of plastic, which reduced the traction, their size also decreased the maximum speed achievable as well.

## 2. Introduction

This project's main aim was to, as a team, design and make a buggy automatically follow a line on a track. In order to do this, the group had to develop many skills over the first semester, including extensive knowledge on reflectance and sonar sensors, control theory, and software design. These skills were applied to designing the buggy through labs and design reports on several topics: DC motors, reflectance sensors, chassis design and software design. In semester two, in an ideal world these designs would then be applied towards creating and building the buggy initially designed, following a budget and co-ordinating it as a team on-campus.

However, due to COVID-19, this was not possible. Instead, a Puzzlebot (a pre-designed buggy) was supplied to each group, which had to be constructed and coded to follow a line on a track. The functionality of the MATLAB code, including the controller of the buggy, was improved, and tested through 4 Technical Demonstrations, until the system was fully functional and robust. The code developed would then be uploaded to an identical Puzzlebot, where the demonstrators in charge of the ESP project tested it on a track on-campus.

Figure 1 shows the constructed buggy, having been tested round a home-made track. The PCB control module is connected via GPIO pins to a servo motor, a sonar sensor, and a reflectance sensor (All of the GPIO pin connections were supplied in the Puzzlebot manual [1]) and is powered by a power bank. This was all mounted to a predesigned chassis constructed from several mechanical components, two bases and a castor ball.

Team organisation was an extremely important part of the project, as there were many tasks and reports that had to be completed. The team communicated via a WhatsApp group and Zoom meetings were undertaken each week, with internal deadlines needing to be set in advance of the hard deadlines in order to help ensure that each section/task was done on time.

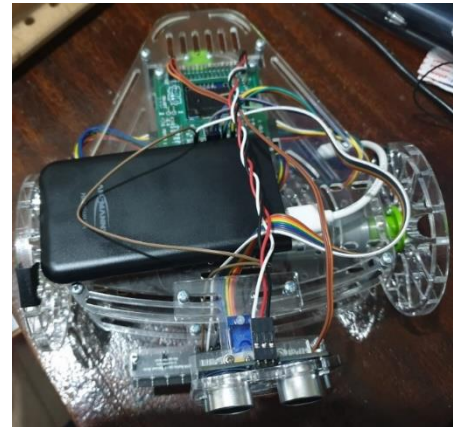


Figure 1 - picture of constructed Puzzlebot

## 3. Final System Component Summary

The hardware components used for the buggy were three different types of sensors, brushed motors, LCD screen, power bank, ESP32 microcontroller and the actual chassis and wheels.

The first type of sensors were the reflectance sensors, there were 13 reflectance sensors (made by Pololu [2] and can be seen in Figure 2) available, of which only seven of them were used; to maximise the gap between adjacent sensors and hence reducing the possibility of two sensors having the same level of reflectance. The rule of these sensors was to measure the amount of light reflected from the track; these measurements were then used to maintain the position of the buggy on the white line on the track.



Figure 2 The board containing the 13 reflectance sensors used for line following

Another type of sensors was the ultrasonic sensor, which has been used to measure the distance between the buggy to any possible obstacle within a three meter range in the direction that the ultrasonic sensor was pointing. The model used here was different than the one in the original design [3], however the operation is the same, with the only difference being that the echo and trigger pins were combined into a single signal pin [4]. This meant that instead of signalling interrupts to 2 separate pins, both the triggering and the actual sensor measurements occurred using the same pin.

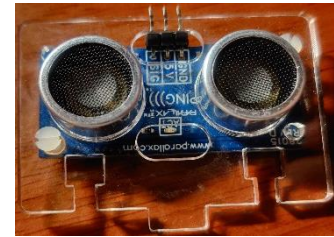


Figure 4 The PING sonar sensor used on

To control the position of the ultrasonic sensor, a servo motor was attached to it, which has the ability to rotate in  $120^\circ$  in either direction [5]. This can be controlled by adjusting the duty cycle of the servo. The servo has a total period of 20000 ms and the high part of that signal varies from 1ms to 2ms, where 1 ms gives  $-120^\circ$  and 2 ms gives  $120^\circ$  [6]. After that the ratio between angle and the time that the signal was high was calculated and hence the angle was determined [6].



Figure 3 The servo motor used on the puzzlebot

The final type of sensors was the quadrature encoder [6], where two encoders were used, one for each wheel. These were used to measure the speed and the direction of each wheel. The way these work is by counting the number of times the light passes through the encoder disk each second, and then by applying the appropriate equation find the speed of the wheel [6]. The 2<sup>nd</sup> encoder in the quadrature encoder is  $90^\circ$  out of phase from the 1<sup>st</sup> encoder disk and this is used for figuring out the direction that the wheel is turning in [6]. This encoder is on the input shaft, unlike the original plan; this allows for more accurate results. These encoders came with the brushed motors, as a single component [7]. The motors have a gear ratio of 34 and hence a maximum speed of 170 rpm [7]. The actual recorded speeds were lower than this, and this is due to friction and the size of the wheel.



Figure 5 The motor with the attached encoder used on the puzzlebot

A rechargeable power bank was used to supply the required power for the puzzlebot, that can provide a maximum of 5 V and 2.1 A. This was used to power the puzzlebot when the buggy was being tested on the track. This was used in replacement to the originally planned battery pack due to the change in drive board and microcontroller.



Figure 6 The power bank used to power the buggy when testing on the track

The last piece of electronic hardware used was the ESP32 microcontroller board. This was used instead of the STM32 [3] and as such required a different platform for coding and uploading the code to the microcontroller. Other than the different pins (that was found in the puzzlebot manual [1]) structure and the platform, the operation was the same and the programming language remained C++. This also included an LCD screen which contained the IP address of the robot and other identifier information; however, this was overridden when the new code was uploaded.



Figure 7 ESP32 microcontroller used on the puzzlebot

The rest of the hardware where the wheels, double layered chassis, and the castor ball. These were purely mechanical parts used to hold everything in place, balance the robot and ensure that the robot moves then the rest of the hardware is operating.

MATLAB simulation was the main advantage that the Puzzlebot had over the first semester software design; it provided a safe, fast, and reliable way of testing and debugging the code before implementing it on the physical buggy [1]. Like any other simulation, however, it does not provide the exact behaviour of the real buggy, but it does provide reasonable data to work with. MATLAB libraries were provided, the functions were given so only the algorithms were testable on MATLAB; In C++, these libraries would have to be constructed via header files or classes.

The main Arduino code consisted of five classes and two PID controllers, two of the five classes were provided as libraries that were the “MotorDriver.h” and the “ReflectanceSensor.h” [1]. Starting with the MotorDriver.h library, this provided a couple of functions that were used to setup the motor pins and channel, setting the motor’s sign, and setting up the PWM base frequency [1]. The ReflectanceSensor.h library was used to setup the sensor pins and initialise a timer to read the sensors values constantly [1].

A sonar class was written to configure the sonar by setting its pin as an output, then setting the pin High, Low, High, and then Low again to emit a short ultrasonic burst and hence triggering the sensor to start sensing. The pin was then configured as an input to receive the echo of the pulse [4]. The time for the pulse to be received by the sonar was used to work out the distance between the buggy and any obstacle within a three meter range [5]. The distance was be read in the main loop using the “GetDistance” function. The method for calculating the distance followed the following formula:

$$\text{sonar distance} = \frac{\text{duration}}{2 \times 2900} \quad (1)$$

Another class was made for setting the servo angle to aim the ultrasonic sensor in a specific direction. This was achieved by configuring the servo pin as an output and setting it to High state for a specific pulse width [5] using the following equation:

$$\text{Pulse width} = (\text{Required angle} + 120) \times \frac{1000}{240} + 1000 \quad (2)$$

Then the servo pin was set to Low state for a period of 20000 – *The Pulse width* [5]. The required servo angle was provided using the “SetServoAngle” function, the angle of the servo and the pulse width can be accessed using the “GetServoAngle” and “GetpulseWidth” respectively.



The last class implanted was the encoder class. For this, interrupts were called every time a rising edge trigger was detected from one of the encoders per wheel [6]. However, as classes cannot contain interrupts directly, the scope function was used to code the ISRs and call the interrupts outside the class, where inside only the member function prototype was coded. The tick rate was obtained by incrementing a counter every time a rising edge was detected for a period of one second, from which the counter was reset [6]. This counter value was then divided by the CPR of the encoder (this value was later divided by four as the CPR of a quadrature encoder is equal to four times the PPR of the quadrature encoder [8]) to obtain the revolutions per second. After that, the value was multiplied by  $2\pi$  to give the angular speed of the wheel. This can be summarised by the following equation:

$$\omega_{\text{wheel}} = \frac{\text{tick rate} \times 2\pi}{\frac{\text{CPR}}{4}} \quad (3)$$

After that multiple members get functions were established to obtain the values of linear speed for the wheels, the angular speeds, and the total linear and angular speeds of the buggy. These can be summed up as the following equations [6]:

$$v_{\text{wheel}} = \omega_{\text{wheel}} r \quad (4)$$

$$\omega = \frac{\omega_r - \omega_l}{r} l \quad (5)$$

$$v = \frac{\omega_r + \omega_l}{2} r \quad (6)$$

For the direction control, the same thing was done for the 2<sup>nd</sup> encoder on the wheel, from which the states of the two encoders on each wheel were compared each time a rising edge trigger was read. This allowed for the direction of the wheel to be accounted for in the encoder measurements [6].

The control mechanism used for the buggy was PID control. In total there were three PID controllers in the software, two for the motors (right and left) and one for the line sensors. The 1<sup>st</sup> controller for the line sensors was placed in an outer loop where the sampling time was 10 ms. The motor PID controllers were placed in the inner loop where the sampling time was 100 ms. The reason the two controllers were placed in two different loops with different sampling times is due to oversampling, where the outer loop samples ten times before sending the data to the inner loop. This allows for fewer issues surrounding data being corrupted at the transition points of the digital signals.

The role of the reflectance sensor controller was to take an average of all of the values received by the reflectance sensor and hence output an appropriate error value. At first a binary value approach was used, but after further testing it was seen that up to four sensors had the chance to be on top of the line at the same time, and this would result in many if else statements giving an arbitrary value to each case. As such, a weighted average approach was used using the following formula:

$$\text{error} = \frac{\sum_{i=1}^{i=N} (\text{sensor\_value}_i) \times i}{\sum_{i=1}^{i=N} (\text{sensor\_value}_i)} \times 1000 - 4000 \quad (7)$$

As mentioned earlier, seven sensors were used and as such here  $N = 7$ . This approach ensures that the error is zero if no sensors are on top of the line and if sensor four is on top of the line. Also, if any of sensors one through three were on top of the line, the value of error would be positive and if sensors five through seven were on top of the line the value of error would be negative. This would mean that the error would automatically differentiate between the right side of the line and the left side of the line, and as such provide the correct output for the controller. The reason this was better than the binary method is that it required fewer lines of code. However, the downside of this was that when two or more sensors are on top of the line, the values for the error do not vary in correlation to the position on the line; rather there are areas where the error is higher near the centre than near the far end of the sensors. This is where the arbitrary method, where arbitrary errors were defined was useful, however, as mentioned earlier, this was time consuming and resulted in many lines of code slowing down the code readability. By comparing both methods, in the end the average approach was deemed to provide more value than the binary approach.

After the value of error was obtained, it was fed into a PID controller as seen below [6]:

$$u = K_p \times error + (u_{prev} + K_i \times error \times T_s) + K_d \left( \frac{error - previous\_error}{T_s} \right) \quad (8)$$

Here  $T_s$  represents the sampling time of the outer loop,  $K_p$ ,  $K_i$  and  $K_d$  are the constants of the PID controller [6].  $u_{prev}$  representing the previous output of the controller from the previous iteration (initially having a value of 0), while  $previous\_error$  is the error from the previous iteration of the controller (initially having a value of 0).

The values of  $K_p$ ,  $K_i$  and  $K_d$  were tuned accordingly as to provide stable operation. After the controller outputted suitable values, these values were then fed in as correction values to the set point for the speed of the buggy. The values were fed in an equal and opposite fashion to the left and right motor speed set points accordingly, where the right wheel was negatively corrected, and the left wheel was positively corrected. This is due to the fact that the output of the controller is positive when the right set of sensors are on the line and negative when the left set of sensors are on the line. When the right sensors are on the line the right wheel should slow down and the left wheel should speed up accordingly, and the opposite should happen when the left sensors are on the line. There was no need to reverse these corrections as the output of the controller swapped signs meaning that the opposite effect on the other side of the line occurred, as was desired.

After the set points for the right and left motors were obtained from the outer loop, the values were sent to the PID controllers in the inner loop to output a PWM signal to the motors. The same formula (8) was used for the PID controller; however, the error was calculated in a different fashion. The error was calculated using the following formula:

$$error = set\_speed - wheel\_speed \quad (3)$$

Where  $set\_speed$  is the set point obtained from the outer loop and  $wheel\_speed$  is the speed obtained from the encoder of the wheel. This equation was used twice. Once for the right wheel and once for the left wheel. Once the two values for error were



obtained, they were put into their respective PID controllers and values for  $K_p$ ,  $K_i$  and  $K_d$  were tuned accordingly, the output of this controller was then fed into the Motorwrite() functions so as to set the duty cycle of the motors and hence the speed of the motors. This controller was especially useful on the ramp as the speed changes on the ramp, and as such the controller automatically adjusts the speed back to the set point.

The sonar sensor was implemented as a single if else condition as it was only needed for the turn and the servo motor has internal PID, so only the set point was specified using the class as mentioned earlier.

This is consistent with the plan from semester one, with the only changes being the addition of the servo motor and the implications of an inner and outer loop for oversampling. The servo was useful in terms of adjusting the angle that the sonar would be pointing at any point in time. The sonar was included in the plan for semester one, but a different model was used, so some research on that was necessary. The size of the buggy was approximately the same as the plan from semester one, such that it was consistent with the original plan. This made it so that the plans from semester one were not hindered too much.

#### **4. Team Organisation and Planning**

In semester one, the group studied the theoretical knowledge required and conducted theoretical analysis of the buggy's motion in an ideal situation. In semester two, this was applied to real experiments of the buggy at home, in addition to running simulations in MATLAB. Due to the COVID-19 pandemic, no access to the physical track on the University's campus was granted. The practical operation of the project was more challenging than designing the buggy during the first semester.

One of the main project objectives was simulating the algorithms in MATLAB before implementing them on the buggy. First, the simulator platform and the libraries were downloaded from Blackboard from which the code was written to set the working functions of the motors to control the speed of each motor and the direction that each motor was turning. The buggy was able to keep moving at a constant speed in a straight line, in addition to turning left or right by changing the speed of one motor, which once visually simulated successfully demonstrated the completion of the initial set-up. Next, it was important to configure the line sensors and the sonar. The function of the line sensors was to detect the line and ensure the buggy was on the correct route. The sonar was used to prevent the buggy from hitting the wall, secondary to the line sensor and ultimately to protect it. Together, these were encoded and simulated on MATLAB until working functions were developed. Then, the control algorithm was developed. PID was used, which meant that the constants (mainly proportional) had to be altered until the optimal conditions for the buggy movement were found. Finally, all was combined, and the buggy was made to follow a specific track; one containing a ramp where the controller would be tested and a wall at the end to turn around after reaching the top. This track was simulated and tested against the code until success was achieved.

Another one of the main project objectives was using C++ code to control the real buggy via Arduino. This was more challenging than the simulations on MATLAB since no one in the group had much experience with Arduino before, and members had to accustom themselves to the change in syntax. The university gave two

buggies to each group; so, two students were able to do experiments at home. When the students received the package of the Puzzlebot, there were many components included, such as brushed motors, a PCB control board, and a chassis along with wheels (mentioned in section 3). Members with the buggy need to assemble the components in addition to creating a track using black paper and white tape, highlighting practical skills. This meant that the team had to organise where the buggies went to; originally one was sent to China and one was kept in the UK (where it moved between three members throughout the second semester). These were moved due to several changes occurring: people travelling or the construction of the track being more feasible in a different household. This displayed the well organised nature of the team.

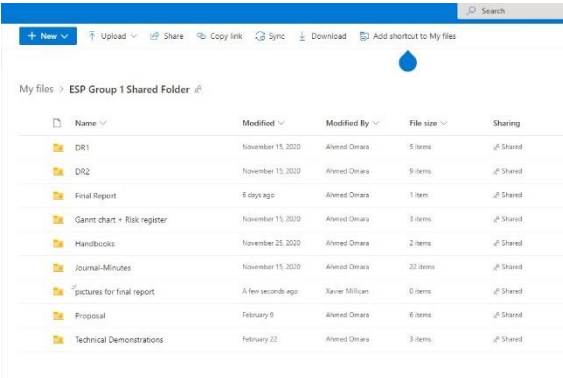
Initially upon beginning a task, all members start by reading the mark scheme and handbook in preparation and highlighting the date of the deadline, so as not to miss any key points or miss the deadline. The group had meetings twice or more each week and anything that had to be discussed was spoken about on the WhatsApp group chat. If the objectives changed during the project, any alterations to plans were discussed quickly and effectively, to avoid any time being wasted and ensure the workload was shared efficiently. This sometimes involved splitting a task between more people, in order to get something completed faster or improve someone's knowledge and experience with something. These objectives are all inter-related, and acted like a chain, so if one objective changed, it would affect other objective and thus the amount of work would increase for the group. Contact with the ESP tutor was maintained at least once a week to ensure everyone was on track and to clarify any underlying problems that may stand.

To limit conflict and for work to be efficient, it was devised to meet regularly throughout the course of the project, clearly discussing any opinions on a variety of potential difficulties. A minimum of one meeting per week was undertaken, even if it were short, whereby we would attempt to resolve issues. Some examples of these problems were: delayed work from members of the group, having to push back an internal deadline, workload problems, miscommunications within the group and technical difficulties.

These problems were ultimately resolved through the WhatsApp group chat or in a Zoom group call. Issues regarding delayed work caused problems amongst the group, such as certain sections having to be completed prior to other parts being able to start. In addition, if a deadline was fast approaching, it was likely to increase the stress of the group. Most of the time, individuals were urged to finish their part of the work, or work was re-delegated with clear communication. It was important that the group always had an internal deadline (approximately one week prior to the actual deadline), so there was always time to make corrections and to allow for any potential unforeseen delays within the group.

An additional problem was that the buggy never arrived in China, limiting the ability of the group to test the buggy more thoroughly, since the group only had one buggy to upload code and test on. This was detrimental to the group; however, it was coped with by ensuring all new code created was sent to the buggy holder and tested as efficiently as possible.

To try and prevent miscommunications between members, an organised shared folder was prepared, where all our work was saved [3]. Although all members speak English, it is not everyone's first language. This meant that within calls, members must make an effort to speak clearly and if something was not understood, it was repeated until necessary to ensure all members were included. As time has progressed, the group have become more comfortable with this.



Name	Modified	Modified By	File size	Sharing
DR1	November 15, 2020	Alfred Omeara	5 items	Shared
DR2	November 15, 2020	Alfred Omeara	5 items	Shared
Final Report	6 days ago	Alfred Omeara	1 item	Shared
Gantt chart + Risk register	November 15, 2020	Alfred Omeara	3 items	Shared
Handbooks	November 25, 2020	Alfred Omeara	2 items	Shared
Journal-Minutes	November 15, 2020	Alfred Omeara	22 items	Shared
Pictures for final report	A few seconds ago	Xavier Millican	0 items	Shared
Proposal	February 9	Alfred Omeara	6 items	Shared
Technical Demonstrations	February 22	Alfred Omeara	3 items	Shared

Figure 8 The group shared folder containing all of the relevant files for the group

If the project were to be done again, a few things could improve, including: clearer role designations, in which assessment of each other's strengths and weaknesses can be made and the assignment of members to certain types of work can be made with more distinction. When a report had to be split amongst the group, it was commonly split through random assignment, which worked with limited complaints, however, a proper understanding of the group members could have made this division more suitable. On top of this, to limit stress within the group, more importance could have been placed on the internal deadlines, so most of the work would be completed by a date set by the group. This would allow more time to finalise and improve work.

A Gantt chart was created at the start of the second semester with all the deadlines [3]. However, due to the number of deadlines and increased workload during the second semester, it became difficult to update the Gantt chart. Furthermore, as stated previously, the group held regular meetings, usually twice a week during the lab sessions. This allowed the group to ask for help if there was anything that wasn't understood. The number of meetings held each week changed throughout semester two depending on how much work had to be done.

The proposal document was split into tasks and distributed evenly between the group members, which allowed each individual group member to have less work to do. Furthermore, an internal deadline was set a few days before the actual deadline to allow the group to fix and improve the report.

For TD1, the group members decided to work on the tasks together during the meetings, which resulted in the group having 4-5 meetings a week before the deadline of TD1. On the other hand, for TD2 the group members decided to split the tasks and meet up just twice a week, which allowed for better productivity and less time wasted.

Starting from TD3, the group started facing challenges when it came to completing the tasks. TD3 required code to be written on Arduino to use on puzzlebot. Earlier in semester two, the group members decided to send one puzzle bot to China, and to keep the other one in Manchester. However, the one in China never arrived, and the one in Manchester had problems. The problems with the puzzlebot took over a week to fix, and the PCB had to be replaced. This caused the group to have only a few days to write and test the Arduino code and consequently missed the testing session, which resulted in added stress and pressure. This resulted in the group's TD3 mark being lower than those obtained for TD1 and TD2. For TD4, the group still only had

one puzzlebot to test the code with, which made it harder for the other group members to help the member with the puzzlebot to write and test the code. This resulted in only one group member being able to properly work on TD4, which led to the group not being able to complete all the tasks for TD4.

## **5. Heats Analysis**

In preparation for the heats the software was first simulated on MATLAB. This included the controllers, wall following, line following, robustness tests, etc. After that, the buggy was constructed and the libraries for each component in the buggy were constructed in Arduino. Each library was tested individually to ensure that it worked before integrating it with the main line code. For the sonar the library, the operation of the PING sensor was looked up and the triggering and measuring mechanism was looked up as per the datasheet and other useful sources online [4]. For the servo, the datasheet was used to understand how to get the setpoint for the angle [5]. Finally, for the encoders, the operation of a rotary encoder and then a quadrature encoder were researched [7]. The values for CPR and the maximum speed, etc. were obtained from the datasheet [7].

After all the research was done, these libraries were constructed as shown in section 3. After that, the code was ported from MATLAB to Arduino. From there, the code was tested. Of course, the controllers needed to be updated and the values for the reflectance sensors needed calibrating, as there are external factors such as noise, light intensity, track reflectivity, etc. that the simulation could not account for. This meant that the controllers needed re-tuning and special care was needed for the differential controllers, as noise was a cause of issue, as the noise was amplified as well, interfering with the controllers. This meant that an extended amount of time was needed to test the buggy and fix the control so that it is robust.

A track was made at home such as to test the buggy, however, the surface that the buggy was being tested on was a glossy surface that is different from the matte surface of the racetrack, and hence calibration of the values between tracks was necessary. Through testing it was evident that sunlight was a source of error as a wide enough dead zone for the controller can cause unpredictable behaviour [6]. As such the dead zone was narrowed as much as possible. However, while narrowing is needed, caution needed to be taken as different tracks have different bounds. This is where the average method (equation (7)) excels as it automatically accounts for the differences and calibrates the values automatically giving the appropriate error. During the heats however, it was noticed that the buggy was very noisy. This meant that the noise was very high and that the corrections of each wheel were very high unnecessarily. This ultimately means that the derivative control needs tuning or even removing. Also, the integral gain was a source of error as this causes the values to spike towards the setpoint which ultimately means that in excess, the values could spike out of control. On the other hand, the proportional controller seemed to work, however the values for the constants were very small.

After the final testing slot, a few modifications to the software were made. An average value of the reflectance sensors was used instead of calibrating the value of each individual sensor, this was done because the buggy did not behave as expected in following the line due to the change of environment and hence the amount of light received by each sensor. The set speed for both wheels have been increased as well to assure a smooth ramp climbing. The buggy did perform good in the heats test, it successfully followed the line, climbed up and down the ramp, and stopped at the end of the track when no line was detected. The sonar did not work as expected; the buggy hit the wall without performing a 180° turn, this was due the incorrect configuration of the sonar pin. Another thing that could be improved is the sturdiness of the PID controllers to make the buggy follow the line smoothly without oscillating much around the white line.

As an additional feature mentioned in the proposal report, the sonar was going to be used for the original buggy [3]. However, the addition of the servo motor made a big difference as this allowed control over where the sonar was pointing at all times. This also helped with detecting the walls in case the buggy veered near the walls. As an additional feature, the buggy could follow a wall and this was tested in MATLAB simulations, however this feature wasn't used in the final heats assessment. The sonar sensor made it easier to signal the buggy at the point at which to turn 180 in the heats assessment.

Another successful feature was the large number of reflectance sensors. There were 13 line sensors equipped on the puzzlebot [1]. This allows the buggy to read a greater number of values (hence increasing the range that the error can go up to), allowing the group to code the puzzlebot in a way that makes it more accurate. Another feature is the spacing of the reflectance such that less sensors can be on top of the line at the same time. In the final configuration, only the odd sensors were used, such that only 7 sensors were used; this increased the spacing even further. Another successful feature was the size of the sensor, such that it was smaller than the originally planned TCRT5000 (figure 9) sensors; this reduced the probability of light interfering with the sensors. The down side of this is that little amounts of light are available, then the sensor may overlook them if it is moving over it quickly. Another feature is the shielding of the sensors, blocking as much sunlight as possible with a physical barrier.



Figure 9 The TCRT5000 sensor used in the initial plan for the project

The encoders were another success, as they provided a very accurate reading of the speed of the wheels as well as the distance covered by the wheels. The encoders were placed at the input shaft, which made the readings out of them accurate as the reding came straight from the source, making the readings more accurate than if the encoder was placed on output shaft. Another success was that the encoder was a quadrature encoder and not a rotary encoder. This allowed for 2 separate measurements that are 90° out of phase such that the direction in which the motor can be determined.

The wheels were both a good and bad feature. The wheels being made of plastic instead of rubber resulted in less traction to the track and as such made it so that the buggy could slip side to side more easily. This also made it quite harder to ascend the ramp as the buggy can just slip back down and not ascend if not enough torque is supplied. The huge size of the wheels means however that the torque is higher and as such the motor wouldn't need to supply a high torque. This overcomes the issue introduced by the plastic wheels on the ramp.

One of the worst features is the shape and size of the puzzlebot. The base of the buggy is wide, which made the puzzlebot somewhat clunky and slow. Another point is that the buggy isn't aerodynamic as the front of the buggy has a flat surface. This made it so that the buggy would have to overcome air resistance to some extent. The corners were round off, to reduce the hazards introduced to the buggy. The most advantageous feature of the buggy is that it contains a double layered chassis. This allowed for all the components to be distributed along a smaller length. This also ensured that the centre of mass of the buggy was near the center and that the buggy was more compact.

## 6. References

- [1] A. Stancu. Puzzlebot User Manual. (2021 Autumn). EEEN21000 Embedded Systems Project. University of Manchester. United Kingdom
- [2] Pololu Corporation. "QTR-MD-13RC Reflectance Sensor Array: 13-Channel, 8mm Pitch, RC Output." Pololu.com. <https://www.pololu.com/product/4153> (accessed May. 12, 2021).
- [3] A. Omara, M. Ali, I. Albrashdi, Z. Zhao, X. Millican, J. Wang. "Proposal Report". University of Manchester. Manchester. UK. 2020
- [4] Parallax. "Communication Protocol," PING)))™ Ultrasonic Distance Sensor (#28015) datasheet Nov. 9, 2009 [Last revision Nov. 9, 2009]
- [5] Pololu Corporation. "FEETECH FS90 Micro Servo" Pololu.com. <https://www.pololu.com/product/2818> (accessed May. 12, 2021).
- [6] L. Marsh. Technical Handbook. (2019 Winter). EEEN21000 Embedded Systems Project. University of Manchester. United Kingdom
- [7] Pololu Corporation. "34:1 Metal Gearmotor 25Dx67L mm LP 6V with 48 CPR Encoder." Pololu.com. <https://www.pololu.com/product/4824> (accessed May. 12, 2021).
- [8] R. Smoot *What's the Difference Between an Incremental Encoder's PPR, CPR, and LPR?* Accessed on: 06 May 2021. [Online] Available: <https://www.cuidevices.com/blog/what-is-encoder-ppr-cpr-and-lpr#:~:text=CPR%20most%20commonly%20stands%20for,two%20outputs%20A%20and%20B.&text=Therefore%2C%20the%20CPR%20of%20an,encoder's%20PPR%20multiplied%20by%204>