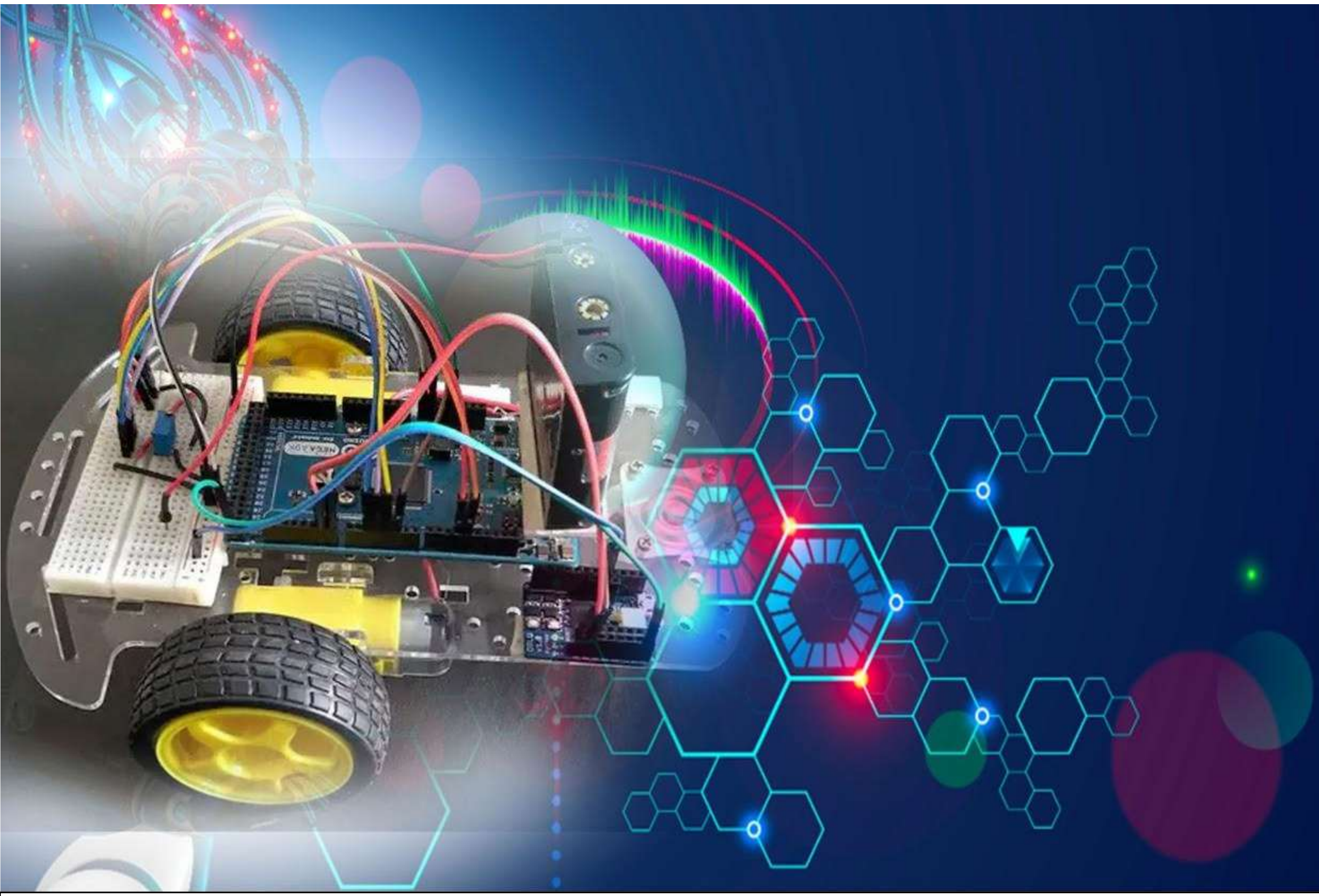


LINE FOLLOWER ROBOT- DESIGN AND BUILD PROJECT

GROUP 1

Queen Mary University of London

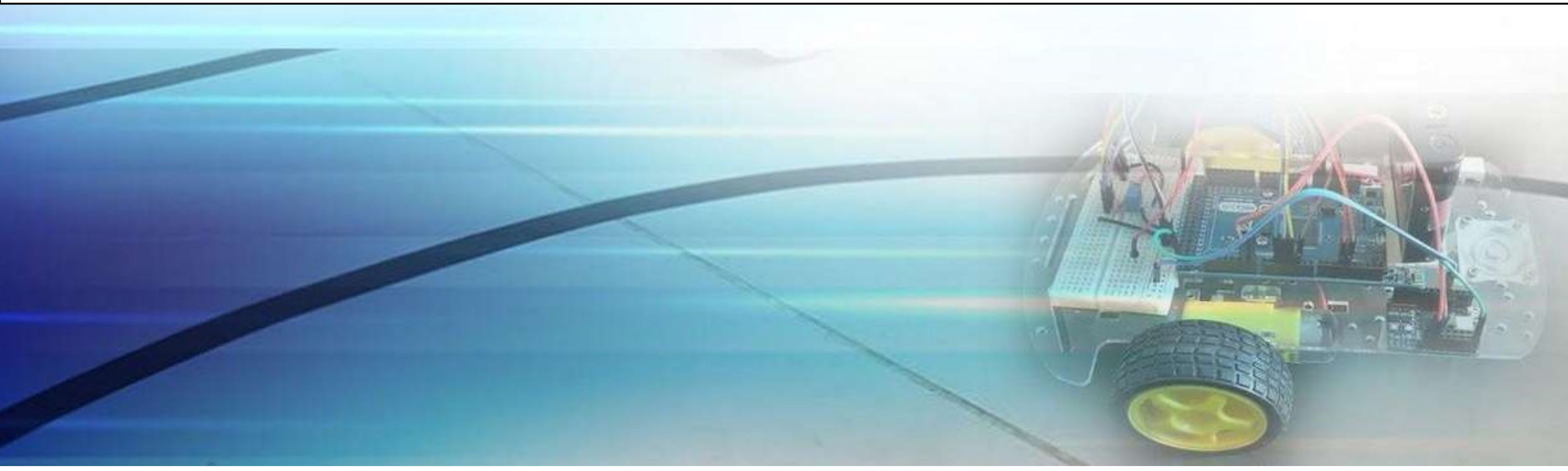
Group Leader : Nader Akmel, ID:180438002, ec18248@qmul.ac.uk

Robel Mebrahtom, ID:180336845, ec18175@qmul.ac.uk

Muhammad Hamza, ID:180354816, Muhammad.hamza@se18.qmul.ac.uk

Abubakar Abubakar, ID :180295058, ec18139@qmul.ac.uk

Phoenix Smart, ID 180481019, ex18255@qmul.ac.uk



Abstract

The aim of this project was to apply the knowledge and skills that we acquired during the first and second academic year in the course of Electrical and Electronics Engineering and design a robot car that follows a black line on a lighter surface.
The main features of this device include:

- The car reads the data of its surroundings and follows a black line using information given by the infra-Red sensor
- To be able to make sharp turns by controlling the motors speed(which are controlled by a microcontroller) and direction.
- Ability to stop if there are any obstacles on the track using the data read by ultrasonic sensors.

This device detects a black line on a white surface, it estimates whether the line under it is moving towards the left or right as it moves over it. Based on that estimation signals are sent to the motors to turn left or right so that they can maintain a steady centre in accordance with the line. An appropriate communication between the software and the hardware enables the robot car to achieve its objectives successfully. The complete procedure of how this robot car works is explained in detail further in the report.

In order to achieve our objectives we used an Arduino UNO, Robot Chassis, 2 wheels along with the 2 stepper motors to make the car, and for sensing we incorporate Infra-Red sensors which accurately detects the reflectance of the surface so that it can follow the line whereas to detect any obstacles in the path ultrasonic sensors are used. The Arduino is coded such that it signals the motors to vary its speed and perform sharp turns hence following the line and stopping when an obstacle is detected Infront of it.

Table of Contents

1. Introduction

Due to immense advancement in technology in this century, companies like Mercedes, tesla, google, etc. have started to implement the idea of self-driving cars. The reason behind this idea is that a high rate of automobile accidents occurs due to driver error so to eliminate this problem the idea of self-driving cars evolved. These cars have the ability to reduce these accidents and save many people's lives as these cars are programmed very specifically, keeping all safety measures in consideration along with the assistance of a software and feedback from the sensors they operate, hence providing a smooth and fear free drive to the passengers.

The robot uses Infra-Red sensors in order to calculate the reflectance of the surface. The way it works is that the black line will have a lesser reflectance value (black absorbs light) than the lighter surface around it. This low value of reflectance is the parameter used to detect the position of the line by the robot. Similarly, the robot uses ultrasonic sensors to detect obstacles and avoid them in the trajectory. All this information is fed back to the Arduino uno(microcontroller) which is coded which then varies the speed of the motors, accordingly, hence enabling the robot to perform the trajectory effectively.

This report will summarise in detail all the research, planning and development procedures commenced by all group members in creating this line following robot. We will discuss the design choices for hardware components along with the software components available and further elaborate what we chose for our design. Moreover, methodology and testing of the design components will also be discussed. To conclude the report, results will be discussed along with the possible improvements which will resolve the problems faced during the manufacturing of the robot.

2. Specification of Requirements

2.1 Problem Statement & Project Aims

The problem to be solved is to create a robot which follows a specified path by the user, in this case a black line over a white background or colored line over white background. Additionally, the robot detects objects in its path and successfully avoids them. The robot is required to function automatically by gaining information of its surroundings through sensors, and without any human intervention. The sensors will sense the changes on the path of the robot and will transmit a signal to the microprocessor, which will change the speeds and direction of the motors connected to the wheels.

2.2 User Requirements

Our requirement is to build a robotic car that detects a black line and follow it. Therefore, the robotic car must be able to detect a black line which is over a white surface. By detecting the black line, it should be able to turn its direction in accordance with the black line regardless of the line's complexity. The car must be able to sense an obstacle which is on its path and it should stop. The sensors should not be sensitive to external change in its path such as light contrast.

2.3 Constraints

- We have decided as a group to constrain our project to only let the robotic car follow a black line over a white background. This is due to the infrared sensors finding it difficult to detect a line with small contrast between the line and the background.
- If black line is not detected the robot will never stop. As there is no stopping mechanism for just a white background, 3 sensors could be used instead of 2 however this will pose a functionality constraint as the middle sensor would be required to stay on the line.
- The robot will not avoid the obstacle instead it will just stop when the obstacle is detected, a mechanism for going around the obstacle can be implemented however it would not be completed within a realistic timeframe and budget.
- To increase the speed of the wheels means to have high output voltage, which is difficult when using an arduino(due to a maximum voltage being 5v).This means the project will have to increase in complexity, which due to time is not the best solution.

2.4 Input requirements

- The inputs must be read from sensors, in this project we are reading two input signals:
    - Infrared signals (used to detect the black line).
    - Ultrasonic signals (used to detect any obstacles).
- Infrared sensors must have implemented an ADC to ensure any signals detected are transmitted to the microcontroller in a constant digital format.
- Ultrasonic sensors must detect sound signals reflected from obstacles in the robot's path and convert the result into a digital signal which is sent to the microcontroller through its echo pin.
- Arduino controllers must take in digital inputs as the microcontroller IC is a digital device.

2.5 Output requirements

- Output in this project must be actuators (wheels and motors).
- Microcontrollers must output signals in analogue format to ensure robotic cars slow down in a non discrete manner when path changes.
- The servo motor output will be determined in the arduino code from the amount of rotation required for the ultrasound sensor.

2.6 Operational Requirements

- Stopping operation - The Robot must stop moving when an obstacle is put in front of it.
- Moving operation - The Robot must move and follow wherever there is a black line.
- On/Off operation - Must stop and start when the switch is turned on/off.
- Turning Operation - Must turn when the line path changes direction.

2.7 Functional Requirements

- Stopping function - the robot uses an ultrasonic sensor to detect obstacles, by receiving an analogue reflected signal from one of the transducers, and sending a digital high to the microcontroller which will then stop the moving operation.
- moving function- takes in inputs from the sensor by detecting the line that tells the processor/ microcontroller to keep moving when a black line is not detected by the sensor.
- On/Off function - User presses switch to turn robot on/off by creating an open circuit between the battery and the microcontroller.
- Turning function - Input is taken when a black line is detected by one of the sensors and causes the robot to turn in that direction, by altering the speed of one of the motors.
- Optimising speed - takes input from sensor and senses when a path bends, which means that robot must slow down in order make sure it follow

3.Methodology and design

3.1 Design approach

In this section, we will be discussing what our main design approach for all hardware and software components that were implemented in this project. This project was split into two phases:

- Phase I:
  - Design and Implementation of a motorised and automated vehicle/ robot that will be able to follow a line.
- Phase II:
  - Avoiding obstacles and detecting traffic and danger.

Along with the requirements of both phases, we were also informed in the project briefing of a possible change in specification requirements. This highlighted to us the one the main design approach we needed to take was one that would ensure that our robotic vehicle would be built and designed robustly to the addition of phase II's obstacle avoidance and any changes in the specification.

When designing the main design approach, we needed one that ensured that all specification requirements were met, especially those of the design. Some of these being:

- The Robot must stop moving when an obstacle is put in front of it.
- The Robot must move and follow wherever there is a black line.
- Microcontrollers must output signals in analogue format to ensure robotic cars slow down in a non-discrete manner when the path changes.
- Takes in inputs from the sensor by detecting the line that tells the processor/ microcontroller to keep moving when a black line is not detected by the sensor.
- Be within budget – components chosen economic value in mind ≤ £100 in total.

3.2 Microcontroller

We first began by deciding what microcontroller we would be using, being that all other design aspects would be centred around this choice we needed to take into account many aspects. Some of them being:

- Price
- power consumption
- processing speed
- code implementations
- compatibility with the other components.

We researched all these aspects and decided to pick the Arduino Uno R3 as the microcontroller.

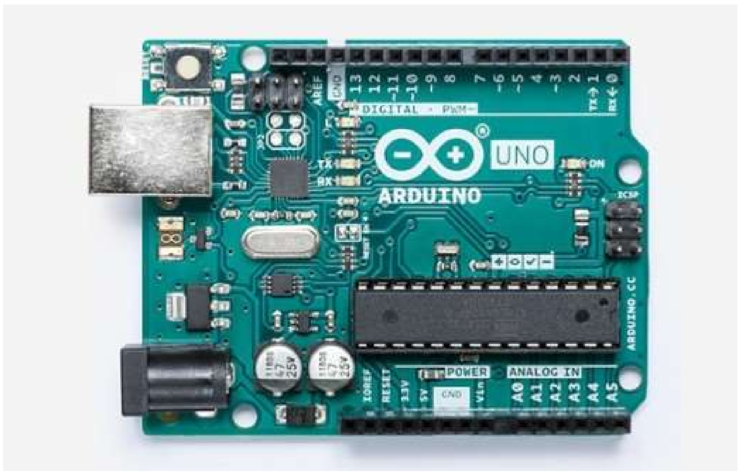
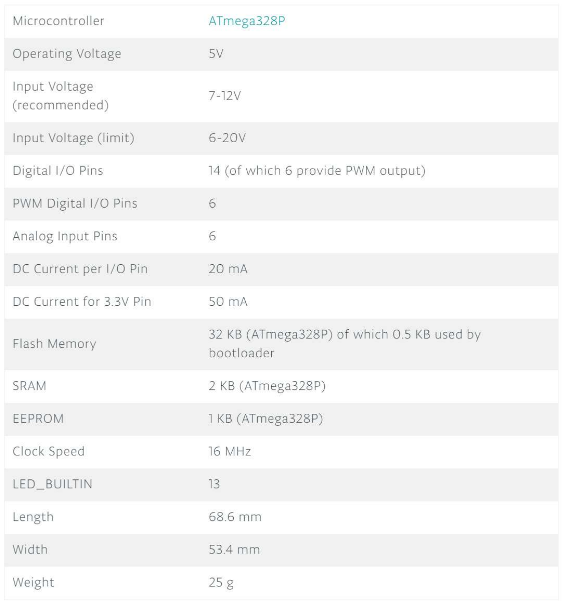


Figure 3a: Arduino Uno Rev3.

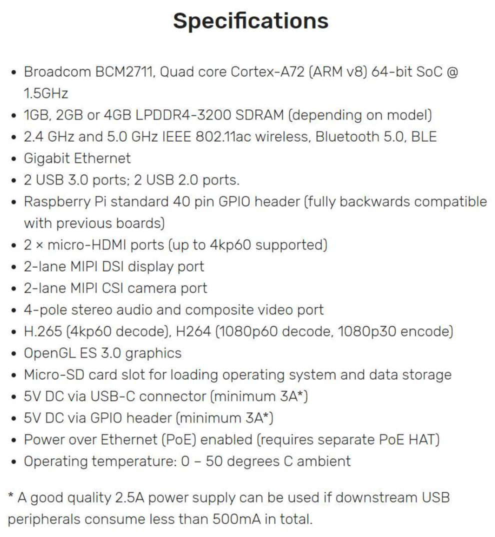
The Arduino Uno is a microcontroller that is part of the computer and is a microcontroller board that is based on ATmega328 chip. The board can also be powered using two ways, one way is using a USB cable another way is connecting the board to an external battery source with a recommended range of 7 to 12 volts however can go up to 20 volts.



Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Figure 3b: The technical spec for Arduino Uno Rev3.

Another competing microcontroller is the raspberry pi is a minicomputer that comes with a fully functioning operating system called the Raspberry Pi. Its specification is shown below in figure 3c:



Specifications

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 1GB, 2GB or 4GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

Figure 3c: Specification for the Raspberry Pi 4

Firstly, let's consider the price, according to the Arduino's official website the Arduino Uno Rev 3 cost 20 euros [1] which is around £17.38 because our budget is up to £100 this could be considered very cheap since the microcontroller would be the most expensive component in our design. Comparing this to the price of the raspberry pi 4 model B which is $35 [2] this would cost £27.97. Therefore, taking in the factor cost the Arduino is cheaper which allows a bigger budget on other quality components and/or bigger quantities for other components.

A Raspberry Pi microcontroller can run multiple programs at the same time whereas an Arduino can only run one program consecutively. For this reason, to function the raspberry Pi needs 5 volts and a 3A supply (15 W) [3] with a minimum current of at least 2.5A. The Arduino also requires an operating voltage of 5 volts [1] however draws a maximum current of about 20mA per I/O pin (5 W)[1]. This means that Raspberry Pi has a higher power consumption compared to the Arduino, therefore, highlighting that if we use this raspberry pi microcontroller our battery will run out much faster than the Arduino.

Figure 3d: Raspberry Pi 4.

A key advantage of the Raspberry Pi is that it has higher processing power, up to four times the processing speed of the Arduino. The Pi also has 128000 times [4] more RAM than the Arduino. This is important especially because at the end of this project groups were expected to race for who can finish the track the fastest, this means that the microcontroller would be able to process instruction and conduct decision faster which means those that have higher processing speeds will ultimately be able to finish the track the quickest.

The Raspberry Pi can also be coded using python (this is the recommended language), C and C++. Whereas the Arduino only uses C/C++ languages [4]. However, since we have taken a previous module on C programming, we believed that we would be finding the code simpler and so, therefore, the Arduino was enough for us to use.

When considering compatibility with other components both the Arduino and Raspberry Pi have many components that are sold specifically to be compatible with their microcontroller. So, therefore, they have many libraries that make easier to for testing and coding and so as a group we had to conclude, which one these microcontrollers would we be able to code easily using these libraries and being that we tested both the Raspberry Pi and the Arduino we found that Arduino was easier to use. Simply because it was more compatible with windows whereas raspberry Pi needs Scratch, IDLE or anything that can support Linux to develop the code.

Taking all these variables into consideration it was clear that the Arduino software was the best fit for this project. The main task of this project is for a robotic car to complete a black and white track with a complex formation and after doing extensive research on the microcontroller it was clear that and Arduino is best for doing a repetitive task and being that algorithm we use for sensing the black line needed to be repeated (this will be discussed later) throughout the task, it was clear that the Arduino was our best choice.

3.3 Line sensor

One of the essential components of the project was the sensors, sensors are used to detect the black line the robotic car would be following. When deciding what sensors to choose we needed to consider our input requirements since the sensors would be used as an input component in our robot and some other variables. These being

- The inputs must be read from sensors, in this project we are reading two input signals:
  - Sensors used to detect the line.
  - Sensors used to detect any obstacles.

- Sensors must have implemented an ADC to ensure any signals detected are transmitted to the microcontroller in a constant digital format.

- Arduino microcontrollers must take in digital inputs as the microcontroller IC is a digital device.

- Response time – The sensor needs to be able to respond to changes to the environment quickly.

- Sensitivity – Must be sensitive enough to detect the black line, but not too sensitive that it can be affected by a change in the environment.

In our project, we used two IR sensors to detect the line that the robot needs to follow. The IR sensor is made of two components: one infrared light emitter and the other is photodiode which receives the reflected signal from the surface. The robot determines the black line due to the intensity of the signal reflected from the ground.

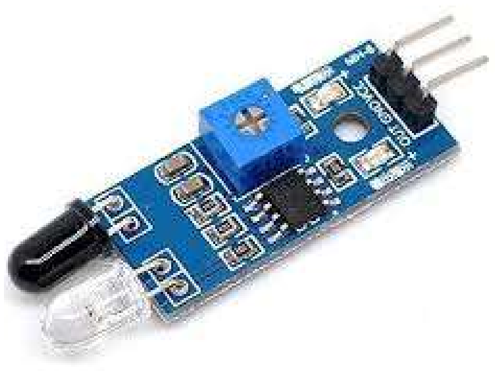
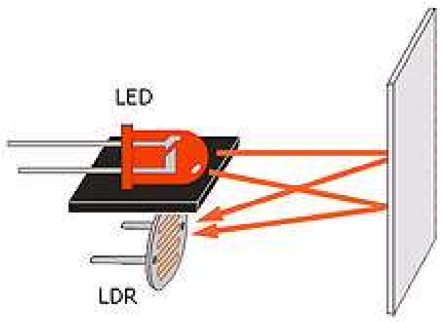


Figure 3e: Infrared Sensor Module.

If the infrared light falls on a black surface, the light is absorbed by the black surface and there will not be light rays reflected. However, if the IR light falls on the white surface, the light will be reflected as shiny surfaces are good reflectors, and the photodiode will receive the light signal. The output of the sensors is an analogue signal which depends on the amount of light received in the sensor and this analogue signal is transferred to the comparator to convert it to a digital signal which is 0 and 1s.

An alternative sensor for detecting the line is a combination of LDR and LED, LDR is a light dependent resistor and varies in resistance depending on how much light it detects. It ranges from 1M ohms (or more) when there is reduced light tp 80 ohms in brighter light. The LDR is connected to a network of resistors that make up Individual voltage dividers. When both the LED/LDR pairs go over a white surface as large amounts of light get reflected in the LDR, therefore, reducing the resistance in the circuit and ultimately making the voltage across the LDR drop. When the LED/LDR pair detect a black line the intensity of light reflected will be low and so which means resistance and voltage across the LDR will be high. A disadvantage of using an LDR sensor is that its response time can be very high compared to the infrared sensor, it also has a low range and can be sensitive to a sudden change of sunlight.

The infrared sensors can also be sensitive to sunlight; however, we were able to find a way around this by attached a black electrical tape around the sensors so that it reduces the sensitivity of the surroundings. Taking these variables into consideration it was clear to see that the infrared was suitable to detect our black line.



3.4 Motor driver

An Arduino board is not capable of providing direct current to the motors therefore, a motor driver is needed to act as an amplifier and controls the flow of current that travels through the motors. The motor driver we used in our project was the L298 full H-bridge. This motor driver can drive two motors simultaneously. The motors can also be rotated in the reverse and forward direction which is important for our method of turning left and right.

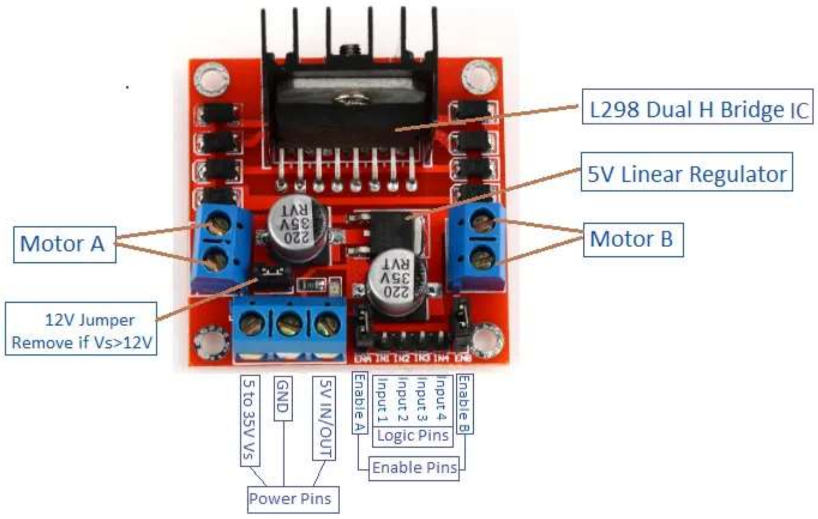


Figure 3g: L298 full H-bridge

In our initial research, we were able to find that there were two main types of motor drivers that would be best suitable. One of them being the L298 full H-bridge and the other being the L293D motor driver. One of the main requirements that we set for ourselves was that the motor driver must provide sufficient current to the motors to ensure motors are working at their optimum. We needed to also take into consideration the price of the motor driver.

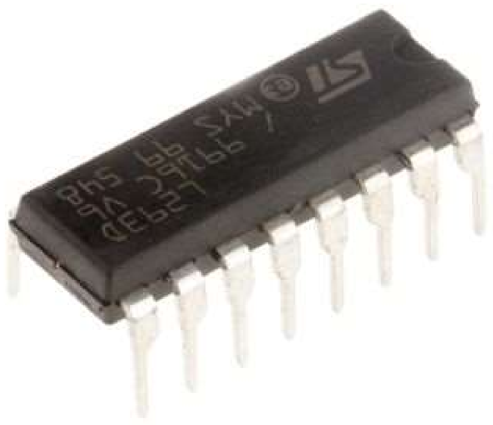


Figure 3h: L293D motor driver IC.

Both the L298n AND L293d can control two motors simultaneously, however, the L293d can only output current of 600mA whereas the L298n can output a current of 1A, which therefore increases motor speed. One disadvantage of using the L298n is that it costs more than L293d. The L298n motor driver however comes with an in-built heat sink to protect the motor from overheating, which indicates we are getting more for our money.

3.4.1 Motor

The motors in this project are the driving force for the robot. When choosing a suitable motor, we needed to consider what kind of approach we would take. Whether we wanted our robot to be fast with less torque or slow with high torque was a very important discussion. For our project, we decided to have robots that would be slow but with high torque, because it was important for us that the robot was slow enough for the sensors to detect turns and have enough horsepower to make those sharp turns. Another requirement we needed to consider was efficiency, as we needed to make sure no power was lost through motors since the microcontroller and sensors require high power consumption. Some of the design choices we considered included the stepper motor, servo motor and DC geared motors.

An advantage of the stepper motor is that it provides precise positioning, by accurately moving between their many poles without the help of the position encoder[5]. They also provide low torque at low speeds and are much easier to control. The disadvantage of the stepper motor is that they are not very efficient, they constantly draw maximum current which results in the motors overheating.

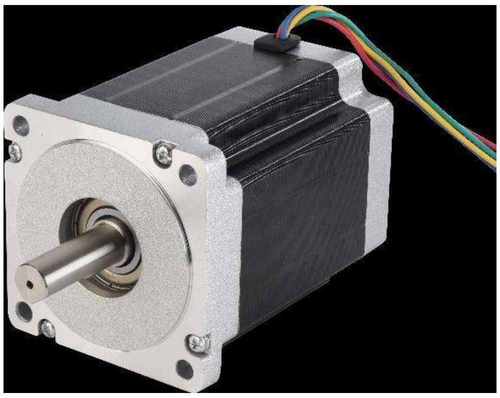


Figure 3i: Hybrid Stepper motor.

The servo motor can generally generate excellent peak torque at high speed, this is because they have a constant close- feedback mechanism. However, after some research, we found that the servo motors work better for more precise and detailed movements such as robotic arm, as it is measured mostly in degrees rather than rpm. They are also limited to 180 degrees motion [5] and have a history of causing bots to jitter due to it trying to correct itself from drifting.

Figure 3j: Mini servo motor.

The geared DC motors can be controlled by simply applying a voltage, low voltage slows the motor down and reversing the voltage reverses the direction to the motor. They also have brilliant torque at low speeds, brushed DC motors are about 75-80% more efficient [5]. One disadvantage is that it may produce high noise but being that our robot is not required to be quiet or have a high speed when traveling this would not be a problem . This would, however, affect the race competition at the end of the competition.

Figure 3k: DC geared motor.
After comparing all the suitable design options, we decided the DC geared motor was most suitable since it met most of the requirements that we felt were most important to us.

3.5 Power supply

A vital part of the project is the power supply as it provides, the power to each of the individual components in the robot.

The total amount of power required by the robot to have optimal functionality as well as the voltage requirement for the components is an essential factor to consider. To determine the voltage and capacity of the battery needed we used the current and voltage rating of each of the components on their datasheets. The cost of a battery is also another requirement, as batteries can run out and may need to be replaced throughout testing.

Considering these factors, one option is to use lithium-ion batteries as they provide high current and have a relatively low self-discharge. However, a disadvantage is these batteries can be very expensive and can be subject to ageing even when not in use.

Figure 3I: polymer lithium-ion battery.

In our project we felt that it was sufficient to use normal AA alkaline batteries to power our robot as they were inexpensive compared to alternative options and could power all components. To understand how much voltage we needed to supply, firstly we determine the total current draw which is calculated by totaling up the current rating of the individual components resulting in the current that battery needed to provide at the maximum load while factoring in the highest voltage rating of the component using the collective datasheets of the components. We found the motors were the component with the highest voltage rating using datasheets of the components. We use this value to suggest the voltage needed to be supplied from the battery to the components.

Figure 3m: Alkaline AA batteries.

The collective current and voltage of our components:

Motors: 6v @0.15A MAX, 3v @0.20A MIN (x2)

IR Sensors: 5v @100mA (x2)

Microcontroller: 6v @ 20mA per I/O pin

Current Supply = ∑ (required currents of all components)

When working out the current that needs to be drawn from the battery it is important, we consider the worst-case scenario, and assume that motors require the maximum current and voltage:

0.2 +0.2+0.1 +0.1+0.02+0.02+0.02+0.02+0.02+0.02= 0.72 A and maximum voltage of 6 volts.

We have now calculated our total current, we can choose a battery that would supply this amount of current and voltage. Since each battery we used is 1.5 v and has about 3000 mAh, we, therefore, need to connect 4 batteries in series which would provide 6 volts of voltage for about 4.16 hours. This meant that we had to keep changing the batteries after a couple of hours of testing, a better alternative option was to use the lithium-ion batteries since they provide much higher current.

3.6 Design & Implementation

Now that we have decided our components next, I will be explaining the working of our robot. The method we chose as a group was simple; detect the black line and the robot will move along the black line. To detect the black line, we use two IR sensors which we believed to be sufficient, as anything more would make it difficult to calibrate and cause complications in the algorithm. When the IR sensors are moving along the white surface the reflectance is high and so, the infrared light emitted by the infrared sensor will be detected by the photodiode that is placed next to the Infrared emitter. In the case of the black surface, it has very low reflectance which means that infrared light gets absorbed by the black line once it is detected and so less light reaches the photodiode.

Using this principle, we space both infrared sensors so each sensor is on either side of the black line so that when an IR sensor detects a black line, the sensors will then send this change to the Arduino Uno in digital binary format. The Arduino then sends this signal to the motor driver. To get the robot to turn right and left we use PWM to control the speed of the wheel.

This work by using high-frequency square waves and varying the duty cycle of this wave, PWM (Pulse- width modulation) have an output of high level and low level of the square wave that lasts for a period alternately. The total time is usually fixed, the time at which the output is high is called the pulse width, and the percentage of this pulse width is called the duty cycle. The longer the high time last the larger the duty cycle:

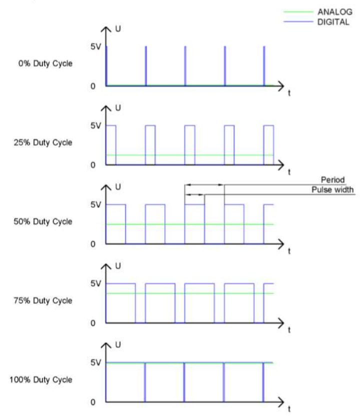


Figure 3n: PWM duty cycle output square waves.

The larger the duty cycle means the larger the output power and so, therefore, the greater the speed of the motor. In the Arduino code the duty cycle from 0-100% is mapped in the range 0-255.

To turn right the right sensor detects the black line which indicates that the black line is curving right, the right wheel is then reversed by reversing the PWM value on the pins connected to the motor driver and similarly to turn left the left sensor detects the black line which means the black line now curves to the left and the left motor is now reversed. The motor whose direction the robot is not turning in runs at normal speed. The Arduino microcontroller monitors data that comes from both sensors and moves the robot accordingly.

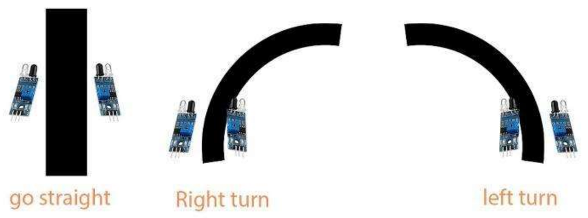
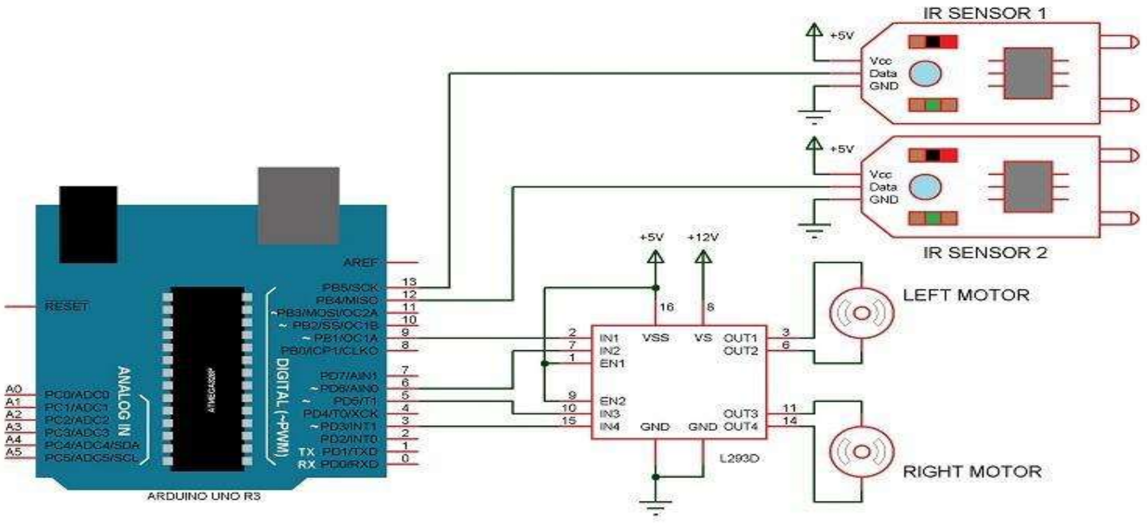


Figure 3o: schematic of IR sensor placement.

3.7 Circuit diagram

This diagram below shows all the connections between the microcontroller and the



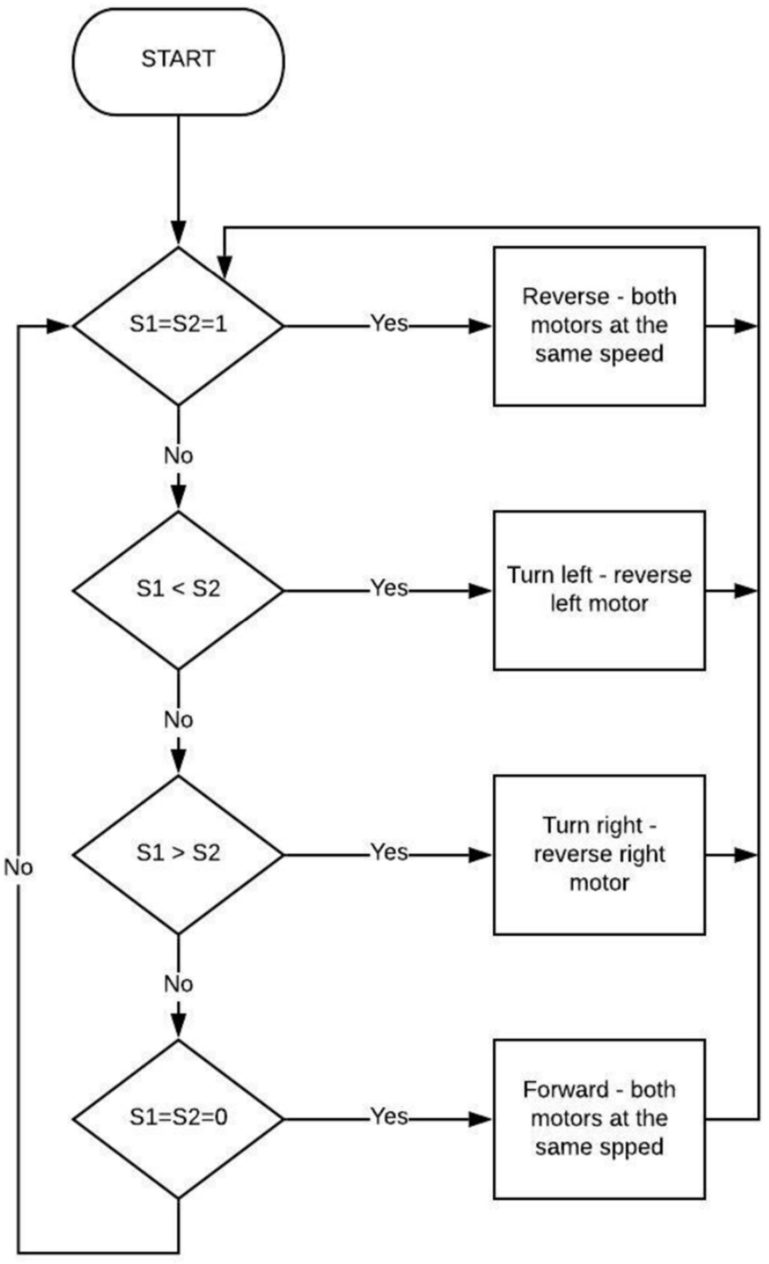
input/outputs. Figure 3p: schematic of the circuit showing all connections.

3.8) Arduino Working logic

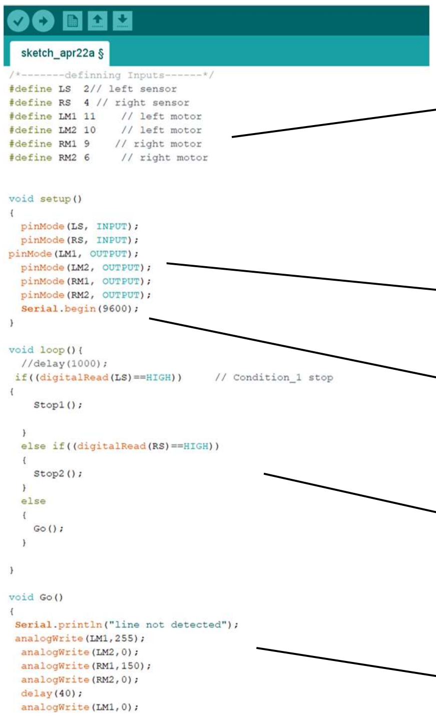
The microcontroller gets 2 inputs from the sensor circuit, below is a description of what each output is depending on the input. Where S1 is the right sensor and S2 is the left.

No	Status of sensor	S1	S2	Command	Left motor	Right motor
1	S1=S2=1	0	0	Reverse	Reverse	Reverse
2	S1 < S2	0	1	Left turn	Reverse	Normal speed
3	S1 > S2	1	0	Right turn	Normal speed	Reverse
4	S1=S2=0	1	1	Go straight	Normal speed	Normal speed

3.8.1 Flowchart


START
S1=S2=1
Yes
Reverse - both motors at the same speed
No
S1 < S2
Yes
Turn left - reverse left motor
No
S1 > S2
Yes
Turn right - reverse right motor
No
S1=S2=0
Yes
Forward - both motors at the same spped
No

3.8.2 Code Analysis



```
/*-------definning Inputs------*/
#define LS  2// left sensor
#define RS  4 // right sensor
#define LM1 11    // left motor
#define LM2 10    // left motor
#define RM1 9    // right motor
#define RM2 6     // right motor


void setup()
{
  pinMode(LS, INPUT);
  pinMode(RS, INPUT);
pinMode(LM1, OUTPUT);
  pinMode(LM2, OUTPUT);
  pinMode(RM1, OUTPUT);
  pinMode(RM2, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  //delay(1000);
 if((digitalRead(LS)==HIGH))     // Condition_1 stop
{
    Stop1();

  }
  else if((digitalRead(RS)==HIGH))
  {
    Stop2();
  }
  else
  {
    Go();
  }

}

void Go()
{
 Serial.println("line not detected");
 analogWrite(LM1,255);
  analogWrite(LM2,0);
  analogWrite(RM1,150);
  analogWrite(RM2,0);
  delay(40);
  analogWrite(LM1,0);
```

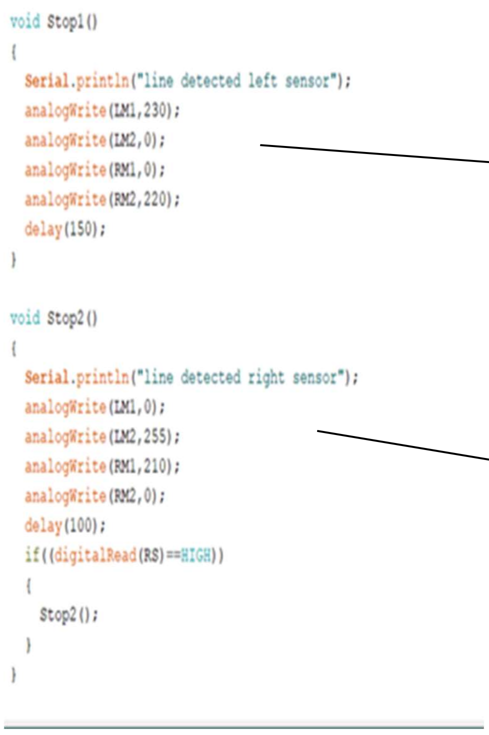
Declaring the pins on the controller that

Initializing the pins as either an input port

Setting the baud rate for the serial

If conditions that reads value from

This is the method that makes robot travel forward when only white surface is detected, “line not detected” prints to serial monitor.



```
void Stop1()
{
  Serial.println("line detected left sensor");
  analogWrite(LM1,230);
  analogWrite(LM2,0);
  analogWrite(RM1,0);
  analogWrite(RM2,220);
  delay(150);
}

void Stop2()
{
  Serial.println("line detected right sensor");
  analogWrite(LM1,0);
  analogWrite(LM2,255);
  analogWrite(RM1,210);
  analogWrite(RM2,0);
  delay(100);
  if((digitalRead(RS)==HIGH))
  {
    Stop2();
  }
}
```

The left sensor detects the black line, the left motor is reversed and the right motor rotates

The right sensor detects the black line, the right motor is reversed and the left motor

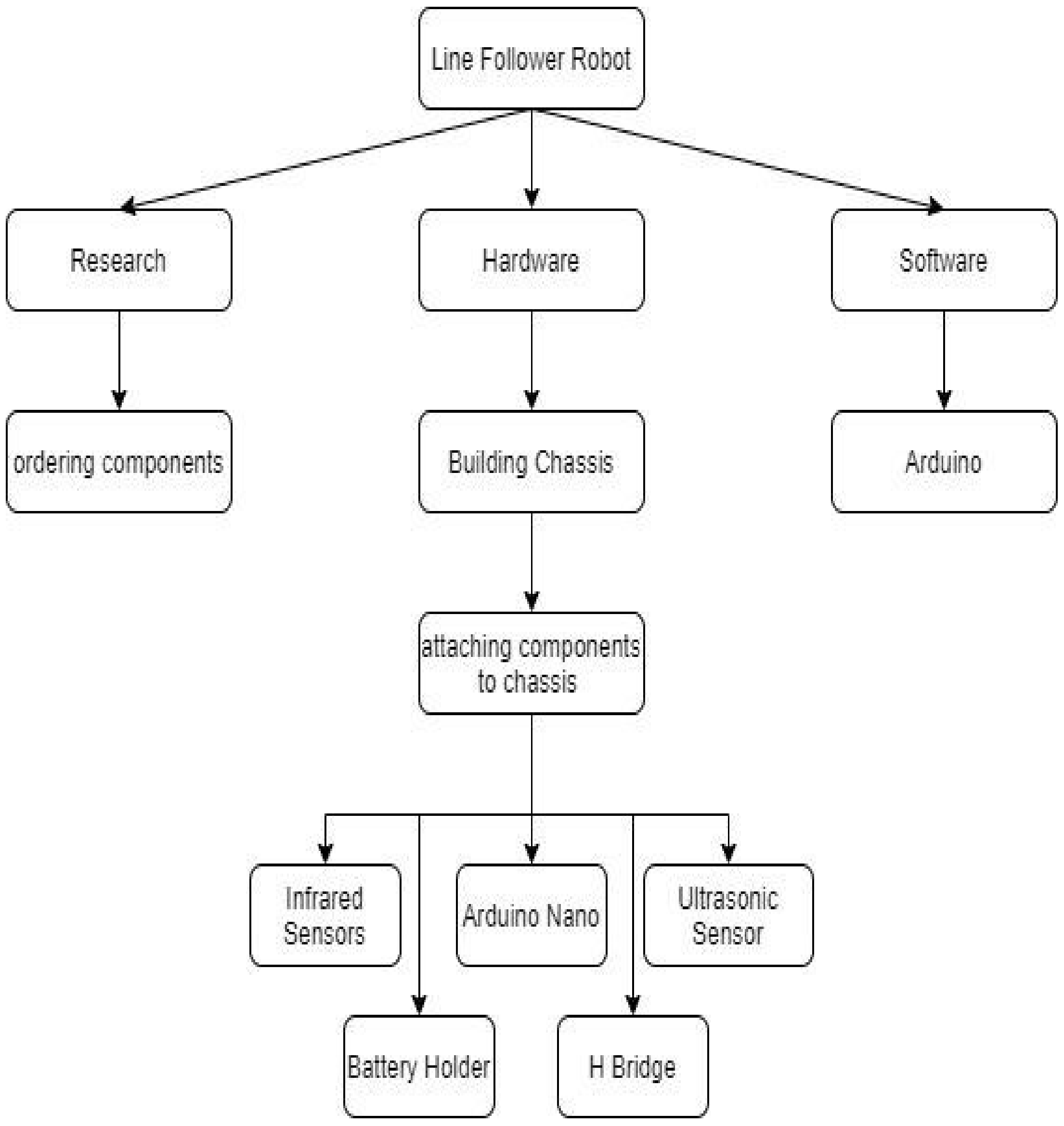
3.9 Final price list

The total cost of our project is £74.94, which is below our budget requirement of £100. Below is the breakdown cost of each component:

Component	quantity	Supplier ID	Total Cost
Arduino Uno microcontroller	1	A000066	£19.73
H- bridge motor	1	B01 BTNH S6	£4.69
170 pin mini breadboard	1	B01N0YWIR7	£5.99
GP Ultra Alkaline AA battery	12	18-2107	£3.42
Infrared Sensor Module	5	B07FCHFXMF	£8.68
2WD Chassis	1	B076BPY2L3	£10.99

4. Results and Outcomes

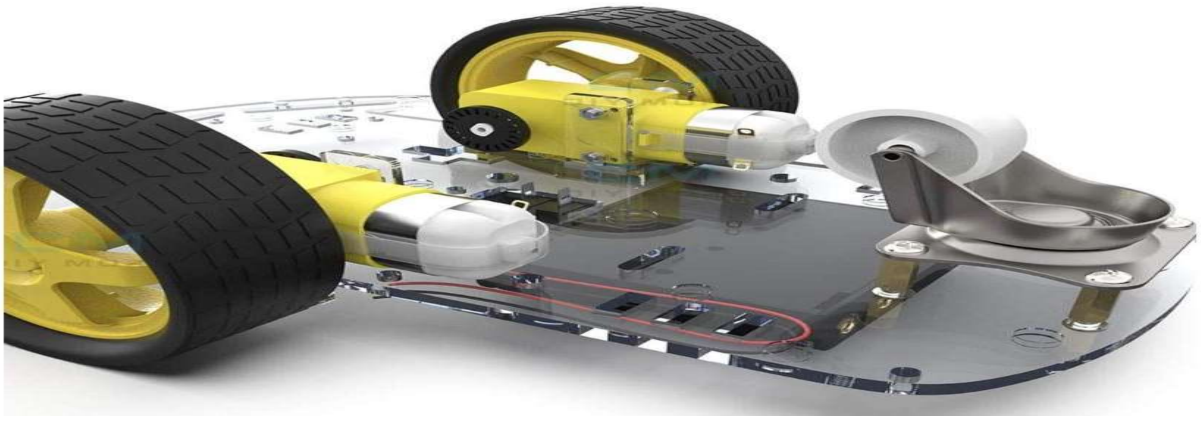
This section will explain the steps involved in the design stages of the project. In addition to the technical findings of the trials



4.1 Research

As part of the first phase before beginning any practical work, research was required since it was necessary to plan out the design stages of the project. In addition we needed to decide what items we would need to order to design the line follower robot. In this first phase of our project, our main and initial design challenge was deciding how many wheels we would be using. After carefully considering the tracks we would be testing on we decided 4 wheels will give more stability at the cost of turning speed which would be a problem since the tracks provided had various sharp turns. The 2 wheels option would provide less balance and stability and would require a much smaller chassis, at the benefit of turning speed however this may end in the robot car not completing the track due to faults. So instead we decided to implement 3 wheels with a smooth caster ball wheel which would ultimately provide the same balance and stability as the 4 wheel option, but with a greater turning speed since the caster ball wheel provides minimal friction.

Figure 4a: 3 wheel chassis with caster ball wheel seen at the front



4.2 Motor Calibration

During the testing of the robot car on the tracks we noticed the car would drift off to the left whilst both motors were working at full power. This led us to believe that the left motor was travelling slower than the right motor, which would be the result of the left motor being slightly faulty. Since buying a single motor compatible with the right motor would not be possible without spending unnecessary money, so instead we decided to rewrite the code for the arduino to account for this speed differential. In order to do this we must use AnalogWrite when applying speed to the motors, this allows us to choose a speed from 0-255, after various tests on a blank white surface we found that a 255 to 150 ratio between the left and right motor was required for the robot car to travel in a straight line.

Arduino code for forward function(line not detected by both sensors):

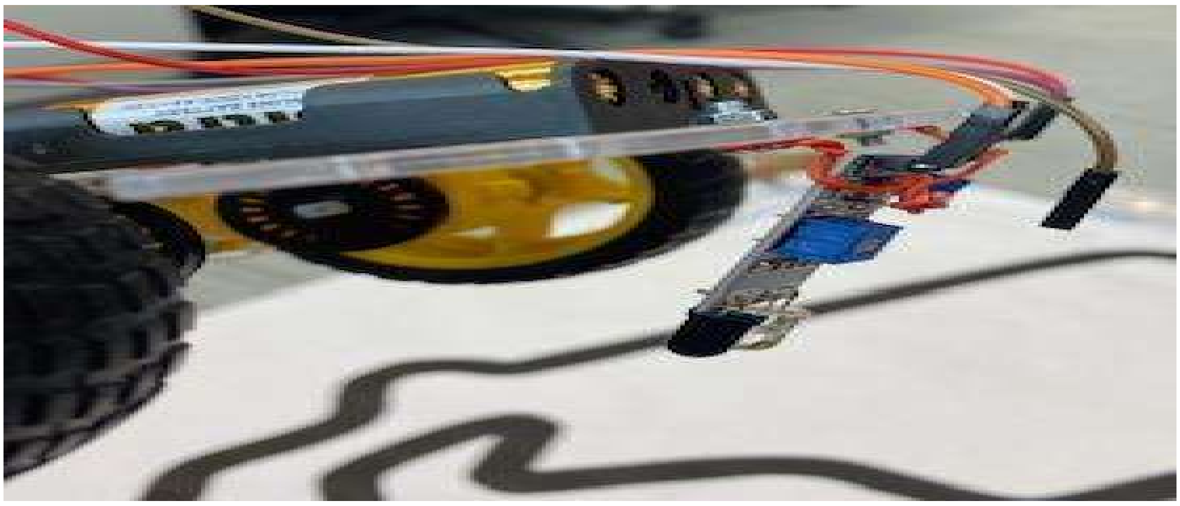
```
void Forward()
{
 Serial.println("line not detected");
 analogWrite(LM1,255);
  analogWrite(LM2,0);
  analogWrite(RM1,150);
  analogWrite(RM2,0);
  delay(40);
  analogWrite(LM1,0);
  analogWrite(LM2,0);
  analogWrite(RM1,0);
  analogWrite(RM2,0);
  delay(50);
}
```

The arduino code above shows the forward function used by the line follower robot, as mentioned earlier the 255 to 150 ratio allows the robot to travel in a straight line. However we noticed that although it was travelling straight, speed became an issue. Having attempted to reduce the speed through means of analogWrite, once the analogWrite value goes below 120 the speed drastically drops to a halt. To combat this we decided to use delays in between stopping and starting the motors, which in effect causes the robot to jitter with a reduced speed.

4.3 Sensor calibration

Once the Robot car was built calibration of the motors was the next task, however we noticed no matter the speed of the motors upon turning, the robot was still unable to make some turns with a degree of difficulty, and if the turns were successful they were not able to be reproduced. So we decided to look into the placement of the sensors, and we found that if the sensors were more aligned with the wheels it would be easier to follow the line around the sharp turns. So we decided to aim the sensors on the front of the robot backwards.

Figure 4b: infrared sensors aimed at a 45 degree angle to line up with wheels



This design could be improved by placing the sensors through the chassis, with the wheels aiming straight down. This would in effect increase the accuracy of the infrared sensors since the infrared light will travel a shorter distance and at no angle that could lead to error. However due to the limited space on the chassis this was impractical.

4.4 Number of sensors

Initially we believed 3 sensors would be a better fit than the two sensors we used, with the middle sensor following the black line, and the right and left sensors detecting changes on the white background. However in practice the middle sensor does not provide much aid to the robots ability to stay on the track, in actuality it was very difficult for the middle sensor to remain steady on the black line which led to a lot of unnecessary correction movements by the robot. So we decided to test 2 sensors, we found that the amount of time taken to complete the tracks was decreased.

4.5 Power Sources

Throughout the testing of the robot we faced a big problem with powering our robot: the batteries used drained far more power than we realised, so we had to explore the reasons for this power loss. Initially we used a motor driver IC to route power to the motors, but for this to work a breadboard would be needed with many intermediate connections in between, to tackle this we replaced the IC with L298 H Bridge, which we found would use less power in our research. Although the drainage was reduced it was not effective enough to remove the problem at hand, so we decided to power the infrared sensors separately from the Arduino Nano and motors. In order to do this we used the batteries to power the infrared sensors, and since the arduino is compatible with a USB type A/B cable, we used a power bank as the second source of power as it is: small, portable and is easily mounted on the chassis.

4.6 glossy track

Once we had the robot car functioning for the tracks at week 7, a new track was introduced. However, a glossy black line on a glossy white surface was used instead of the normal paper used on the other tracks. Upon testing this we saw that the robot car could not differentiate between the black line and white surface, this was due to the glossiness of the surface. Using the Arduino Serial monitor function along with the Analog Read of the infrared sensors, we found that the sensors returned the same values on the black and white surfaces. We tested the different sensitivities of the infrared sensors and even used translucent materials, such as tracing paper to try and remove the reflective effect the surface had, however this was to no avail. In order to successfully follow this new track we decided that we were to use an LDR sensor instead which would detect reflections of led lights, unlike the infrared light the infrared sensors would emit. This was as far into the design process we had gotten so the sensors could not be implemented.

4.7 Ultrasonic sensors

We was not able to add the ultrasonic sensors sensors due to limited time on the project

5. Testing

For this section we will be testing individual components of our robot car. The objective of this is to ensure all the functions of the robot are working correctly and to have observable data on what changes are the most optimal. Changes in the hardware and software will be based on these test results and all changes will be made to increase operational efficiency on the three different tracks given.

5.1 Motor

The motors have the purpose of moving the robot car under different situational conditions. We found that the motors worked fine and responded to the infrared sensors in time to complete the beginner course. However, we realized for medium and hard difficulty tracks the robot car would fail to pass the harder turns when just simply turning the outside wheel to complete a turn as the separation between the black and white areas was much smaller therefore testing of a different turning mechanic was necessary;.

Motor Test:

Test 1

Intro:
We will be testing if causing one wheel to reverse when turning increases success rate. Additionally, we will test the optimum amount of power that the reverse wheel should have to make the turn.

Test Method:

1. We will first test how many times our robot car fails when doing a turn.
2. Then we will change our code to cause the inside wheel to reverse and see how many times it fails.
3. Finally, we will record the amount of failures at multiple motor power values of the reverse wheel.
4. This test will be repeated on the medium difficulty track and the hard difficulty track. Easy track won't be tested on as turns don't require complex turning.

Control Variables:

- To keep it a fair test will make sure the analogWrite on the outside wheel will remain the same. We will make the value 255
- We make the turn on each test for each track the same

Medium Difficulty Map

Outside wheel turns, inside wheel stops	Amount of failures out of 20
analogue Write Inside wheel - 0	15

Outside wheel turns, inside wheel reverses	Amount of failures out of 20
analogue Write Inside wheel - 10	15
analogue Write Inside wheel - 20	12
analogue Write Inside wheel - 30	8
analogue Write Inside wheel - 40	4
analogue Write Inside wheel - 50	19

Hard Difficulty Map

Outside wheel turns, inside wheel stops	Amount of failures out of 20
analogue Write Inside wheel - 0	20

Outside wheel turns, inside wheel reverses	Amount of failures out of 20
analogue Write Inside wheel - 10	19
analogue Write Inside wheel - 20	15
analogue Write Inside wheel - 30	3
analogue Write Inside wheel - 40	12
analogue Write Inside wheel - 50	19

Conclusion:
From our results we can see that in most cases having the inside wheel reverse instead of stopping results in a higher success rate. Additionally, we can see that having power at 30

for the medium difficulty map works best while 40 works best for the harder difficulty map. From this we can deduct that it is more efficient to have separate code for each track difficulty. The one drawback from reversing one wheel is that it moves the car backwards a small amount which has led to an increase in turn time.

5.2 Infrared Sensors

The infrared sensors are meant to detect the change from black to white and vice versa so the robot car knows which direction to turn and when to go forward. We found that we had to change the position on the infrared sensors on the chassis to move optimally around the track.

<u>Infrared Sensor Test</u>
Test 2:

Introduction:
There is a delay between the sensor detecting a change from white to black or black to white and the motor responding to that change. For the wheels to turn in time we must put the sensors at an angle. We will be testing the most optimum angle for the sensors to be at.

Equipment:
-Stopwatch
-Protractor

Control Variables:

- Battery Power will be at 6V
- Separation of the infrared sensors will be kept the same

Test Method:

1. We will test the optimum angle by first starting at 0 degrees vertical to the ground and then increase the angle in intervals, the angle will be measured with a protractor.
2. We will be recording the number of failed attempts and also the time it takes to complete a turn for each interval.
3. We will repeat this for the three difficulties of track.
4. For the medium and hardest track, we will be testing the hairpin turn. The easiest track will just be tested on the 90-degree turn.

Beginner Track:

Angle (degrees)	Failed attempts out of 20	Average time taken to complete turn (seconds)
0	16	5.2
10	7	4.5
20	2	3.8
30	0	3.2
40	0	3.6
50	0	5.6
60	3	7.2

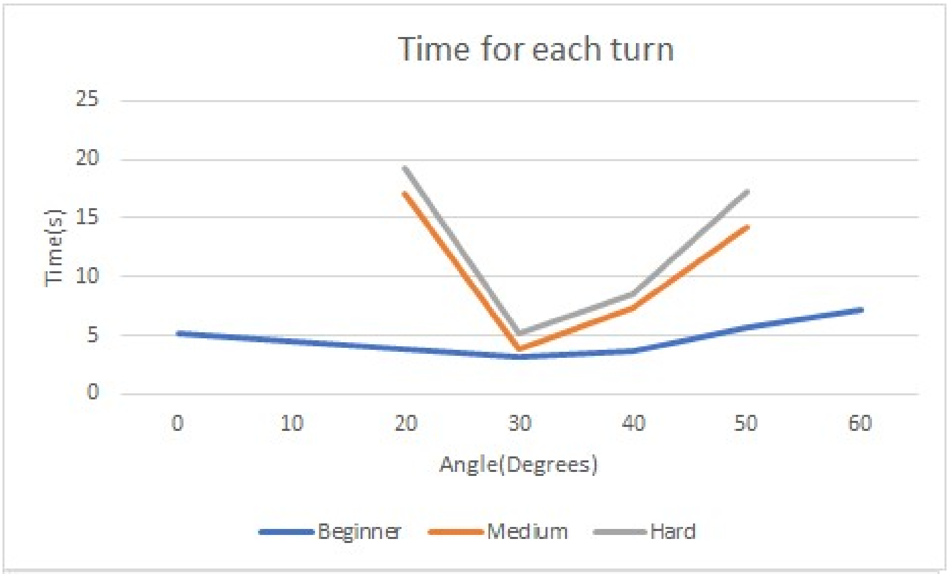
Medium Track:

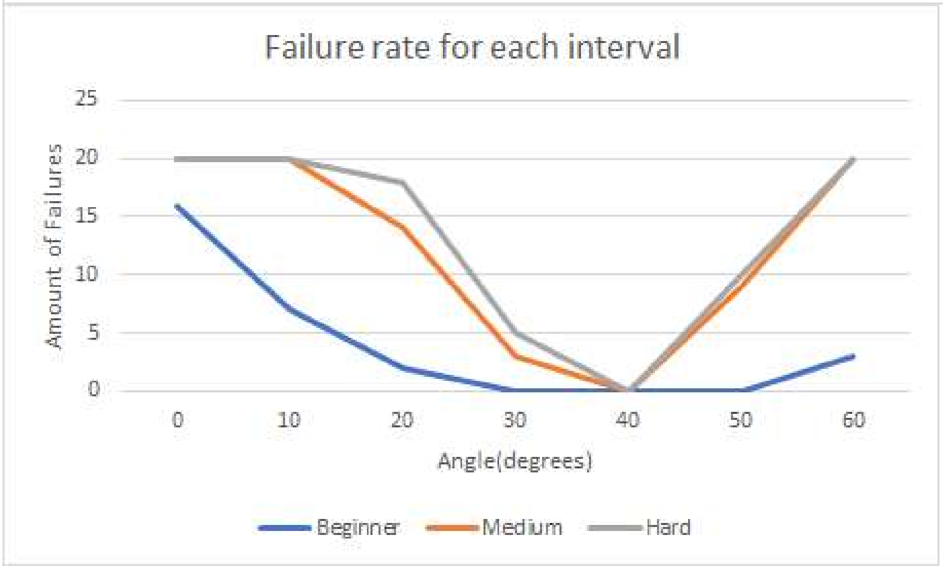
Angle (degrees)	Failed attempts out of 20	Average time taken to complete turn (seconds)
0	20	N/A
10	20	N/A
20	14	17.1
30	3	3.9
40	0	7.4
50	9	14.2
60	20	N/A

Hard Track:

Angle (degrees)	Failed attempts out of 20	Average time taken to complete turn (seconds)

0	20	N/A
10	20	N/A
20	18	19.2
30	5	5.2
40	0	8.6
50	10	17.3
60	20	N/A





Conclusions:
From our results we can see that 40 degrees is the optimal angle as we have zero fails at this angle. However, 30 degrees is optimal for the sensors if we want to complete the track in the fastest period of time. As the battery power decreases the speed of the motors decreases as well this may cause the robot car to work better at lower angles since the motors do not have to change as quickly when the robot car is moving slower.

Infrared Sensor Test:

Test 3:

Introduction:

This will be a test to find the functionality of infrared sensors on various types of surfaces to see if the sensor detects a change or not.

Test Method:

The infrared sensor will be connected to a power supply. The LED on the infrared sensor should go off or on depending on whether a black or white surface is detected. We will put the sensor 20mm above each surface that we will be testing.

Control Variable:

- Variable resistor on the sensor will be set to the same value throughout the tests.
- We will make sure the surfaces are in the same lighting conditions
- The infrared sensors will always receive the same amount of power

Surface	LED	Fail or success
White printed paper	On	Success
Black printer paper	Off	Success

White Matte	On	Success
Black Matte	Off	Success
White Glossy	On	Success
Black Glossy	On	Fail
White Luster	On	Success
Black Luster	On	Fail
White Metallic	On	Success
Black Metallic	On	Fail

Conclusion:

Both metallic and glossy are high gloss papers meaning they reflect a lot of light. This meant the infrared sensor couldn't detect the black line properly as the black part was reflecting light off its surface. Even though luster paper has lower gloss than these two it still creates the same issue. To combat this, we would use a different kind of sensor that detects colours instead such as LDR sensor.

5.3 Software

Our software worked functionally in all situations however we decided to add a delay to sacrifice speed for turning accuracy. Adding a delay additionally allowed us to more easily pinpoint problems in the robot car and troubleshoot them.

Test 4:

Introduction:
In order to turn more accurately a delay needs to be present. We will be testing how long it takes the robot car to complete the track and how many times it fails at different delay times.

Test Method:

```
Void Stop1()
{
        Serial.println("line detected left sensor");
        analogWrite(LM1,230);
        analogWrite(LM2,0);
        analogWrite(RM1,0);
        analogWrite(RM2,220);
        delay(x);
}

Void Stop2()
{
        Serial.println("line detected right sensor");
        analogWrite(LM1,0);
        analogWrite(LM2,230);
        analogWrite(RM1,220);
        analogWrite(RM2,0);
        delay(x);
}
```

- We will be changing the values inside delay(x) in the code above. We start off with no delay and will increase the delay to 200 milliseconds at regular intervals to find the optimal delay.
- Each time we change the delay we record the average amount of time it takes to complete a lap, we will repeat this 5 times.
- Furthermore, we will test this for each map difficulty.

Beginner difficulty:

Delay (milliseconds)	Failed attempts out of 5	Average Time taken to complete a lap (seconds)
0	0	28
25	0	35

50	0	48
100	0	61
150	0	89
200	0	113

Medium difficulty

Delay (milliseconds)	Failed attempts out of 5	Time taken to complete a lap(s)
0	5	N/A
25	3	48
50	2	59
100	0	73
150	0	99
200	0	136

Hard difficulty

Delay (milliseconds)	Failed attempts out of 5	Time taken to complete a lap(s)
0	5	N/A
25	5	N/A
50	3	82
100	0	91
150	0	139
200	0	192

Conclusion:
From our set of results we can see that increasing the delay results in longer lap times. The lower the delay the more failed attempts on the track occur. At 100 milliseconds and longer there are no more failed attempts. However, a longer delay causes the lap time to be longer therefore 100 millisecond is the optimal delay time as it has zero failed attempts and takes the least time to complete the lap.

Motor Test:
Test 5:

Introduction:
We have found out our left motor spins at a faster rate than our right motor. We will be changing the value in analogWrite of the left motor to see which value will make the robot car move in a straight line.

Test method:
We will first set the analogWrite for both motors at 255 (the max value) then slowly decrease the value for the left motor in regular intervals until the robot car moves in a straight line. We will use two strings 20 cm apart and of 70cm length. If the robot car passes over the string it fails in moving in a straight line.

Left Motor Power	Succeeds to move in straight line
255	No
230	No
215	No
200	No
175	Yes
150	Yes
135	No

Conclusion:
The results show that the left motor has to be set at between 175 and 150 for the robot car to move in a straight line. Therefore when the left motor is between these values in

analogWrite the robot car will be more efficient in completing a lap as it won't have to stop during the straights.

5.4 Battery

Battery:
The battery needs to be powerful enough to enable all functions of the robot car. The motors, Arduino and IC all take up power so voltage consumption for all these components need to be considered.

Test 6:
Introduction
Now we will be testing whether to see all components of our robot car will function properly at different voltage supplies.
Test method:
We will put the robot on the beginner track and test if the motors are turning and if the infrared sensors are able to detect colour change. We will start off with the lowest voltage level for the batteries and then increase it. We will be measuring efficiency of different power levels by timing how long the robot car takes to complete a lap if it manages to complete a lap.

Beginner Track

Voltage level	Infrared Sensor Works?	Motors work?	All functions of robot care are present?	Did it finish a lap?	Lap time(s)
1.5V	No	No	No	No	N/A
3V	Yes	No	No	No	N/A
4.5V	Yes	Yes	No	No	N\A
6V	Yes	Yes	Yes	Yes	60
7.5V	Yes	Yes	Yes	Yes	58

Conclusion:
The results show that when the voltage of the batteries is less than 6V the robot car cannot finish a lap even if some components are working partially. We also found that increasing the voltage to 7.5V led to a similar lap time as 6V. Therefore, 6v is the best choice as adding extra power doesn't change the lap time significantly.

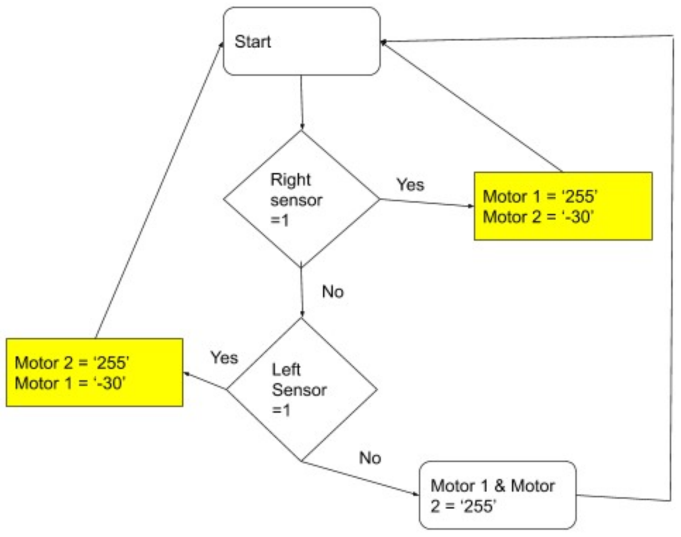
Motor Test:
Test 7:

Introduction:
At slower speeds the robot is less likely to encounter a situation where it won't be able to turn around a corner. However, this is at the cost of going around the lap in a longer period of time. By decreasing the max power to the motors incrementally we should find the point where the car doesn't fail to turn a corner and is at its maximum speed possible.

Control Variables:

- The same track will be used
- Starting points
- The reversing wheel will stay at a constant power
- Sensor position on the chassis will remain the same

5.5 Flow Chart


Start
Right sensor =1
Yes
Motor 1 = '255' Motor 2 = '-30'
No
Yes
Left Sensor =1
Motor 2 = '255' Motor 1 = '-30'
No
Motor 1 & Motor 2 = '255'

Method:

- Choose a corner from a track you will be testing on, this will be kept the same, we will be using the hairpin corners
- Mark a position where the car will always start from with something that won't change the white background of the track
- Choose a set amount of power going to the reverse wheel
- Change the power going to the forward wheel motor to the max
- Record the amount of times the car can complete the turn and record it in a table
- For any successful attempts record the amount of time it takes the car to complete the turn
- Repeat this while decreasing the power to the forward motor wheel by 10.

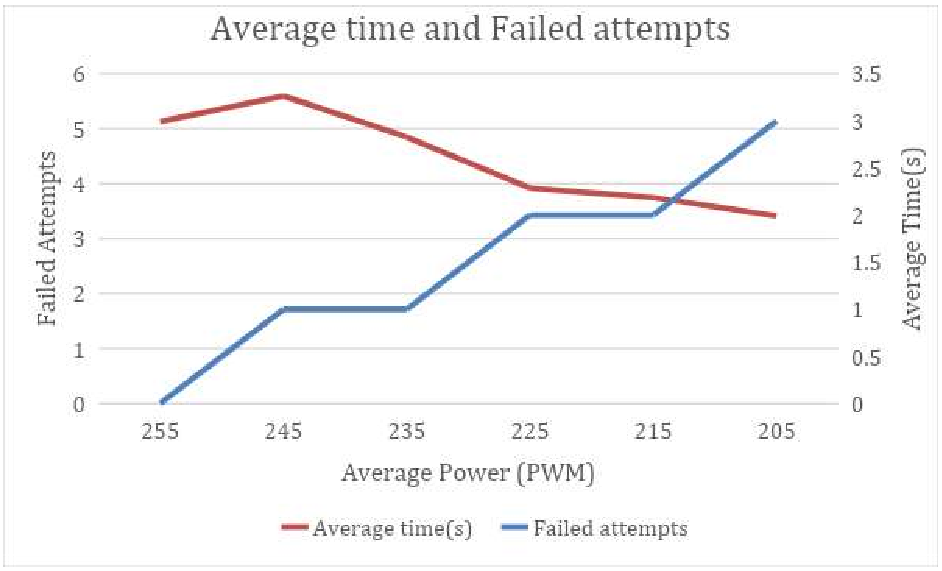
Average Power to motor(PWM)	No. Successful attempts	No. Failed attempts	Time to complete turn(s)	Average time(s)
255	1	3	3.41 | 0 | 0 | 0	3.41
245	2	2	3.80 | 3.70 | 0 | 0	3.75
235	2	2	4.21 | 0 | 3.63 | 0	3.92
225	3	1	5.01 | 4.91 | 0 | 4.63	4.85

215	3	1	5.53 5.93 5.34 0.	5.6
205	4	0	5.06 4.40 5.33 5.71	5.13

Possible Measurement errors:

- Reaction time

Graph



Conclusion:
The time it takes to complete the corner is increased due to the reduced average power to the motors. The lower the average power to the motor the more successful turns are made.

We can see from our graph that when the value of average power is 247 it is at the optimal point between the two variables of time to complete and failure rate

6. Discussion

This section covers challenges we have faced over the course of the project and how we combatted them. Additionally, we'll go over what our robot did right and what it did wrong. Furthermore, we'll also go over some improvements that could have been made in our design.

6.1 Challenges

Motors - Halfway during testing one of our motors became weaker than the other one. We had to constantly compensate with our code for the Arduino. We first found it was a problem with the power source and then it was a problem with the connections to the motor. This could've been avoided if we used lower power motors.

Battery life – Our battery life was a constant challenge in our project. At first we thought 4.5v would be enough to power the motor but as we added more components we realised we had to increase it to 6v. At 6v the power would slowly decrease causing the motors to become slower, we ended up buying more efficient batteries which solved most of the problem however if we bought rechargeable batteries it would have saved us more money in our budget.

Breadboard- During testing one of our motors completely stopped working meaning the car would just turn around in circles. It took us a long time to find the issue, as we didn't think the breadboard was the issue. We switched from a larger to a smaller breadboard so that the chassis would weigh less however the first smaller breadboard we used was faulty and broke easily. The metal strip in the breadboard was disconnected meaning the IC was not sending instructions to the motors. We could've avoided this by using higher quality breadboards or even introducing a pcb.

Software – Each track had different types of turns we had to overcome, we found it challenging to find a single code that could work efficiently on all tracks as their turns were so different. This meant during testing we had to keep switching out the uploaded code in the arduino which slowed us down.

Infrared Sensors – With our infrared sensors we ordered we found that some of them didn't work properly so it took us time sorting through which ones worked. Additionally changing the variable resistor on the sensor was very fiddly as it was sensitive and our sensors would constantly be moved around to find the perfect position and angle.This meant we had to do a lot of testing of the infrared sensor itself before we could get the robot car to start moving.

Cater wheel – Using a caster wheel allowed us to increase our speed however sometimes the caster wheel had to be manually moved by us to make it align straight with the chassis, this stopped it from moving to the side when travelling in a straight line. At lower speeds this wasn't a problem but at higher speeds we would find it would cause the robot car to divert sometimes.

Wires – We used wires that were given in the electronics lab however these were of all long length which made it hard to manage them and see which wire goes where, if we ordered wires or even cut and stripped them ourselves we could've made our cables much more organised.

What we did right and wrong:

Our robot car was successful. Each individual component worked correctly. We managed to combine all components and circuits together. The Arduino correctly sent instructions to the IC to move the motors according to the input from the sensors. We managed to complete all tracks of various difficulties. Our software handled nearly all the situations that we had to encounter.

However, our robot car was very slow during turns and couldn't follow a track on a glossy surface perfectly. We had issues combatting the reflection from the glossy background, this could've been handled with a different sensor for example an LDR sensor. Our speed was also limited by the amount of delays we had to have in order to make our turning success rate higher.

The biggest change we made that made our robot car more successful was changing how our code worked. Initially we had the black line in between our infrared sensors and the infrared sensors would cause the motors to change whenever a black line was detected. However, we realized it was more efficient to have one infrared sensor on the black line so that when both sensors detected white the robot car would turn, this allowed us to do the hairpin on the harder difficulty maps.

Our Arduino was one of the main reasons why our battery life would be reduced so quickly. We could've implemented a sleep function for our microcontroller so that the peripheral components didn't use up unnecessary power. This would've made the battery life on our robot car much longer and additionally keep the motors at a higher speed for longer.

6.2 Future improvements

These are improvements we could have made if we had more time or if we handled our budget better. All these improvements are things that would've made the robot car more efficient when completing tracks during testing.

Colour Sensor/LDR sensor – A Colour sensor that could detect lines of different colours not just black would allow our robot car to move on different background colors. Additionally a LDR sensor would have made the robot car work on glossy surfaces more easily. A colour sensor would've also helped with any scuff or shadows on the track as a lot of the time these would be detected as a change from white to black causing the robot car to turn.

Reset button – A reset button would have allowed us to stop and start the robot without taking out the wires connected to the battery supply during testing. When the robot car goes off track, we can easily disable it with this button so the car itself doesn't get damaged by hitting something.

Ceramic castor wheel – Using a ceramic caster wheel would've increased our lap speed as there would be less friction. However, it would take time installing the wheel onto our chassis as it would've had to come as a separate part. Additionally, a ceramic wheel would be more expensive.

Ball Castor wheel- Instead of using a wheel we could've used a ball; this would've given the chassis more stability as the ball can move in all directions and would eliminate having to set the caster wheel straight at the beginning of the testing. The only issue with this would be installation and we might have to possibly change the chassis design as the chassis would be much lower to the ground than originally.

Rechargeable Battery – A lot of the time our battery would drain out very quickly resulting in lower speeds and sometimes causing one motor to turn faster than the other motor. We wouldn't have to compensate with our code if we had a rechargeable battery and we also wouldn't have to keep on buying batteries and eat away at our budget.

Physical holder for infrared sensors- During testing we used tape to stick down the sensors, however after time the tape would weaken meaning we would have to secure the infrared sensors down again. Having a solid permanent holder would've saved us time positioning the sensors.

PCB- we could've implemented our IC into a PCB instead of a breadboard to reduce the amount of wire management we had to do. Only downside to this is that it would be harder to make any changes to the circuit and it would make it harder to troubleshoot any problems in the circuit so we would have to make sure the PCB was our final circuit design.

PIC microcontroller – Implementing a PIC would've allowed us to spend more of the budget on other design aspects of the chassis, the Arduino was one of the main costs of our project, using a PIC microcontroller would've been cheaper. The only issue is that our code would be more complex as we wouldn't have the safety net of the Arduino integrated development environment.

Protective shield – A protective shield would've stopped the infrared sensors from hitting objects when going off track and stop any other components from getting damaged. One downside to this is that it would increase the weight of the chassis and possibly make the robot car slower.

HC05 Bluetooth – we could've used the HC05 to program our Arduino wirelessly over Bluetooth. This would've saved time during testing as we wouldn't have to plug the Arduino back into the computer to upload any new changes we made to the code. Buying a HC05 would be a cheap addition with most components costing only one to two pounds.

Conclusion

The aim of our design and build project was to construct a robot car which can autonomously move over predefined track which specifically black line over a white background without any human support, as well as detection any obstacles presented in its track. The implementation of this robot car was based on an Arduino board and this was involving programming. So, we had the opportunity to apply our C-Programming language skills as well as the technical and theoretical knowledge we gained during the first and second year of our study to a real Engineering task. This project also allowed us to learn new concepts in circuit design and to further build out teamwork skills.

References:

[1]: Arduino, "Arduino Uno Rev3", Store.arduino.cc, 2019. [Online]. Available: https://store.arduino.cc/arduino-uno-rev3. [Accessed: 21- Apr- 2020].

[2]: Jayat, "Arduino vs Raspberry Pi - Difference between the Two", Circuitdigest.com, 2016. [Online]. Available: https://circuitdigest.com/article/arduino-vs-raspberryp-pidifference-between-the-two. [Accessed: 21- Apr- 2020].

[3]: The Pi Hut. 2020. Official UK Raspberry Pi 4 Power Supply (5.1V 3A). [online] Available at: <https://thepihut.com/products/raspberry-pi-psu-uk> [Accessed 21 April 2020].

[4]: Electronics Hub. 2020. *What Are The Differences Between Raspberry Pi And Arduino?*. [online] Available at: <https://www.electronicshub.org/raspberry-pi-vs-arduino/> [Accessed 21 April 2020].

[5]: Seeedstudio.com. 2020. *Choosing The Right Motor For Your Project – DC Vs Stepper Vs Servo Motors*. [online] Available at:

<https://www.seeedstudio.com/blog/2019/04/01/choosing-the-right-motor-for-your-project-dc-vs-stepper-vs-servo-motors/> [Accessed 21 April 2020].