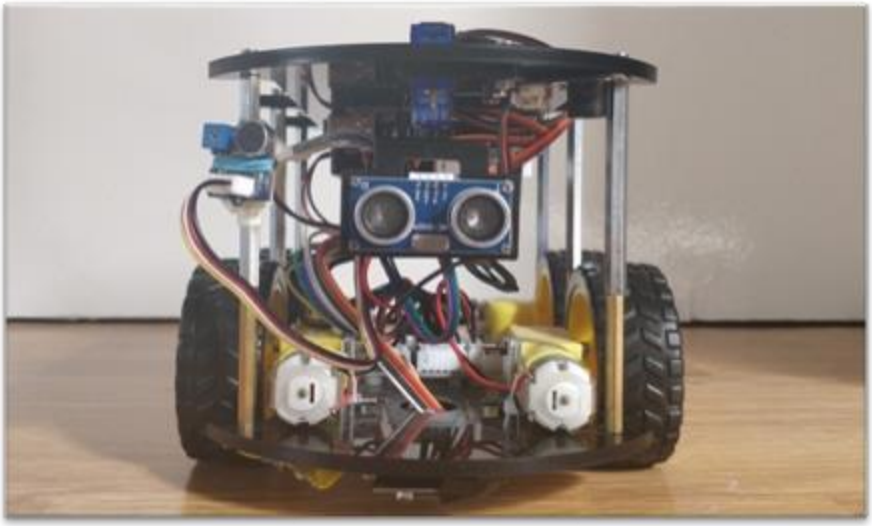
An Intelligent Transportation System That Communicates Intent



PROJECT REPORT

ECS514 Design and Build Project in Electronic Engineering

Group 9

Nafia Meem
190188399

Suvi Häärä
190234283

Nasmin Uddin
190436131

Michal Makowka
190290337

1 Abstract

The aim of this project was to design and build a model of an Intelligent Transportation System (ITS) that could communicate intent with its surroundings in addition to tracking different colour lines and detecting and avoiding obstacles. Over an 11-week period, the team designed a solution for the set specifications and conducted a series of tests. The data was used to change the design of the ITS model to better fulfil the requirements. A notable design decision to change the original line tracking module of the ELEGOO Smart Robot Car Kit with a TCS3200 Colour Sensor was made after extensive testing of the line tracking module revealed that even when used to read analogue data, it could not be used to create the robust system desired. A roundabout method was implemented to allow for obstacle avoidance on lines of different curvature. A Grove Loudness sensor and digital buzzer were added to the design to allow the ITS to communicate intent with its surroundings. It is essential that an ITS can communicate intent with its surroundings to ensure road safety. After the 11-week period, the team completed the model of the ITS and achieved level 4 automation.

Table of contents

2 Introduction

The aim of this project was to design and build a model of an Intelligent Transportation System (ITS) that could communicate intent, track various types of lines and detect and avoid obstacles in various scenarios. The aim was to create a system that could track lines of any colour or curvature, detect any obstacle placed in front of it with a height greater than 70mm, emit a sound when it detects an obstacle and give the obstacle time to move out of the ITS's path, avoid the obstacle if necessary, successfully return to the original path after avoiding the obstacle in addition to detecting and reacting to external sounds.

Technological advancement grows in almost an exponential manner and it won't be much longer until the bulk of traffic consists of ITSs. They are the future of transportation and have the potential to make roads safer for drivers and pedestrians alike. The motivation for creating a way for the ITS to communicate intent with its surroundings is to allow for better road safety when incorporating these systems on public roads. Merely programming the ITS to operate by traffic rules would not suffice, as it cannot be guaranteed that the human pedestrians or cyclists would adhere to the same rules as the ITS. While human drivers and pedestrians can communicate intent in both verbal and nonverbal ways, an ITS is not guaranteed to have this ability. This discrepancy could cause traffic accidents and therefore some form of communication between the ITS and its environment must be engineered.

This report details how the hardware and software of the system were designed and implemented and the processes the group went through to achieve the abovementioned features. It begins with the background research we conducted into autonomous vehicles, the sensors we used and the aims and specifications we set for ourselves. It then details how the software was designed and the reasoning behind the decisions made when designing the software and hardware of the system. This is followed by documentation of all testing that was carried out over the 11-week period and how these results impacted our design. An extensive evaluation of our design and any possible improvement leads to the conclusion of this project.

3 Background

3.1 Autonomous vehicles

An autonomous vehicle – also known as an auto, ITS, self-driving car, or driverless car – is a vehicle that can navigate through its environment without human input [4]. Current forecasts by industry professionals suggests that by 2030, 90-95% of new car sales will comprise of autonomous vehicles that only need human intervention when faced with unexpected conditions [2]. These vehicles typically use GPS, radar, laser lights, computer vision, SLAM (Simultaneous Localization and Mapping), cameras and other sensors to identify the desired route, possible obstacles in the way and react appropriately to their surroundings [3-4]. While ITSs typically are thought to be safer than human driven cars, an issue arises when testing and implementing them on to public roads where human drivers are still present. We are now approaching the time when it can be expected that increasing numbers of ITSs will be present on our roads. Vehicle manufacturers state that the expected period for these cars to appear on public roads is between 2020-2030 [3].

There are different levels of automation for vehicles. The levels below are based on cross referencing the different levels of automation defined by the NHTSA (National Highway Traffic Safety Administration), ISAE (International Society of Automotive Engineers) and VDA (Verband Der Autoindustrie), the German automotive industry association [3].

Level 0	No automated features. Full control of vehicle belongs to human driver, even if car can inform the driver of hazards. Driver is responsible for monitoring the environment.
Level 1	The vehicle is equipped with some automatic functionality, for example, automatic breaking or steering (not simultaneously), which can be activated spontaneously or by the driver. However, it is still the driver's responsibility to monitor the system and environment and take control when necessary.
Level 2	The vehicle is equipped with systems that allow the driver to no longer operate them, such as systems maintaining the vehicle in lane or adaptive cruise control. However, the driver is still responsible and must be ready to take control when necessary.
Level 3	The vehicle can perform all driving tasks under some circumstances, but the human driver must be ready to take control at any time necessary.
Level 4	High level automation. The vehicle can perform all driving tasks and monitor the driving environment in certain circumstances. In most circumstances, human drivers do not need to pay attention.
Level 5	Full automation. The vehicle is independent from human driver in all conditions. Driver acts in passenger role.

Table 1. Different levels of automation in vehicles based on data from NHTSA, ISAE and VDA
Source: [3][5]

3.2 Road safety

Approximately 1.35 million people die and 20-50 million suffer non-fatal injuries each year due to traffic accidents [1]. Out of eight causes of road accidents identified by the World Health Organisation, four are related to human error. Speeding, driving under the influence of drugs or alcohol, distracted driving and people not using seatbelts or helmets impact greatly whether an accident occurs and how severe such an accident would be [1]. An ITS would not experience the first three of these issues, as a vehicle that can drive without human intervention could be programmed to not speed, to always follow traffic rules and it would not experience distractions that bother human drivers such as fatigue, social distractions and emotions.

3.3 Strategies for Communicating Intent

Non-verbal strategies play an essential role in the communication between human drivers and pedestrians [2]. Human strategies such as eye-contact, hand-gestures and verbal communication have proven crucial in facilitating the safe flow of traffic [2]. As ITSs occupy public roads, it is possible that human pedestrians will struggle to adapt, as these subtle yet important strategies cannot be implemented anymore. Endrit Mulqolli conducted research into how affective different ways for ITSs to communicate intent were when trying to alert pedestrians. This experiment was limited, as it was a survey where participants were given time to interpret the signs placed in front of them, which they might not have in a real-life scenario [2]. While the study found that in most cases, visual stimulus was correctly interpreted by pedestrians, especially in cases where directions were written or based on traffic signs, its applications in real life are limited [2].

While visual stimulus can be used to communicate intent, it is limited, especially considering how distracted pedestrians in the modern world are due to, for example smart phones. We are constantly visually stimulated by our environment and our attention spans have increasingly decreased in the last few decades. Auditory stimulus triggers the human alarm system even in our sleep [6]. Sound can be detected in almost every case, while our other senses are limited to a specific scope (e.g. turning the head to see) [6]. A sudden noise can alert us that something has happened and the brain processes sound 10 to 100 times faster than visual stimulus [6]. Therefore, implementing a communication strategy of intent for ITSs based on sound could be highly beneficial for pedestrian road safety.

3.4 Traffic rules in the United Kingdom

Rule 112 of the UK Highway Code states that a driver should sound the horn “only while your vehicle is moving and you need to warn other road users of your presence”, that you should “[n]ever sound your horn aggressively” [8]. Driver should also not use their horns while stationary on the road or when driving in a built-up area between 11.30pm and 7.00am [8]. These things should be taken into consideration when designing ITSs that communicate intent to their surroundings within the UK.

Rules 162 to 167 of the Highway Code guide drivers on the appropriate ways to handle overtaking on UK roads. For example, drivers should only overtake when "it is safe and legal to do so", when "the road is sufficiently clear ahead", other "road users are not beginning to overtake you" and "there is a suitable gap in front of the road user you plan to overtake" [9]. They should primarily overtake from the right, as the traffic in the UK is right-hand sided. These things should be considered when implementing a design for a vehicle that would be put in use in the UK. The direction of traffic and basic overtaking rules have been considered in this project.

3.5 *Tracking roads*

A road tracking technique that has been fine-tuned for reliability and speed is detecting road lane-lines via a pipeline of computer vision algorithms [13]. The algorithms would take in raw RGB images, ideally from a front mounted camera, to produce the required lane-line segments that represent the boundary of the road for the car [13]. The method for lane detection stage generally tends to be the Hough Transform [14], which is a popular and relatively fast method for finding simple mathematical forms such as lines and circles in an image [15]. The research of Aly [16], Borkar et al. [17], Voisin et al. [18] and Wang et al. [19], all apply Hough Transform in their approaches.

Figure 1: Raw RGB image with Hough Transform line segments
Source: [13]

When the midpoint of the lane is found, it can be set to be the steering direction. However, this method by itself is not robust enough for practical application and is often combined with other methods to develop more suitable algorithms. For example, Wang et al. uses an algorithm called CHEVP (Canny/Hough Estimation of Vanishing Points), which is the combination of Canny edge detector and Hough Transform [19]. The general research focus has been on tracking lanes via front mounted cameras and computer vision algorithms, rather than sensors underneath the car that would allow the car to detect lane boundaries as it moves over them. However, the principle behind both techniques happen to be the same: directing the steering towards the middle of the lane or line. In this project, the tracks are significantly more narrow than the prototype itself, and therefore the track is treated as a line to be followed (via various

sensors directly underneath the scaled prototype), rather than a lane that the car is strictly required to stay inside of.

3.6 Sight in ITSs

One of the most complex tasks to implement on unmanned vehicles is obstacle detection and avoidance as the vehicle needs to detect the obstacles size, shape, and position to enable its control algorithm to find the safest course. Therefore, autonomous vehicles need an active safety system (DAS: Driver assistance system), which includes collision avoidance system, automatic breaking, adaptive cruise control and lane departure warning system [24].

The most popular approach by the automotive industry today is sensor based, as it provides ample number of options such as: LIDAR, RADAR, ELECTRO-optical, Sonar, Infrared, and Acoustic sensors. However, the most competitive sensors in the market currently are LIDAR and RADAR sensors.

LIDAR sensors are laser-based probes that are highly efficient in light detection and ranging and are often used with integrated photonics to emit precision-timed pulses to provide depth resolution. They work by emitting a succession of minuscule laser pulses – with the help of integrated photonics – to enable the LIDAR system to swiftly render a detailed profile of the object in its path down to the tiniest detail. These sensors are often placed on rooftops and hoods of the cars and rely on a complex spinning motion to scan the surrounding area. Although LIDARs are very precise and effective in almost all-weather conditions, they are expensive, bulky and not aesthetically pleasing [25]. Waymo (a subsidiary of Google's parent company Alphabet) is one of the leading providers of LIDAR sensors, in 2009 the cost per sensor was $75000, however through self-manufacturing the cost was reduced to $7500 per unit. However self-driving cars require 4 LIDAR sensors, making the cost increase to $30000 for the sensors alone and therefore increasing the overall manufacturing costs [26].

Figure 2: Self- driving Google car with LIDAR sensor
Source: [29]

As a result, vehicle manufacturing companies such as TESLA, opted to rely on a sensor fusion approach by integrating RADAR sensors with cameras [27] which are both cheap and small. RADAR sensors launch pulses of radio or microwaves to learn the location of the obstacle by

calculating the time taken for the beams to bounce back. These sensors work well in all weather conditions, they are effective at detecting speed and range for both short and long-distance detection. However, RADAR sensors provide low resolution data, therefore, cameras are used to augment the range resolution which provide excellent contrast information [28].

Finally, all autonomous cars use neural networks to integrate technical and environmental aspects to provide cars with the logic and sight of a human, neural networks allow cars to learn and adapt to their surroundings. However, this technology requires to process immense amount of data before the network can successfully be self-reliant. Consequently, no manufacturing company in the world has been able to produce a fully autonomous car. Due to the financial and practical challenges that RADAR and LIDAR present, it is worthwhile to investigate alternative methods to implement "sight" in ITSs, such as ultrasonic sensors.

3.7 Friction

In order to safely hand over control from the driver, as would be required in an ITS, we need an accurate estimation of how the tires would behave. i.e. the interaction between tires and the road. Tire dynamics tend to be non-linear and depend on several parameters, such as the slip angle, the coefficient of friction, the load distribution and tire conditions [10]. Testing friction conditions is difficult to mathematically model and requires pushing the vehicle to its limits to see how well it performs and this poses several challenges. Destructive testing methods are not plausible as self-driving vehicles are expensive to build or repair, sometimes in the order of millions of dollars. From this arose a need for small scale autonomous driving platforms that can model full scale vehicle dynamics [12]. In 2009, Lapapong et al. developed and experimented with similar scaled testbed, Pennsylvania State University Rolling Roadway Simulator (PURRS) [11]. The vehicle was tested on a treadmill and the data gathered indicated a good correlation between full scale vehicle dynamics and scaled models.

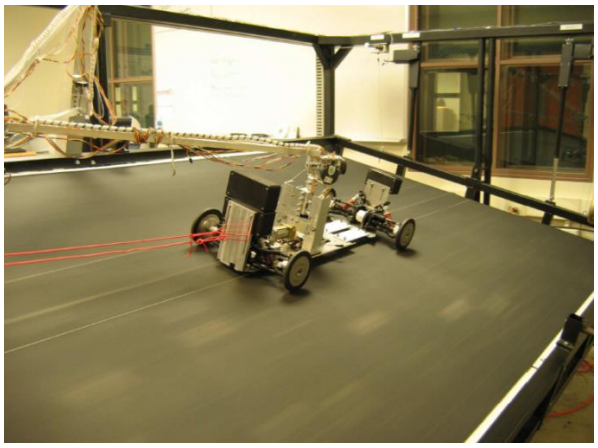


Figure 3: Pennsylvania State University Rolling Roadway Simulator
Source: [11]

While building such a bespoke testbed is outside the scope of this project, throughout the entire timeline of the project, it can be expected that effects of friction will be visible via sliding effects that the wheels often demonstrated. Hence, we must investigate the effects of friction and determine an optimal speed for the DC motors to operate at.

3.8 Sensors

3.8.1 Ultrasonic sensor

The ultrasonic sensor is used to measure a distance between the car and the nearest obstacle. It also facilitates the process of obstacle avoidance by establishing the direction in which the car must go in that process. It is controlled by reading digital values from the output through specific GPIO inputs. The sensor itself is composed of ultrasonic transmitters, receiver, and control circuit [30]. It has a range of 20mm to 4000mm and its accuracy can reach to 3mm [30]. When the sensor receives a high signal level on the I/O trigger for at least 10μs, the sensor sends eight 40kHz pulses [30] and detects whether the pulse signals are received back. This indicates whether there is something in front of the sensor, as the pulses are reflected towards the sensor. The sensor then outputs the time it takes for the pulse to be received by outputting high for the equivalent time. This time can then be used to calculate the distance between the sensor and the object.

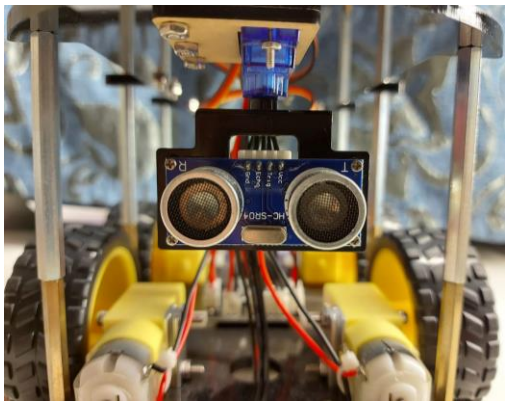


Figure 4: Ultrasonic Sensor

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Table 2. Ultrasonic Sensor Specifications
Source [30]

Line tracking module

The original line tracking module contains three reflective sensors (TCRT5000), each including an infrared emitter and phototransistor in a leaded package which blocks visible light [32]. The phototransistors present in the line tracking module receive the reflected infrared light from a surface and can determine whether the line is present or not depending on the intensity, as surfaces of different colours reflect different amounts of light. The distance between each sensor on the module is such that if the middle IR sensor records a line being present, the left and right sensors would normally detect white surface (on the provided Silverstone and EE Lab: Short Circuit tracks). The wavelength of the emitter LED is 950 nm that is on the top end of the infrared electromagnetic spectrum [32].

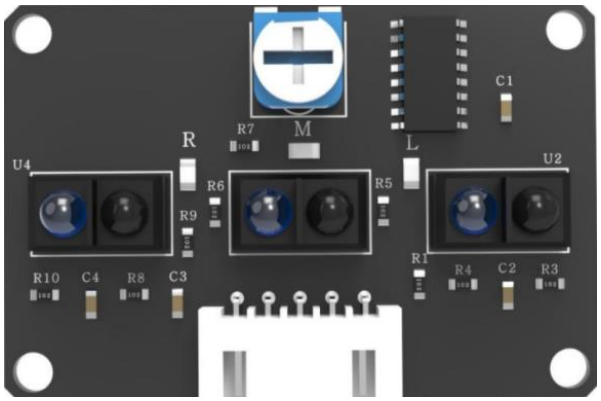


Figure 5: Original Line Tracking Module containing 3 TCRT500 sensors
Source: [33]

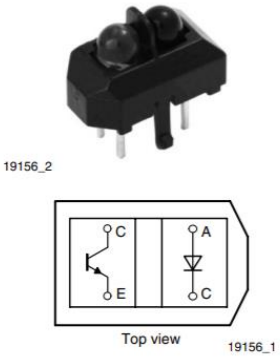


Figure 6: TCRT5000 Sensor
Source: [32]

3.8.2 Colour sensor

The original line tracking module has been replaced for a TCS3200 colour sensor which allows for a reliable and efficient line detection along the surface. Operation of the former one was based on an infrared reflection by a given surface, so that it is possible to distinguish between light and dark objects. Unfortunately, that did not suffice for a reliable detection of colours different than black and white. Hence, a new colour sensor has been introduced.

Here, the data is being outputted by a single PWM signal. The configuration regarding frequency scaling and colour channel selection is done as indicated in table 4. The f_o is hardwired to the value of 20%, as this one is the most optimal for ATmega328P external crystal resonator. See the system schematic for more details regarding the module interfacing.

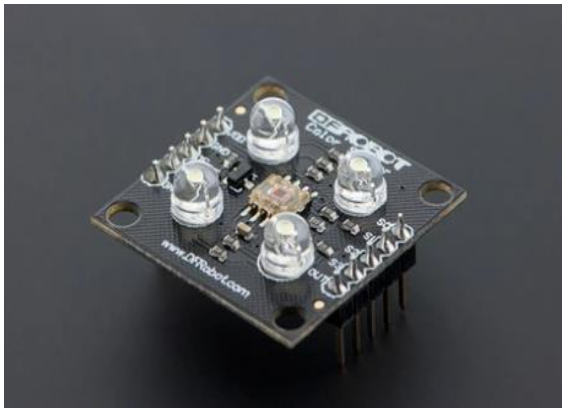


Figure 7: TCS3200 Module
Source:[20]

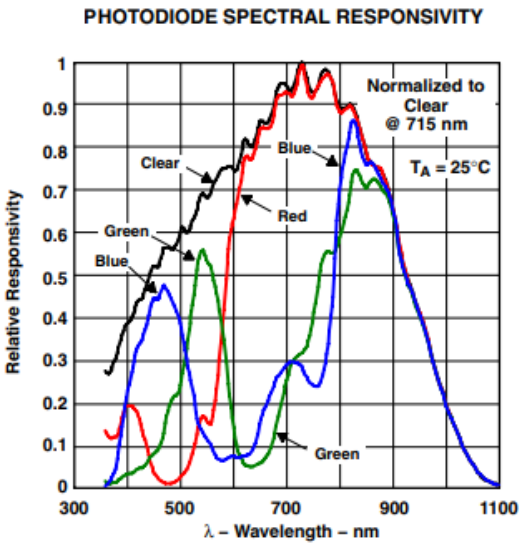


Figure 8: TCS3200 Relative responsivity per given colour channel
Source: [20]

The diagram presented above indicates a relationship between sensor's responsivity to a particular colour wavelength. It is possible to perform sampling and verification of a particular colour by taking into consideration all three RGB channels. Hence, we can obtain maximum operational accuracy and precision for this particular module.

		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD}		2.7	5	5.5	V
High-level input voltage, V_{IH}	V_{DD} = 2.7 V to 5.5 V	2		V_{DD}	V
Low-level input voltage, V_{IL}	V_{DD} = 2.7 V to 5.5 V	0		0.8	V
Operating free-air temperature range, T_A		–40		70	°C

Table 3: TCS3200 Recommended operating conditions
Source:[20]

As visible above, the recommended supply voltage V_{DD} is between 2.7-5V. The high-level and low-level input voltages are within the range of a standard TTL logic family.

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Table 4: TCS3200 Digital selectors configuration pins
Source:[20]

Table 4 indicates all possible selector configuration and corresponding frequency scaling/colour channel selection. By changing the values of S2/S3 and performing subsequent readings from the output channels, an entire colour spectrum of the wave can be properly processed.

3.8.3 Sound sensor

The sound sensor (also referred as *loudness sensor*) is designed to detect the environmental sound around the car. Its gain can be adjusted by manually rotating the on-board potentiometer. Analogue signal from the microphone is amplified by Texas Instruments™ LM2904 Op-Amp and outputted to one of the ADC channels of the ATmega328P.

Voltage	3.5~10 VDC
Working Frequency	50~2000 Hz
Sensitivity	-48~66 dB
Signal-to-noise Ratio	>58 dB
Output Signal range	Analog Signal (0-1023)

Table 5: Loudness Sensor Specifications
Source: [7]

Figure 9: Seeed Studio 101020063, Loudness Sensor for Grove System
Source: [7]

3.8.4 Buzzer module

Sound generation in the system is achieved by forcing a particular logic state on the NPN transistor embedded onto the buzzer module. Hence, the piezoelectric buzzer is active high.

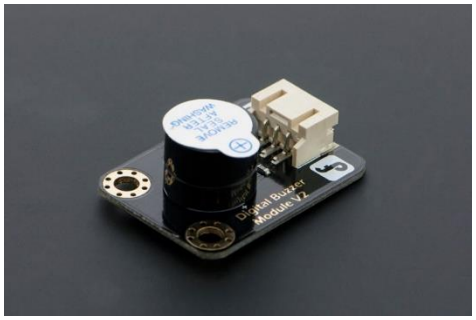


Figure 10: Buzzer module
Source:[31]

4 Aims of the project

The aim of this group project was to complete a working model for an ITS that could communicate intent during an eleven-week period. Our objective was to modify the provided ELEGOO Smart Robot Car Kit, so that it would be able to honk at the sight of an obstacle and react to external audio in addition to tracking a line and avoiding obstacles without human intervention. Each of the three sub-functionalities were divided into two phases of work. The first phase focused on getting the more integral, primary functions of each task to work. The aims in Phase 1 were:

- Have the model follow a black line
- Have the model stop when it detects an obstacle
- Have the model emit a sound at when it detects an obstacle

The aim of phase two was to get the vehicle to avoid obstacles and detect traffic or danger and perform with more accuracy and in more varied environments. This included ensuring that the vehicle could detect and follow non-black lines. We planned to ensure that the model could detect an obstacle and avoid it by passing it from the right whenever possible. This adheres to UK traffic rules, as cars that wish to overtake another will do so from the right. The vehicle should also return to the line it was avoiding the obstacle. If a wall or other obstacle that cannot be avoided is detected, the vehicle would find an alternative route. The aims in Phase 2 were:

- Ensure the model can detect and follow non-black lines
- Have the model avoid an obstacle
- Have the model return to the line after avoiding an obstacle
- Have the model react to external audio input by stopping temporarily
- Have the model react to multiple consecutive external audio peaks

Additional aims of the project included:

- Ensuring the reliability, consistency, and accuracy of our designs
- Creating a robust system that could handle a variety of scenarios and environment, such as:
    - An obstacle is located on a straight line
    - An obstacle is placed on a sight curve
    - An obstacle is on a sharp curve
    - The line colour is not black
    - The line colour can be any colour
    - The model can navigate around the tracks in the opposite direction
    - The tracks are different (EE Lab tracks, Silverstone tracks, any tracks)

5 Specification of requirements

5.1 User requirements

User requirements are subdivided into two categories: fundamental requirements and supplementary requirements. The fundamental requirements address all the functionality aspects necessary for the car to function and the preconditions set by our lecturers. The supplementary requirements address all the functionality that can be additionally added to the model to make it more refined.

5.1.1 Fundamental requirements

- Move in different directions: this is to allow basic movements of a car (forward, backward, left, and right).
- Detect obstacles from an adequate range: this is to allow the car to be aware of its surroundings and prevent collisions with other vehicles, objects, pedestrians, and animals etc. Additionally, it needs to be aware of obstacles from at least 6 cm away to allow the model to detect the obstacle from a safe distance.
- Avoid obstacles: this is to allow the car to avert obstacles safely.
- Track a black line on white background: this is to allow the car to function autonomously and travel safely without going off course.
- Resume the line path after obstacle is averted: this is to allow the autonomous car to adapt to unexpected course disruptions and successfully complete the journey without diverting from the original path.

5.1.2 Supplementary requirements

- Detect different colour lines: this is to test how well the car adapts to different roads – in the real-world roads are not uniformly coloured. This will make the model more reliable, adaptable, and therefore scalable.
- Detect noise inputs: this is to allow the autonomous car to detect environmental factors that might affect its trajectory that are not necessarily in front of it but are surrounding it and allow communication with other vehicles.
- React to noise inputs: this is to allow the car to be more responsive to noise and to add an additional functionality to the car that contributes to road safety.
- Output noise: to warn objects that come in front of it that they are in the way (to simulate situation where e.g. a pedestrian has moved in front of the model ITS).

5.2 Constraints

During the various stages of the project plan, we encountered the following types of constraints that impacted (or hindered) the project development stage. These constraints were subdivided into three categories: technical constraints, design constraints and logistic constraints. Technical constraints refer to the hardware constraints such as pin availability and component compatibility. Design constraints focus on the placement of the of the components and making a cohesive and functional model. Logistical constraints highlight the human constraints such as skillset and teamwork.

5.2.1 Technical constraints

- The pin availability on the control board was limited: this means that only a limited number of sensors/components have been integrated in the model as each sensor/component required multiple pins.
- The component compatibility was limited: this is because the voltage requirement varies with each component and different components require different connections that may not be supported by the Arduino UNO or the smart car shield.
- The line tracking module could only detect a line against a monochromatic background: the sensor for Arduino could detect the white lines in the black background and black lines in the white background. Therefore, the line tracking module provided in the kit could not be used to satisfy the user requirements.
- Single colour sensor module: as only one sensor was used -instead of three sensors like the line tacking module- the car needed to oscillate from side to side to compensate for the lack of sensors and therefore compromise smoothness of line tracking to be able to detect different coloured lines.
- Obstacle detection range: the ultrasonic sensor had a restricted effectual angle which meant that obstacles could only be detected if they were directly in front of the sensor.
- Friction on surfaces varied: the movements of the car were affected significantly as different surfaces caused different frictions and therefore could only handle certain surfaces.
- Components could not be soldered into the board: this was to prevent any damage to the components and allow all the teams to complete the project in an unbiased way. However, this meant that the car could not be fully customised to create a more comprehensive car model.
- Shipping time: different components had different shipping times which were also prone to sudden changes due to the current global pandemic, therefore damaged components were harder to replace.

5.2.2 Design constraints

- Limited time to complete the project: due to the 11-week deadline we had to consider the feasibility of the design that we would like to implement and be realistic about what can be achieved within that time.
- Limited budget (£30): the budget posed a big restriction on what additional components we could add to the model - as sensors and components were quite expensive. Therefore, the model did not have many additional features.
    - With the budget we purchased a colour sensor, a buzzer, an audio unit, and shafts to change the composition of the model to enable more functionality.
- Placement of the colour sensor: the module was placed under the car and remarkably close to the ground to detect the different colours more accurately. However, this meant that the sensor was prone to damage as small bumps in the path could collide with the sensor, change the placement of the sensor, and effect the software implementation.
- Only a certain size obstacle was detected: this is due to the sensor placement. Although the positioning of the sensor has been changed, the issue remained, as the sensor still was not able to detect obstacle that were below it.

5.2.3 Logistical constraints

- Only one person had the fully working prototype: this meant that not every member of the team was able to always access the final model, which affected the efficiency of the testing process.

5.3 Input and Output Requirements

5.3.1 Inputs

- **Frequency:** The programmable colour light-to-frequency converter TCS3200 produces an output (input to the system) frequency that is directly proportional to the amount of light being reflected from the surface [20]. The full-scale output frequency can be scaled by one of three pre-set values, 20% for interfacing with an Arduino, via two control input pins (S0 and S1) [20]. Pins S2 and S3 are used to select which group of photodiodes (red, green, blue) are active since the whole sensor hosts an 8X8 array of photodiodes for detecting different colours [20].
- **Ultrasonic Waves:** When an ultrasonic sensor detects reflected ultrasonic waves, it can indicate the distance at which the obstacle stands by setting the ECHO pin HIGH and having the duration of the HIGH-level pulse be the time between the wave being sent and receiving a reflected wave [21]. The sensor receives an input back for obstacles at distances of 2-400 cm for our HC-SR04 ultrasonic sensor, over an effectual angle of 15 degrees or less [21].

- **Sound:** We are using a loudness sensor to deal with certain safety hazards that are indicated by sound, e.g. horns from other cars, ambulance sirens or other loud sudden noises. It detects the intensity of environmental sound and our car carries out different operations based on the information collected, e.g. it stops whatever movement is it carrying out when it hears a loud noise. The analogue signal (range 0-1023) input from the microphone is amplified by Texas Instruments™ LM2904 Op-Amp and the gain can be adjusted via the potentiometer on-board [7].

5.3.2 Outputs

- **Power from DC Motor Driver:** Actuator L298 dual full bridge driver drives 4 DC motors in the car [22]. Based on automated collection of inputs from various sensors, our motor driver takes low-level voltage and current from the microcontroller and supplies the high levels (5-35V and max. current of 2A) that the DC motors require, controlling both speed (via PWM signal) and rotation direction (via pins that control switches of the H-bridge in the driver module) [22].
- **White Light:** Four LEDs in the TCS3200 colour sensor transmit white light onto the surface the car is travelling on, for the reflected light to be detected by the photodiodes and eventually converted to red, green and blue frequencies. This is used for detecting different colours and hence, the presence or absence of a line.
- **Torque Output:**
    - Produced by the servo motor for rotation of ultrasonic sensor. For obstacle detecting purposes, we need our ultrasonic sensor to be able to rotate through a certain angle in its field of vision. The servo motor we are using, SG90 9g Servo Motor, is able to operate over a 180 degree range (90 in each direction) at a speed of 0.12 seconds/60 degree at an operating voltage of 4.8 V. The torque output produced is 2.5 kg/cm [23].
    - Produced by the 4 DC Motors (wheels of the car), being driven by the L298 dual full bridge driver. Speed and direction of the motors are both commanded by the driver IC [22].
- **Ultrasonic Waves:** When triggered (via a 5 V pulse for 10 microseconds), the ultrasonic sensor transmits 8 cycles of ultrasonic burst (square waves) 40kHz and waits for the reflected ultrasonic burst [21]. This is then used to determine the distance between the obstacle and the car, as the duration of the high-level pulse is be set to the time between the wave being sent and receiving a reflected wave [21].
- **Sound:** To indicate an obstacle it has come across; we have incorporated a buzzer (a piezoelectric speaker) into the system to simulate a car horn by transmitting audio signals.

5.4 Operational Requirements

The system needs to incorporate a certain number of pre-defined operational modes that will be executed depending on the environment at a particular instance. Ones are listed and described below:

- *Colour Sampling*

  This function reads and stores a digital colour value that determines a line.

- *Line Tracking*

  This function executes the line tracking based on a colour sampled.

- *Obstacle Detection*

  This function performs obstacle check, and decides whether the journey may be continued, or the obstacle must be avoided.

- *Obstacle Avoidance*

  This function performs obstacle avoidance.

- *Sound Detection*

  This function is executed when a sound beyond a certain gain threshold is detected.

Each of the modes can be entered asynchronously, or directly after exiting the preceding state. Only one system mode can be active at the particular instance. The operational integration of the car system is presented in Figure 11.

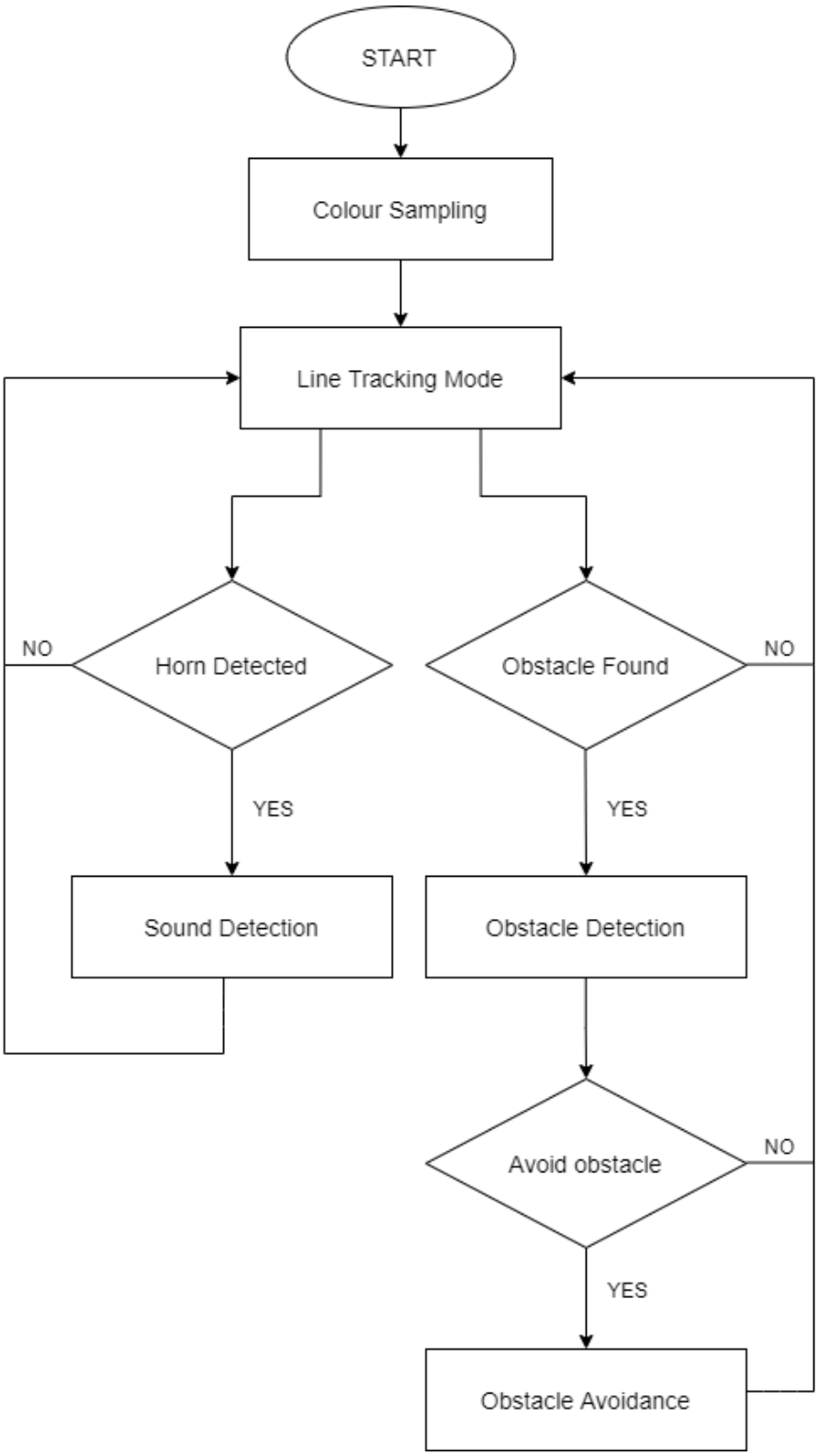


Figure 11. Operational modes

5.4.1 Operational Modes

Colour Sampling – In this mode, the car is initially placed on a track to be followed. The system reads a value from the TCS3200 colour sensor and sets it as a target to be recognised as a line to be followed.

Line Tracking– In this mode the car follows the previously detected line. Due to a nature of the sensor, the line tracking must be performed by implementing synchronous swing motion to make sure that the car is aware of the side it is currently positioned, in respect to the line.

Obstacle Detection - This mode is enabled asynchronously when an obstacle is detected. The car waits for the obstacle to move. If it does not, the system enters the *Obstacle Avoidance* mode immediately after.

Obstacle Avoidance - This mode is enabled when the car needs to avoid an obstacle. By implementing a roundabout approach *(see Program Design*) the model should perform the task safely and efficiently. After re-detecting the line, the system jumps back to the *Line Tracking* function.

Sound Detection – This function stops and delays car operation for the safety reasons. At the same time, it zeros the rotational swing increment to make the car able to detect a line, particularly on a sharp turn.

5.5 Functional requirements

The systems need to embed a special set of functions to meet the operational requirements. The functions specified below integrate sensors and actuators via Arduino Uno board and ATmega328PU microcontroller and form a basis for further software development.

Sensors available: Ultrasonic sensor, TCS3200 colour sensor module, loudness sensor

Actuators available: 4X DC motor, SG-90 servomotor for the ultrasonic sensor

5.5.1 Function set

Idle - An idle state of the system. No action takes place at that particular instance.

Find_line – A function that control the output motors (and therefore the position) depending on the line follower sensor input.

Follow_line - The main function of the system located in the main loop. Active directly after *Find_line*. It takes an input from the line follower sensor and outputs a specific motor rotation.

Detect_obstacle – a function that allows detecting the obstacle by taking an input from an ultrasonic sensor. It makes a decision whether to omit it after a pre-programmed timeout

Avoid_obstacle – a function that allows omitting the obstacle if a decision is made by a car to do so after the detection.

Controlled_delay – A function that checks for an obstacle and sound, while executing the delay. The most simplified form for executing an asynchronous control of the environment.

Emergency – this function is enabled when sound input above a certain gain (dB) is detected by the sensor. Then a specific action is taken by the system.

More details about how these functions are executed in a real-life system are presented later in this document.

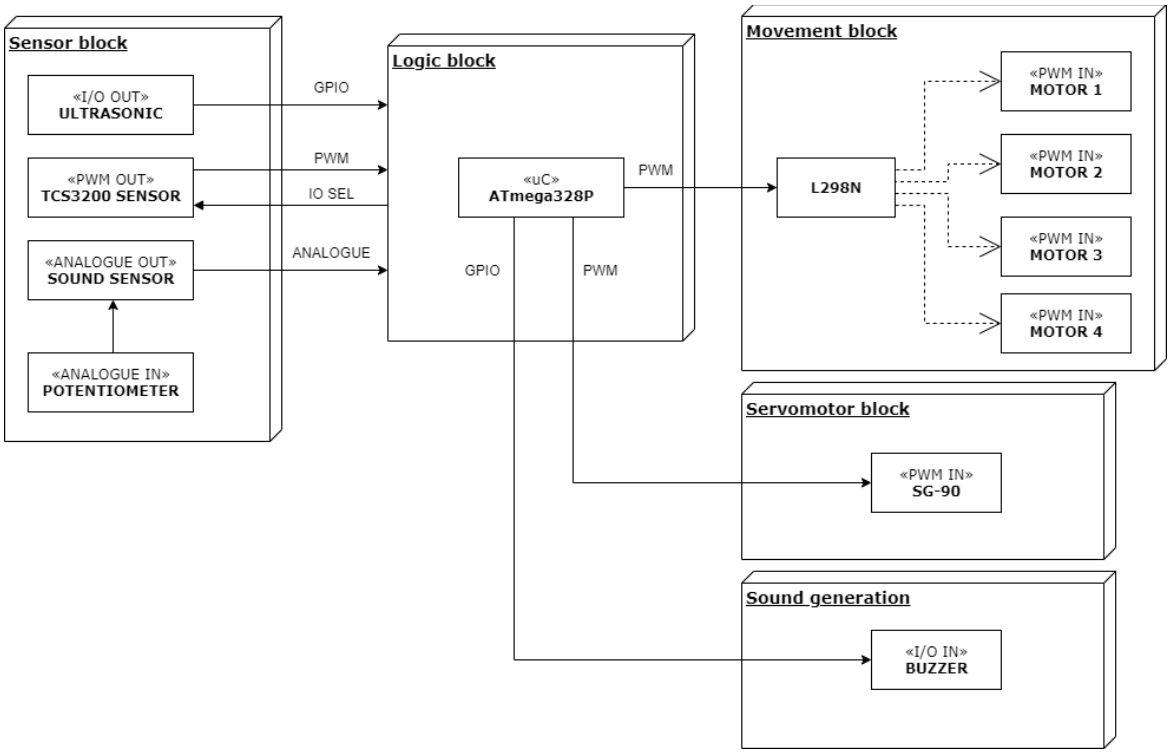


Figure 12. Functional block schematic

As shown in Figure 2, the *Arduino UNO block* is a hardware link between the sensors (*sensor block*), and actuators (*movement block* and *servomotor block*). The entire logic of the functional requirements is embedded into the microcontroller and is determined by the software burned via *Arduino* bootloader. Refer to the system schematic for more details.

5.6 Non-functional Requirements

- **Usability:** The operation of the ITS needs to account for a pleasant user experience. For example, the breaking and acceleration of the ITS must be smooth to ensure the safety and comfort of a passenger.
- **Maintainability:** We must ensure that adding and removing software or hardware is easy, that we can repair issues in the ITS and monitor the functionality of the system throughout the project. Our design must be adaptable throughout the process, so that we can improve upon what we create as we progress and test the functionality of our system.
- **Regulatory:** The model will follow standard UK traffic rules, so that when the model operates as it would be in the environment it is to simulate. For example, the direction of opposite traffic will be kept in mind when swerving to avoid obstacles.
- **Scalability:** Throughout the entire project timeline, we have kept in mind the bigger picture: how our model's properties can be applied on a larger scale. For example, a safety aspect we will consider are hazards on the road indicated by sudden sounds, e.g. car horn or ambulance siren. This leaves room for improvement as the sound detection process can be refined to a more sophisticated version when it is scaled up.
- **Security:** When scaling the use of ITSs to full scale traffic, some form of a security protocol must be implemented to ensure road safety. The protocol would have to safeguard against hacking and cyber-crime, e.g. by having a multi-factor verification system in place, to ensure that operation code is not altered by external people. This can be investigated further in the project if all prior phases are completed.

6 Design

This section of the report focuses on the design we implemented for the ITS model, more specifically the designs we used in relation to hardware and software designs of the project.

6.1 Hardware design

In this section a detailed description of the hardware design and configuration of the final version of the ITS that can communicate intent is described.

6.1.1 Pin configuration

Pin number	Connection
0	TX – serial
1	RX – serial
2	Colour sensor S2
3	Servomotor
4	Colour sensor S3
5	Motor control
6	Motor control
7	Motor control
8	Motor control
9	Motor control
10	Colour sensor output
11	Motor control
12	Colour sensor LED
13	Do Not Use (Arduino bootloader)
A0	Sound sensor
A1	Buzzer
A2	Not in use
A3	Not in use
A4	Ultrasonic
A5	Ultrasonic

Table 6: Interfacing connections

6.1.2 Hardware modifications

Hardware modifications implemented mostly concern redesigning the structure (i.e. increasing structural shafts' length), and replacing or adding additional sensors to fulfil functional and operational requirements imposed. As visible on figure 14, now entire componentry is safely interconnected and placed in-between the black planes. The ultrasonic sensor has been inverted along the horizontal axis to allow obstacle detection 7cm above the ground[1]; sound sensor, colour sensor and buzzer module have been put in place.

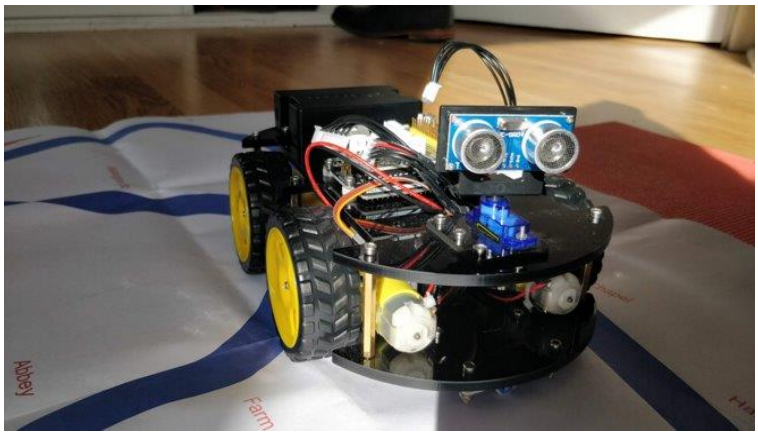


Figure 13: Car kit structure before implementing modifications

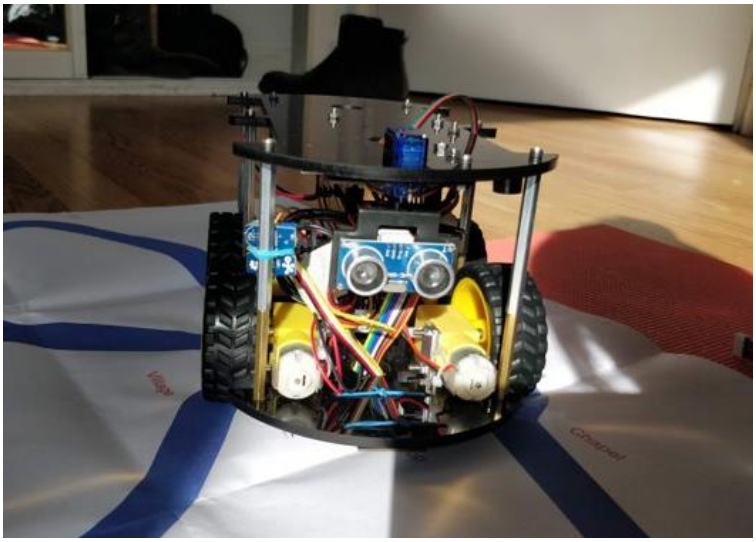


Figure 14: Car kit structure after implementing modifications

[1] In the original design the ultrasonic sensor is mounted 10cm above the ground.

6.1.3 Interconnections

As shown beforehand, all sensors and actuators are interconnected to the Arduino Uno logic block. This section clearly outlines the nature of a signal being transmitted and received between the microcontroller and any of the external modules.

Ultrasonic Sensor:

- Single GPIO pin – Digital signal output

In case an object is detected within the distance set, the logic state of an I/O pin is toggled. This is recognised and further processed by the microcontroller.

Colour Sensor:

- Single GPIO pin - PWM signal output
- Two GPIO pin selectors – Digital signal input

Readings from all three colour channels are outputted through a single GPIO pin as a PWM signal. Channel selection at a given instance can be set via digital selectors

Sound Sensor:

- Single GPIO pin - Analogue signal output

Sound gain is outputted as an analogue signal in the range GND (no sound) – 5V(maximum loudness). This is further processed by an AVR's embedded ADC and stored in a variable within a numerical range 0-1024.

Motor Control:

- (4 Channels) Single GPIO pin - PWM signal input

PWM signal from the microcontroller is inputted to two dual full-bridge drivers which are then connected to four DC motors directly. By changing the frequency, a direction and angular velocity can be controlled.

Servomotor Control:

- Single GPIO pin - PWM signal input

PWM signal from the microcontroller is inputted to the servomotor, thereby a corresponding angular position of the mechanism is being set.

Buzzer:

- Single GPIO pin - Digital signal input

By toggling the logic state of a corresponding GPIO pin, the buzzer is activated through an NPN transistor-based switching circuit

6.1.4 System Schematic

See a complete system schematic on the next page.

KEEP CLEAR

THIS PAGE WILL BE REPLACED WITH A SCHEMATIC

In Adobe Acrobat DC after export

6.2 Program design

In this section is a detailed description of the software design and configuration of the final version of the ITS that can communicate intent.

6.2.1 Line tracking

Stage 1 – Differentiating between colours:

Once the program is set in motion, the TCS3200 colour sensor records the red, green and blue frequencies it is receiving from the surface it is placed on. This is done only once during set-up. For the remainder of the car's journey, these recorded values are considered the benchmark for an indication of presence of a line. The leeway allowed for variation in values is 15 kHz for all three frequency channels. If input values from one or more of the frequency channels fall outside of this range, the program registers an absence of a line.

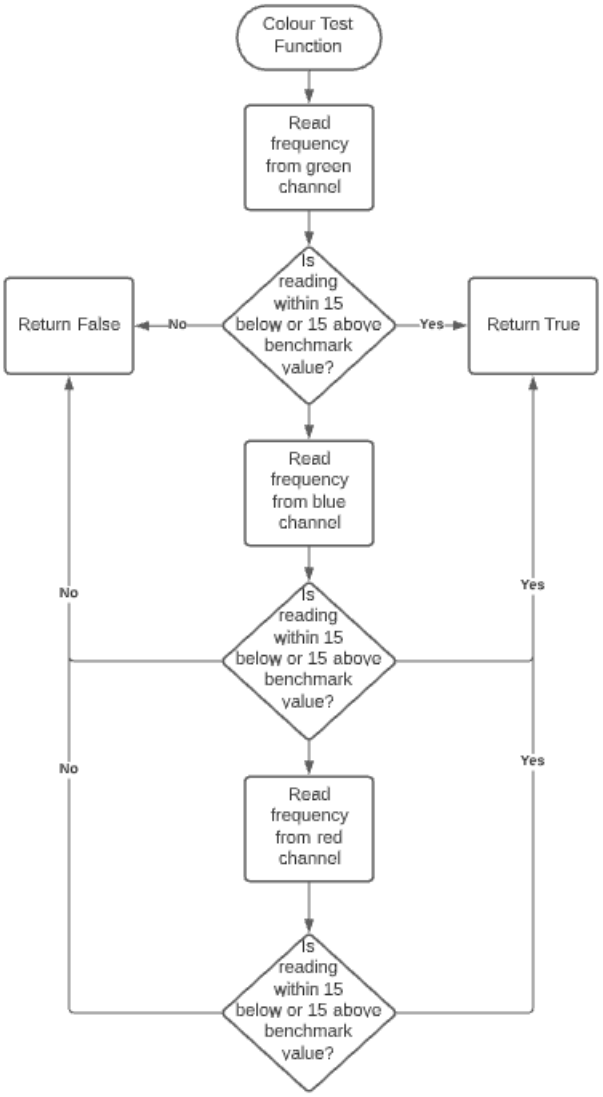


Figure 15: Detecting a Line

Stage 2 – Using colour information to follow track:

The car follows two different protocols depending on whether it has arrived at a turn or is following a continuous section of the track. When it does not detect a turn, as illustrated on the left section of the following flowchart, a bi-directional swinging motion is implemented to allow the car to stay on track. Once the car swings left and is brought off track, it is made to swing right. This perpetual swinging left and right allows for a high degree of accuracy of line tracking as the car moves forward.

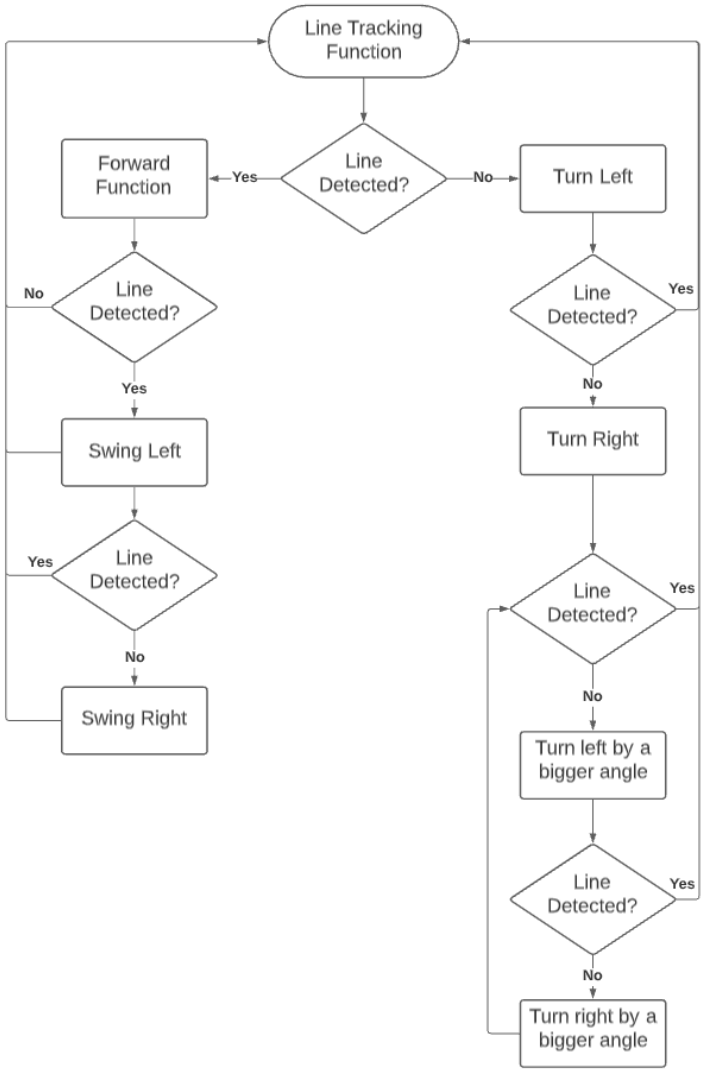


Figure 16: Oscillating Movement of the Car

When the car does arrive at a turn, it turns left by a certain angle and checks for a track. If a track is not detected, it checks again after turning right by the same angle, measured from the middle point. However, as long as a track is not detected, the angle by which the car turns is made to increase in equal increments. The increment is realised by allowing the car to keep turning for a longer amount of time via the delay() function, adding 25 ms in each iteration of a while loop until a line is detected. This makes the turning mechanism more efficient as, for example, the car does not keep turning a full 180 degrees left when the turn may in fact be at a much smaller angle to the right. After it has detected the turn, the car is able to switch to the forward function and keep moving forward.

6.2.2 Obstacle avoidance

Stage 1 – Detecting an Obstacle

The main objective for phase one -in terms of obstacle detection and avoidance- is to detect obstacles using an ultrasonic sensor. Obstacle detection is designed to allow the car to detect any obstructions on the road without contact to prevent collisions with oncoming traffic, objects, animals, and people.

For this project, the car was programmed to detect obstacles that are within 6 cm from the ultrasonic sensor (positioned on the head of the car). This is to make the car aware of its surroundings and enables it to detect obstacle from a safe distance; the distance between the car and obstacle is needed to facilitate obstacle avoidance mode (as obstacle avoidance mode is not able to keep track of the position of the obstacle while moving).

The flowchart below demonstrates how the obstacle detection function works.
Firstly, the car checks if the obstacle is detected. Once the obstacle is detected, the car stops and honks at the obstruction. A delay of 2 seconds is applied to enable the obstacle to move out of the way. If the obstacle has moved, the car enables line tracking mode and continues to move through the path, otherwise, obstacle avoidance mode is enabled.

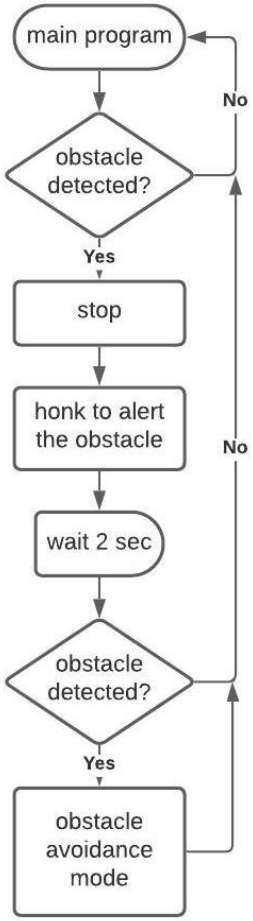


Figure 17: Obstacle detection flowchart illustrating obstacle avoidance mode calling

Stage 2 – Avoiding an Obstacle

Phase 2 focuses on obstacle avoidance of static obstructions.

Firstly, the car checks if the obstacle is still present and within the 6 cm range. Once the initial condition is met, the car starts to assess the surrounding area to calculate the best path to take to avoid the obstacle.

If the distance to the right of the obstacle is greater than the distance to the left of the obstacle – which implies more space to move and therefore reducing the probabilities of encountering more obstacles - the car chooses the path to the right and vice versa.

Once one of the directions is chosen the car slowly moves around the obstacle (in a roundabout method) to constantly check for the original path (i.e. line). To enable line detection while moving around the obstacle avoidance path 100 checkpoints are implemented, this is to allow the car to be able to detect the line as frequently and as accurately as possible.

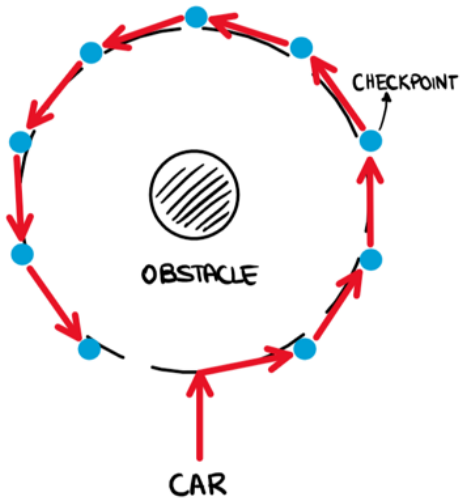


Figure 18: Diagram illustrating how multiple checkpoints are implemented in the roundabout method

Consequently, once the line is detected the car initiates line tracking mode and resumes the journey. However, if there is not enough space on either side of the obstacle to avoid it – which implies either the obstacle is too large to avoid it without collision or there is no additional path to avoid the obstacle – the car reverses and exits obstacle avoidance mode. The ultrasonic sensor is placed upside down therefore left and right in respect to the ultrasonic sensor are reversed.

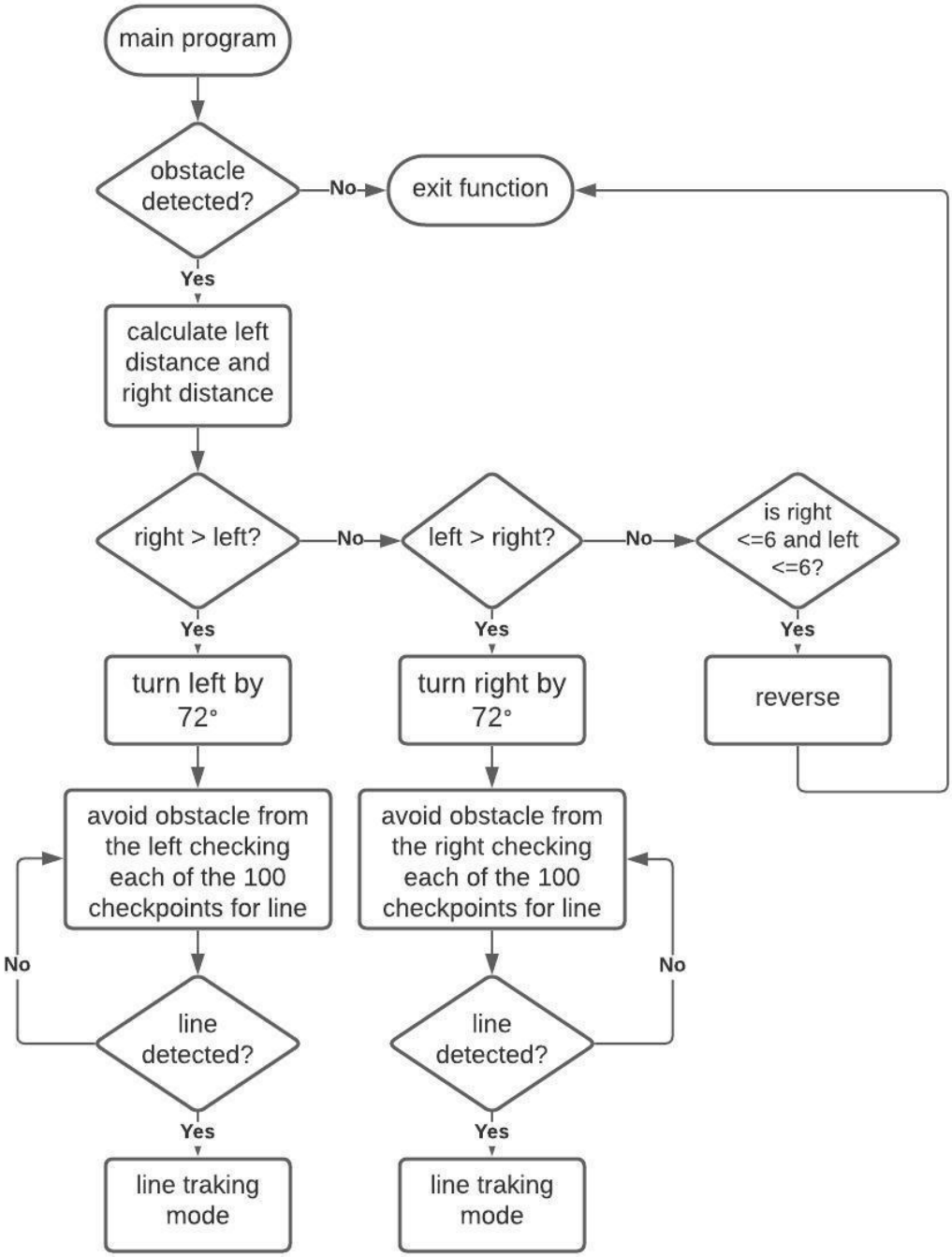


Figure 19: Diagram illustrating how obstacle avoidance mode works

6.2.3 Audio Unit

Stage 1 – Emitting a Sound

Phase one of the audio unit focused on making sure that the ITS could emit a sound from a buzzer module for 0.5 seconds when it detects an obstacle. This would be a way for the car to express intention to its surroundings, as it can make those around it aware of its presence and that there is something in its way. It then is designed to wait for 2 seconds after the buzzer output to give the obstacle time to move out of the way, so that its surroundings have time to react to the ITS expressing its intentions. If the obstacle has moved away from in front of it, the ITS will continue operation in the main program, detecting and following the line. However, if the obstacle does not move away, the ITS will move past the object as designed in obstacle avoidance.

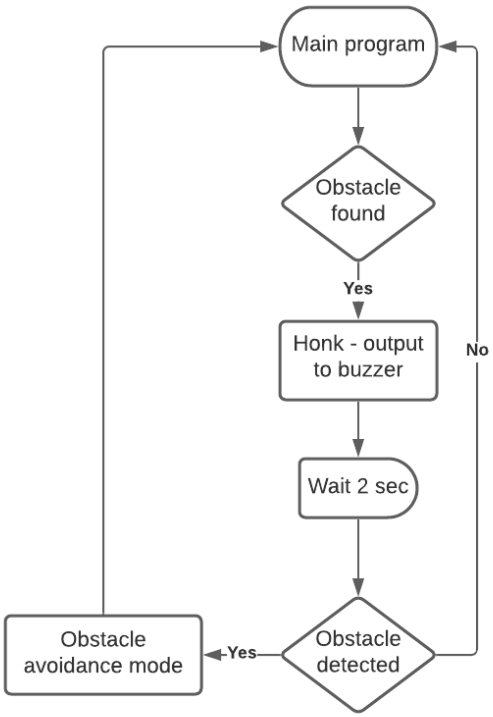


Figure 10: Flow chart of the functionality of the honk function

Stage 2 – Detecting and Reacting to a Sound

The audio unit design would then be improved upon, so that it can detect and react to external audio input. This would be a way for the ITS to acknowledge and react appropriately to the intentions of other vehicles or pedestrians on the road, which is essential for road safety. The car is designed to stop for 5 seconds when an external audio exceeding a predetermined threshold value is exceeded.

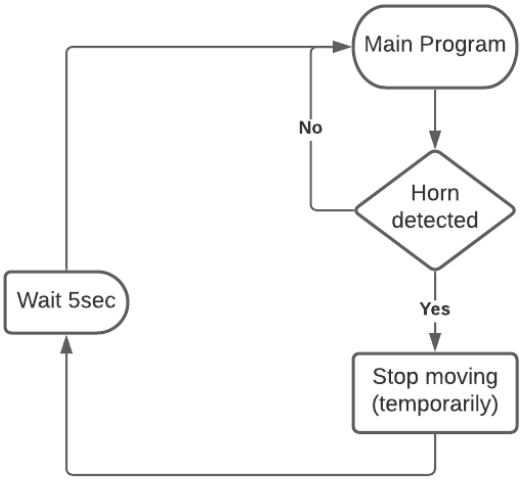


Figure 21: Flow chart of the functionality of the emergency mode function – how the ITS reacts to external audio stimulus

6.2.4 Custom delay

An essential piece of the program we created was a custom delay function that allowed for the sound and obstacle detection to occur even when carrying out the swinging motion at sharp curves caused by the line tracking software implementation. See the snippet of code below:

```
// Sound and obstacle check enabled when using c_delay
void c_delay(u16 deldeldel) {
  for(int i=0; i<(deldeldel/5); i++) {
    delay(5);
    check_obstacle();
    if(check_sound()){
          emergency_mode();
          //reset values for line tracking
          rot_l = 25;
          rot_r = 27;
          inc = 1;
       }
   }
}
```

Figure 22: Program snippet for custom delay function created for model ITS

Without this function, the ITS would be unable to detect and react to external sounds or objects that come in front of it while navigating through a sharp turn. The variables `rot_r` and `rot_l` hold the values needed for rotations to navigate the curves. The `rot_r` value is slightly larger than the `rot_l` value to enable the ITS to handle sharp turns a bit better. When the two values are set to be the same, the ITS tends to turn more to the left due to sliding effects caused by the lack of sufficient friction between the wheels and the surface.

7 Testing procedures, Results and Outcomes

This section of the report is a record of all tests carried out over the course of the project. It is separated sections that are divided into tests focused on the line tracking, obstacle detection and avoidance, audio unit, integrated design, how we came up with a method to quantify our progress and how we overcame a malfunction of the system at the end of our project.

7.1 Line tracking

7.1.1 Test 1 – Motor Speed

Test Goal

- Keeping in mind that slowing the robot down increases accuracy of line tracking, the purpose of this test is to determine the lowest speed at which the DC motors continue to be able to function and move the robot as desired.
- The program tests 4 directions of movement of the car: Forward, Backward, Left & Right at the set speed.

Procedure

a. ENA and ENB pins of the L298N motor driver control the speeds of the right and left DC motors via PWM. We are using the #define directive to set "CAR_SPEED" to the desired value. This value is sent to ENA and ENB inside functions that move the car. Set car speed to 100 in test code and run it.
b. Test 3 times.
c. Repeat for speeds 120, 140, 160 and 180.

Results

A pass indicates it has successfully moved in all 4 directions. If any 1 direction is not correctly achieved, the trial is counted a failure.

Speed	Outcome (Pass/Fail)
100	Fail
100	Fail
100	Fail
120	Fail
120	Fail
120	Fail
140	Pass
140	Pass
140	Pass
160	Pass
160	Pass
160	Pass
180	Pass
180	Pass
180	Pass

Table 7: Performance of DC Motors running car wheels at various speeds

Discussion

The DC motors are able to function for 140 and above for the values tested. However, at exactly 140, the movement is not smooth, and the car goes rigid after left and right turns with very narrow angles. For 160 and 180 the movement is smooth and can achieve wide angles of turns. Therefore, optimal speed can be said to be around 160.

In retrospect, this experiment was carried out after placing the tracks on a carpet, i.e. a surface with relatively high friction. Results may slightly vary based on the surface being used. However, there is no question of 160 stopping the DC motors from functioning on surfaces with lower friction. There may only be discrepancies about the lower bound for optimal speed.

7.1.2 Test 2 – Implementing a Smooth Line Tracking Procedure

Test Goal

To determine accuracy of Version 1 of line tracking code on both black tracks. The implementation of this version allows the car to accelerate in the case of straight lines and intends for it to detect when it arrives at a turn, make a turn in the correct direction and to continue following the track in this manner. We are using 50% as a benchmark to decide whether an implementation is to be improved upon or changed entirely. A pass indicates that it was able to stay on track and make a complete revolution around the track and return to the starting position. A fail indicates that it was unable to do so.

Procedure

a. Make connections from line tracking module to Arduino Uno as follows:

Line Tracking Module (TCRT5000)	Arduino Uno
+	+5V
-	GND
L	2 (digital)
M	4 (digital)
R	10 (digital)

Table 8: Connections between TCRT5000 and Arduino Uno

b. Load Version 1 code into the Arduino Uno and test on the "Electronics Lab: Short Circuit" track.
c. Repeat 10 times.
d. Repeat for "Silverstone" tracks.

Results

A pass indicates that it was able to stay on track and make a complete revolution around the track and return to the starting position. A fail indicates that it was unable to do so.

Electronics Lab - Short Circuit:

Test No.	Outcome (Pass/Fail)
1	Fail
2	Fail
3	Fail
4	Pass
5	Pass
6	Fail
7	Fail
8	Fail

9	Pass
10	Fail

Table 9: Performance of Version 1 code on Electronics Lab: Short Circuit tracks over 10 trials

Silverstone:

Test No.	Outcome (Pass/Fail)
1	Fail
2	Pass
3	Pass
4	Pass
5	Pass
6	Pass
7	Pass
8	Fail
9	Fail
10	Fail

Table 10: Performance of Version 1 code on Silverstone tracks over 10 trials

Discussion

This implementation has a 30% success rate for the Electronics Lab: Short Circuit track and a high success rate for the Silverstone track (60%) only because it often interprets "The Loop" as a straight line due to the narrow gap. It is unable to precisely track the pattern drawn. Therefore, some of the Silverstone set of results are considered to be false positives.

If we use 50% as a benchmark to decide whether an implementation is improved upon or changed entirely, then this current implementation fails to meet the benchmark. Moving forward, we will be using a different implementation of line tracking code.

7.1.3 Test 3 – Implementing an Oscillating Line Tracking Procedure

Test Goal

To determine the level of accuracy of Version 2 of line tracking code on both black tracks. This version implements a swinging motion using the three IR sensors on the line tracking module as opposed to original acceleration. "Forward" motion is essentially swinging from side to side to stay on the track.
We are using 50% as a benchmark to decide whether an implementation is to be improved upon or changed entirely.

Procedure

a. Make connections from line tracking module to Arduino Uno according to Table 8.
b. For this version, the car is NOT to be placed exactly at the centre at the beginning as the initial input required for the car to begin moving needs to be read from two out of the three sensors. Hence a slight left or right shift at the very beginning is sufficient for the code to work. Load Version 2 code into the Arduino Uno and test on the "Electronics Lab: Short Circuit" track.
c. Repeat 10 times.
d. Repeat for "Silverstone" tracks.

Results

A pass indicates that it was able to stay on track and make a complete revolution around the track and return to the starting position. A “Fail” indicates that it was unable to do so.
Electronics Lab - Short Circuit:

Test No.	Outcome (Pass/Fail)
1	Fail
2	Pass
3	Pass
4	Fail
5	Pass
6	Pass
7	Fail
8	Pass
9	Pass
10	Pass

Table 11: Performance of Version 2 code on Electronics Lab: Short Circuit tracks over 10 trials

Silverstone:

Test No.	Outcome (Pass/Fail)
1	Fail
2	Pass
3	Pass
4	Fail
5	Pass
6	Pass
7	Fail
8	Fail
9	Fail
10	Fail

Table 12: Performance of Version 2 code on Silverstone tracks over 10 trials

Discussion

Success rate of this version has been 70% for the Electronics Lab: Short Circuit track and 40% for the Silverstone track. For the Silverstone track, the primary challenge for this version to overcome has been "The Loop". The robot can follow the rest of the track with higher accuracy. If we use 50% as a benchmark to decide whether an implementation is improved upon or changed entirely, then this current implementation is able to meet the benchmark. Moving forward, we will be improving on this version of the code.

It is also important to note having a single PASS/FAIL criterion for the tests does not allow for a great level of insight into specific aspects of the car's performance. Considering the feedback received on this and previous tests carried out, we will be breaking down each track into its constituent sections when recording results for future tests and collate all pass/fail percentages to form a final success rate.

7.1.4 Test 4 – Analysing Analogue Readings from Line Tracking Module

Test Goal

Take analogue readings from the line tracking module (Middle IR Sensor out of the 3 TCRT5000 IR Sensors) to explore possibility of detecting blue colour with analogue readings implemented into Arduino code, rather than digital logic 0 and logic 1.

Procedure

a. Make connections from line tracking module to Arduino Uno as follows:

Line Tracking Module (TCRT5000)	*Arduino Uno*
+	+5V
-	GND
M	A0

Table 13: Analogue Connection between TCRT5000 and Arduino Uno

b. Load the following snippet of code onto the Arduino Uno.

```
#define LT_M analogRead(A0)

void setup() {
  Serial.begin(9600);      //setting up communication with Serial
Monitor
  pinMode(A0,INPUT);       //Middle IR Sensor
}

void loop() {
//prints to the Serial Monitor
    Serial.print(LT_M);
}
```

c. After opening up the Serial Monitor, hold robot over a white surface for 5 seconds.
d. Continuing to monitor the Serial Monitor, hold robot over a blue surface for 5 seconds.
e. Repeat under a bright white light.

Results

White Surface (Without Additional White Light)

	A	B
1	*Bin*	*Frequency*
2	836	13
3	837	260
4	838	159
5	839	110
6	840	116
7	841	156
8	842	258
9	843	15
10	More	0

Table 14: Histogram bins representing raw data (analogue values) from the Line Tracking Module and frequency of those values over a white surface, without any additional lighting added

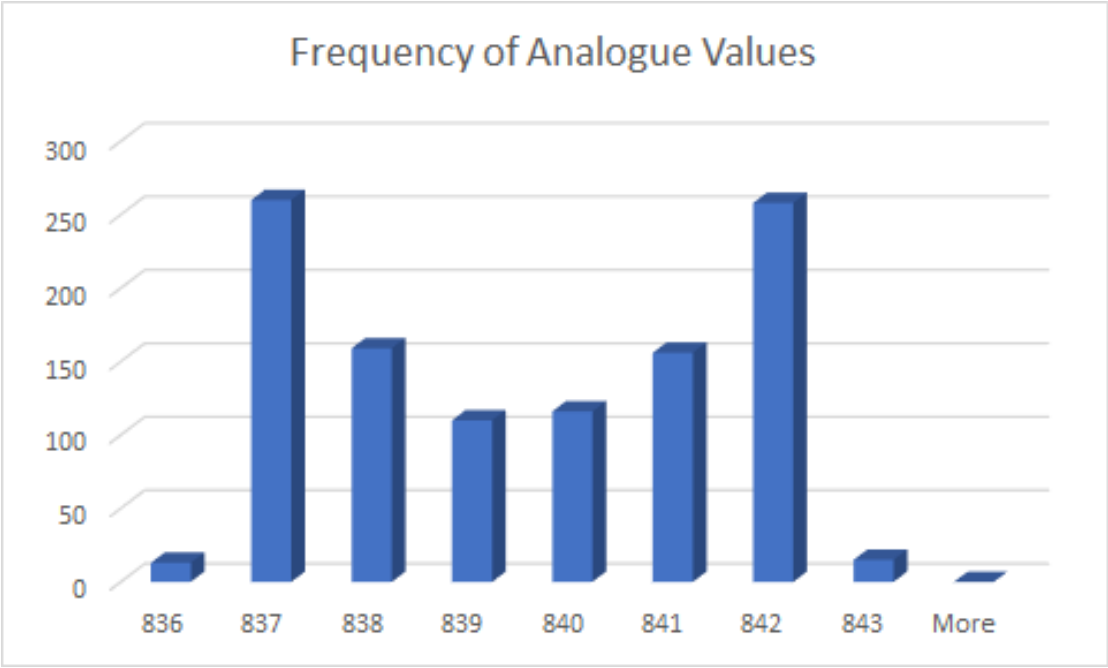


Figure 23: Visual representation of Table 14

Blue Surface (Without Additional White Light)

	A	B
1	*Bin*	*Frequency*
2	840	3
3	841	76
4	842	281
5	843	236
6	844	268
7	845	205
8	846	28
9	More	0

Table 15: Histogram bins representing raw data (analogue values) from the Line Tracking Module and frequency of those values over a blue surface, without any additional lighting added

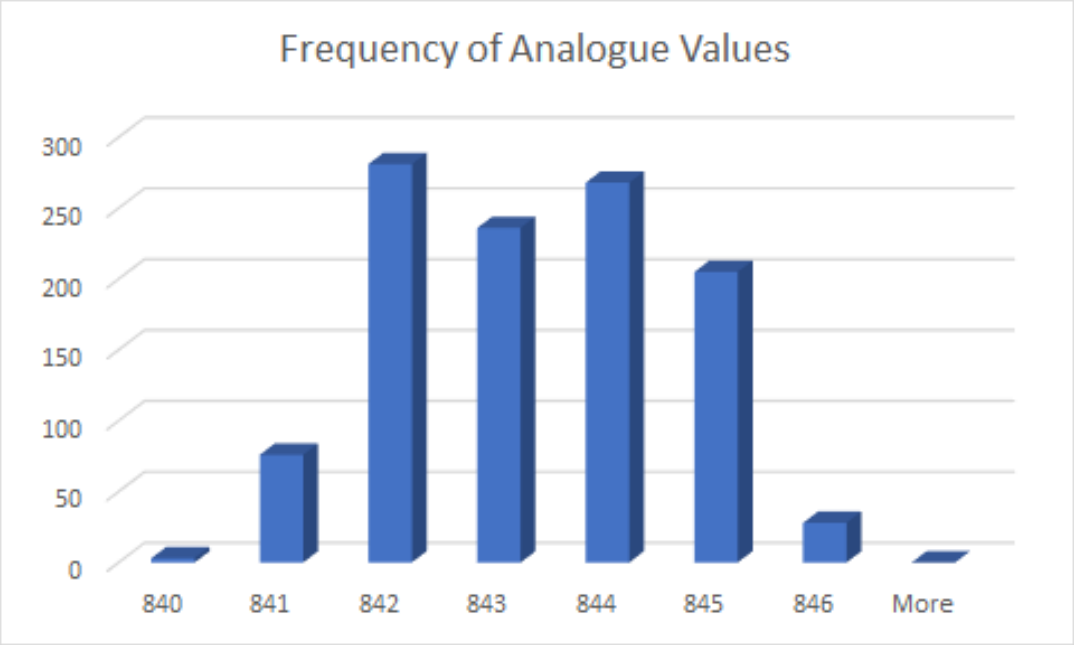


Figure 24: Visual representation of Table 15

White Surface (With Additional White Light)

	A	B
1	*Bin*	*Frequency*
2	788	41
3	789	540
4	790	497
5	More	0

Table 16: Histogram bins representing raw data (analogue values) from the Line Tracking Module and frequency of those values over a white surface, with additional lighting added

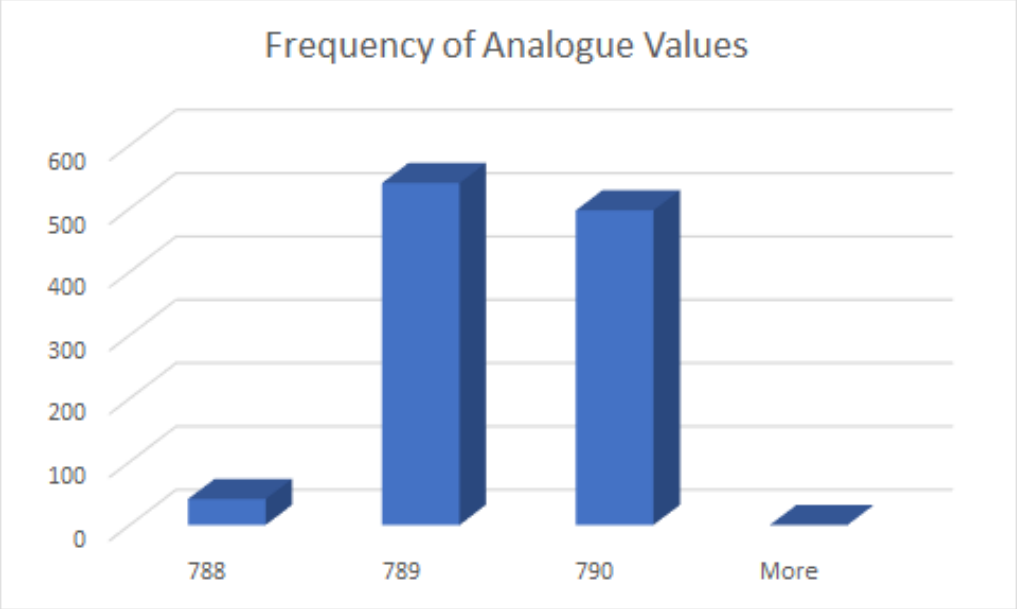


Figure 25: Visual representation of Table 16

Blue Surface (With Additional White Light)

	A	B
1	*Bin*	*Frequency*
2	796	230
3	797	311
4	798	316
5	799	215
6	More	0

Table 17: Histogram bins representing raw data (analogue values) from the Line Tracking Module and frequency of those values over a blue surface, with additional lighting added

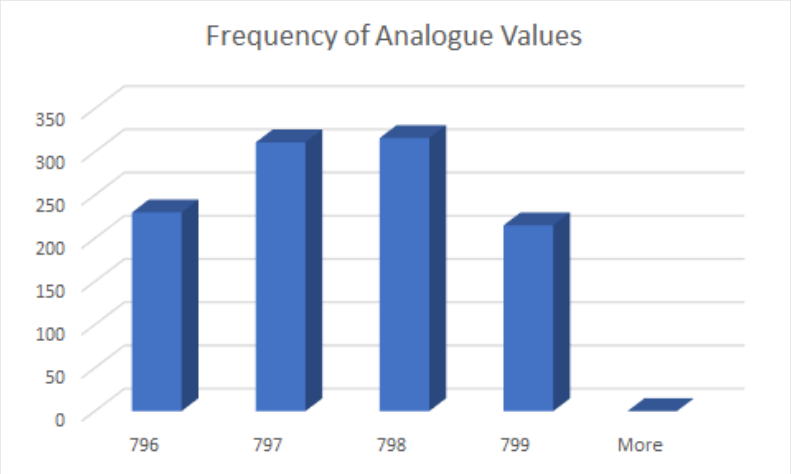


Figure 26: Visual representation of Table 17

Discussion

Experiment conducted under a bright white light produces more consistent readings and a wider margin between readings above a white surface and readings above a blue surface.

However, both margins are very narrow. Keeping in mind that these readings were taken directly above the surfaces, we can conclude that the narrow margin will give rise to discrepancies when readings are taken above, for example, a border between these two surfaces or while the robot is moving. Hence, in order to produce a robust system with higher accuracy, the existing line tracking module will not be used for the blue sets of tracks. We will be looking into purchasing a new colour sensor moving forward in the project. Test Conducted: 08/03/21.

Additionally, as readings are taken at very high rates, it would be more appropriate to use a fixed number of samples rather than a fixed amount of time. This would remove small variations in the number of samples taken each time and ensure consistency when surfaces of different colours are being tested. See Test 5.

7.1.5 Test 5 – Analysing Analogue Readings from New Colour Sensor TCS3200

Test Goal

Take analogue readings from the newly purchased TCS3200 to explore possibility of detecting non-black colours with frequency values implemented into Arduino code.

Procedure

1. Ensure the TCS3200 is connected to Arduino Uno according to the complete schematic on Page 29.
2. Load the test code into Arduino Uno, reading from three frequency channels.
3. After opening up the Serial Monitor, hold robot over a white surface and take 10 samples.
4. Continuing to monitor the Serial Monitor, hold robot over a black surface and take 10 samples.
5. Continuing to monitor the Serial Monitor, hold robot over a blue surface and take 10 samples.
6. Continuing to monitor the Serial Monitor, hold robot over a red surface and take 10 samples.

Results

TCS3200 TEST - White	R	G	B
1	54	103	94
2	52	95	81
3	46	81	72
4	41	75	71
5	41	76	70
6	40	77	71
7	41	78	74
8	43	81	76
9	44	83	76
10	44	81	74

Table 18: Raw data (analogue values) from the TCS3200's Red (R), Green(G), Blue(B) frequency channels for 10 samples taken over a white surface

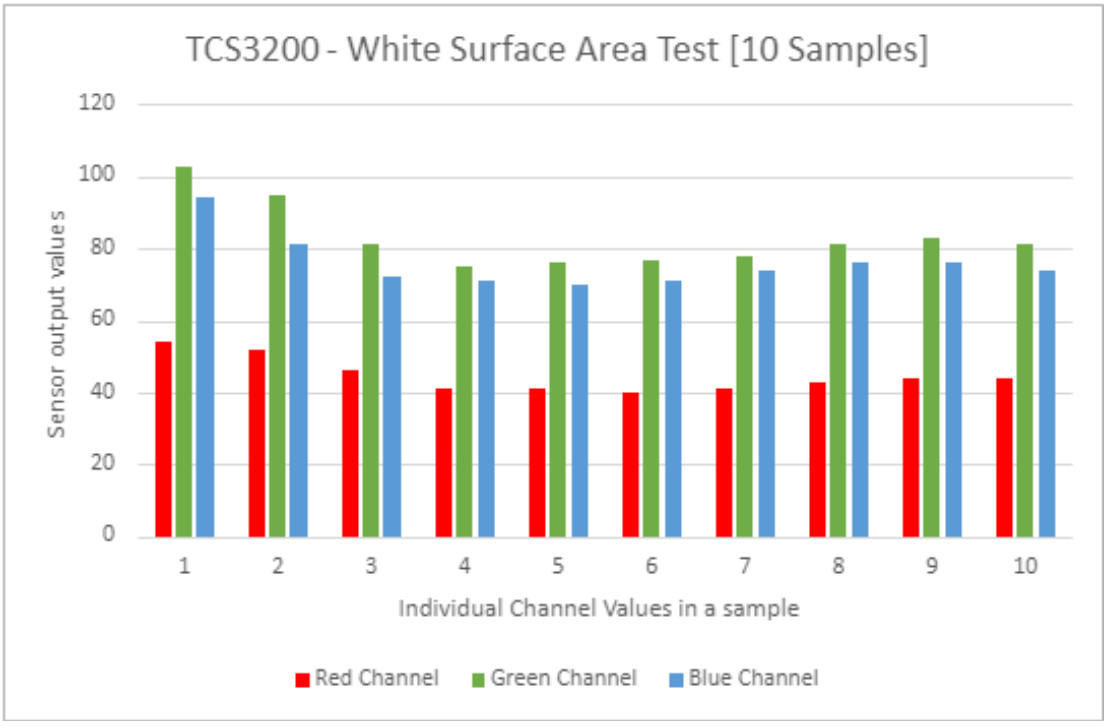


Figure 27: Visual representation of Table 18

TCS3200 TEST - black	R	G	B
1	156	276	219
2	111	186	160
3	85	156	169
4	119	275	298
5	188	386	438
6	263	497	372
7	184	254	201
8	107	175	179
9	111	231	188
10	98	206	213

Table 19: Raw data (analogue values) from the TCS3200's Red (R), Green(G), Blue(B) frequency channels for 10 samples taken over a black surface

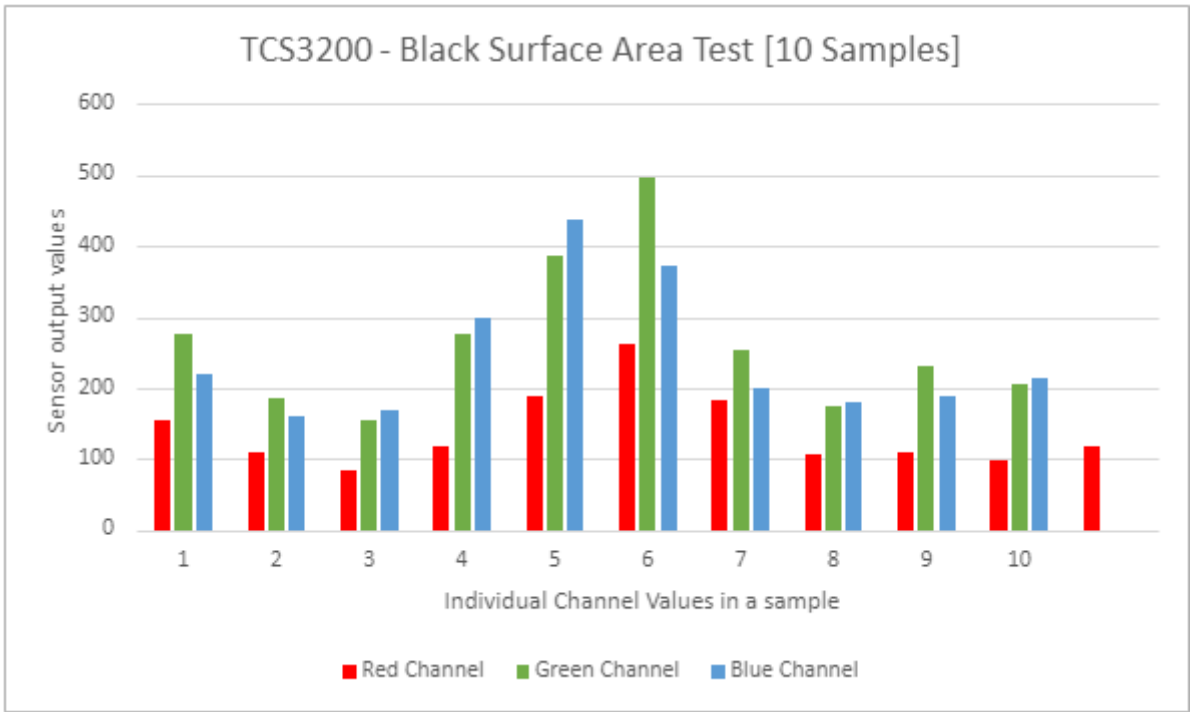


Figure 28: Visual representation of Table 19

TCS3200 TEST - blue	R	G	B
1	36	55	48
2	35	67	59
3	40	68	63
4	40	72	68
5	45	66	54
6	28	63	59
7	32	62	59
8	43	93	55
9	53	91	51
10	41	84	59

Table 20: Raw data (analogue values) from the TCS3200's Red (R), Green(G), Blue(B) frequency channels for 10 samples taken over a blue surface

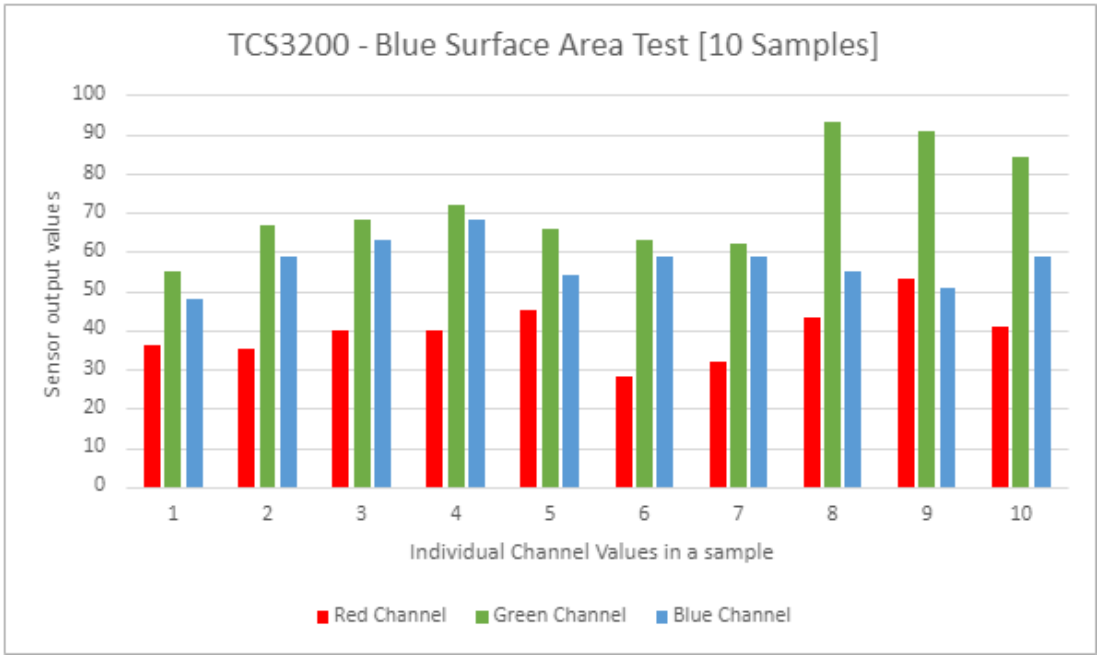


Figure 29: Visual representation of Table 20

TCS3200 TEST - red			
1	29	65	61
2	33	71	79
3	47	124	137
4	72	142	107
5	49	71	54
6	30	60	59
7	35	82	71
8	38	82	78
9	41	85	83
10	46	94	88

Table 21: Raw data (analogue values) from the TCS3200's Red (R), Green(G), Blue(B) frequency channels for 10 samples taken over a red surface

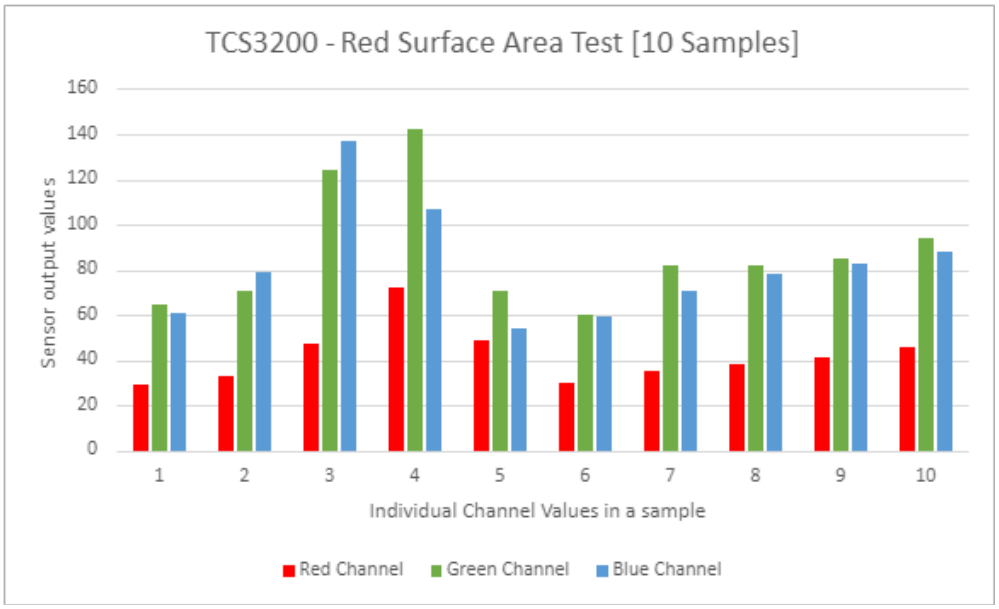


Figure 30: Visual representation of Table 21

Discussion

Looking at the graphs, we are able to draw the conclusion that readings over a surface of a certain colour are consistent and follow a pattern, with rare deviations outside of a narrow range. This is especially visible over graph for white surfaces, where samples 3 through 10 are nearly identical. We are also able to discern significant differences between readings over each colour. Even in cases where one frequency channel may overlap with readings from the same frequency channel of another colour, a combination of three channels ensures there are no identical graphs. This means taking three channels into consideration when differentiating between colours will ensure a robust system and virtually allow our car to detect any colour we want.

Moving forward, we will be incorporating the TCS3200 and frequency values into our line tracking code and eventually, the main program. Test Conducted: 12/03/21.

7.2 Obstacle detection

7.2.1 Test 1 – Determine the angles based on the speed

Test goal: To determine the delay needed to create an angle of 36°,45°,72°,90° at a speed of 160 mm/ms.

Independent variable:

- Delay time in the code

Dependent variable:

- Angle that the car turns

Controlled variables:

- Speed (160 mm/ms)

Procedure:

1. On a plain piece of paper draw a graph with the axis x and y with the origin at (0,0) and mark all the different angles (as specified in the test goal) anti-clockwise.
2. Align the centre of the car with the origin of the graph.
3. Set the car speed at 160 mm/ms.
4. Upload the code and change the delay value until the car reaches each of the angles marked on the paper.
5. Record the values that give the best estimate of the angles.

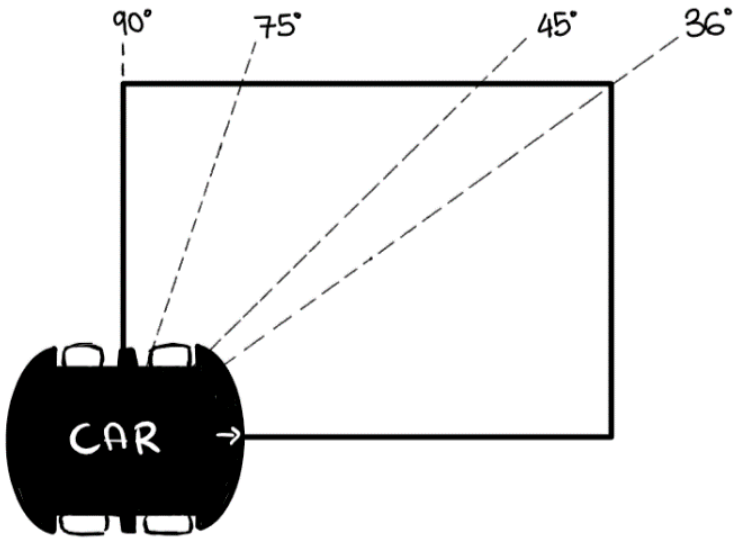


Figure 31: Diagram illustrating the test setup for different angles

Results:

Angles (°)	Delay value
36	320
45	390
90	800
72	600

Table 22: Results of the test illustrating the relationship between angles and delay

Findings:

There is no linear correlation between delay time and the angles. This is due to friction; friction has a significant impact on the tires which means the smallest changes on the surface cause the car to turn at different angles. Therefore, the best approach is to estimate the closest value to the desired angles based on the surface type and then subsequently alter them in small increments to fit the final testing surface.

Additionally, these angles can only be used for a specific scenario as different factor - other than surface type - had to be considered. Such as:

- Speed
- Weight of the car
- Testing surface material softness and uniformity
- Surface inclination
- Consistency of the initial placement of the car on the graph to ensure even readings

7.2.2 Test 2 – Can the car avoid the obstacle on a straight line?

Test goal: To determine the success rate at which the car can detect and avoid an obstacle in a straight-line path

Independent variable:

- Obstacle placement on a straight line

Dependent variable:

- Detection and avoidance of the obstacle

Controlled variables:

- Software program

Procedure:

1. Place the obstacle on a straight line
2. Let the car run the code and check if the car can detect the obstacle, avoid it and return to the straight-line path
3. Run the test 10 times:
   - 5 black tracks
   - 5 blue tracks

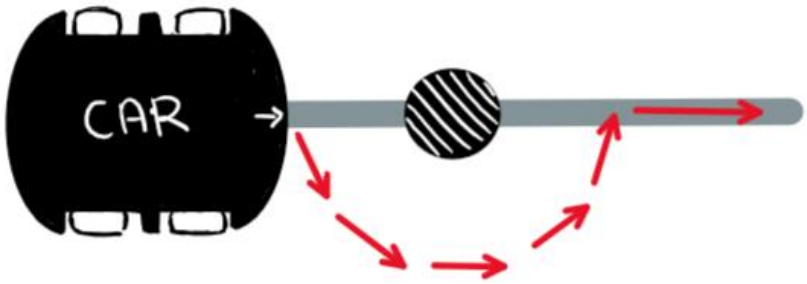


Figure 32: Diagram illustrating the test setup for obstacle avoidance on a straight-line path

Results:

Test number	Track colour	Detects	Avoids	Straight line
1	Blue	1	1	1
2	Blue	1	1	0
3	Blue	1	1	1
4	Blue	1	1	1
5	Blue	1	1	0
6	Black	1	1	1
7	Black	1	1	1
8	Black	0	0	0
9	Black	1	0	0
10	Black	1	1	1

Table 23: Results of the test showing the obstacle avoidance success rate on different colour straight line path

Findings:

The car can detect and avoid an obstacle as well as successfully return to the straight line with an overall success rate of 76% on a straight-line path. Although this is the easiest path the car still struggles to detect the obstacles at times due to the oscillating motion implemented for line tracking which occasionally causes the ultrasonic sensor to miss the obstacle and get too close to the obstacle. This in turn makes obstacle avoidance harder and leads to unsuccessful trials.

7.2.3 Test 3 - Can the car avoid the obstacle on a 45° turn?

Test goal: To determine the success rate at which the car can detect and avoid an obstacle on a 45° turn

Independent variable:

- Obstacle placement on a 45° turn

Dependent variable:

- Detection and avoidance of the obstacle

Controlled variables:

- Software program

Procedure:

1. Place the obstacle on a straight line path
2. Let the car run the code and check if the car can detect the obstacle, avoid it and return to the line at a 45° turn
3. Run the test 10 times:
    - 5 black tracks
    - 5 blue tracks

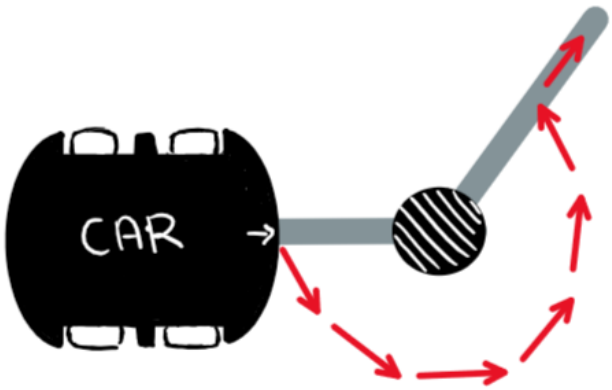


Figure 33: Diagram illustrating the test setup for obstacle avoidance on a 45° turn

Results:

Test number	Track colour	Detects	Avoids	45° turn
1	Blue	1	0	0
2	Blue	1	1	1
3	Blue	1	1	1
4	Blue	1	0	0
5	Blue	1	1	1
6	Black	1	1	1
7	Black	1	1	0
8	Black	0	0	0
9	Black	1	1	1
10	Black	1	1	1

Table 24: Results of the test showing the obstacle avoidance success rate on different colour 45° turns

Findings:

The car can detect and avoid an obstacle as well as successfully return to the line with an overall success rate of 73% on a 45° turn.

7.2.4 Test 4 - Can the car avoid the obstacle on a 90° turn?

Test goal: To determine the success rate at which the car can detect and avoid an obstacle on a 90° turn

Independent variable:

- Obstacle placement on a 90° turn

Dependent variable:

- Detection and avoidance of the obstacle

Controlled variables:

- Software program

Procedure:

1. Place the obstacle on a straight line path
2. Let the car run the code and check if the car can detect the obstacle, avoid it and return to the line at a 90° turn
3. Run the test 10 times:
    - 5 black tracks
    - 5 blue tracks

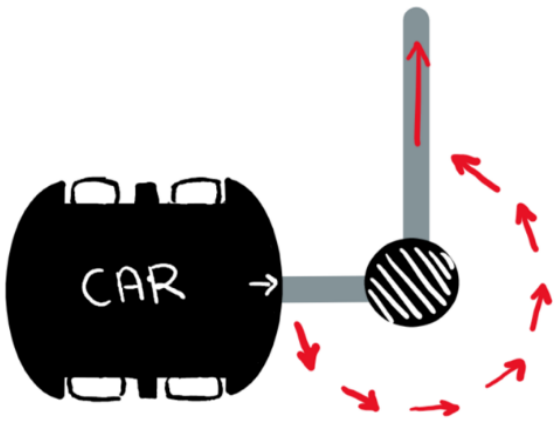


Figure 34: Diagram illustrating the test setup for obstacle avoidance on a 90° turn

Results:

Test number	Track colour	Detects	Avoids	90° turn
1	Blue	1	1	1
2	Blue	1	1	1
3	Blue	1	0	0
4	Blue	1	1	1
5	Blue	1	1	0
6	Black	1	0	0
7	Black	1	1	0
8	Black	1	1	1
9	Black	1	1	1
10	Black	0	0	0

Table 25: Results of the test showing the obstacle avoidance success rate on different colour 90° turns

Findings:

The car can detect and avoid an obstacle as well as successfully return to the line with an overall success rate of 76% on a 90° turn.

7.2.5 Test 5 - Can the car avoid the obstacle on a 135° turn?

Test goal: To determine the success rate at which the car can detect and avoid an obstacle on a 75° turn

Independent variable:

- Obstacle placement on a 135° turn

Dependent variable:

- Detection and avoidance of the obstacle

Controlled variables:

- Software program

Procedure:

1. Place the obstacle on a straight line
2. Let the car run the code and check if the car can detect the obstacle, avoid it and return to the line at a 135° turn
3. Run the test 10 times:
    - 5 black tracks
    - 5 blue tracks

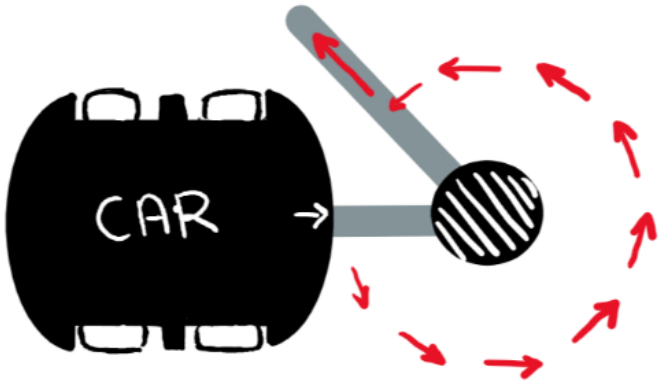


Figure 35: Diagram illustrating the test setup for obstacle avoidance on a 135° turn

Results:

Test number	Track colour	Detects	Avoids	135° turn
1	Blue	1	1	0
2	Blue	1	1	0
3	Blue	1	1	0
4	Blue	0	0	0
5	Blue	1	1	1
6	Black	1	1	1
7	Black	1	1	1
8	Black	1	1	1
9	Black	1	0	0
10	Black	1	0	0

Table 26: Results of the test showing the obstacle avoidance success rate on different colour 135° turns

Findings:

The car can detect and avoid an obstacle as well as successfully return to the line with an overall success rate of 67% on a 135° turn. This shows that as the angle at which the obstacle is placed become larger line detection becomes weaker. This is mainly due to the fact that the roundabout method uses approximations to the angles rather than the exact values, making the decagon shape less consistent.

7.3 *Audio unit*

This section of the report outlines the testing procedures, results and outcomes of all tests carried out when designing and implementing the functionality of the audio unit. Phase one (tests 1-5) of implementing the audio unit was focused on testing how the loudness sensor we bought works and getting the ITS to honk at obstacles when they are detected. The final stage of phase 1 was ensuring that the car waits for 2 seconds after honking at a detected obstacle to give it time to move out of the way before deciding whether to continue with line tracking or obstacle avoidance. Phase two (tests 6-8) focused on the implementation of external audio detection and reaction.

7.3.1 Analogue Output – Signal Analysis

It important to explore a mathematical background of the audio unit circuit in order to better understand testing procedures, as well as their further analysis. In all experimental cases the signal received does have a similar shape[2] with a varying peak voltage as shown in figure 36. For the simplicity of this aspect of the project, only the peak value is concerned in the measurements. However, it is worth exploring the principle of this signal to better predict behaviour of the system. Let's consider two arbitrary circuit examples in figures 37 and 38.

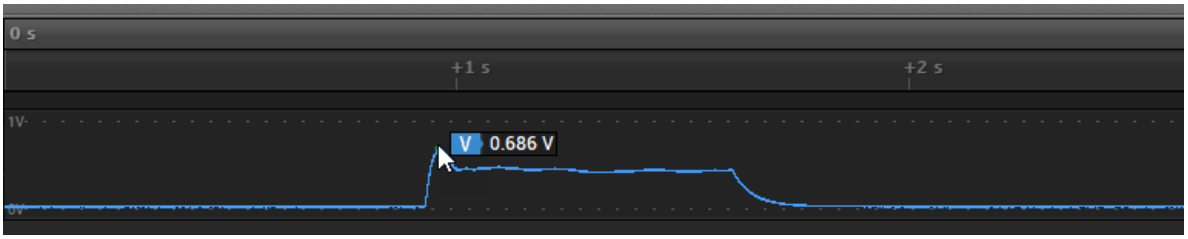


Figure 36: Example output from the ultrasonic sensor

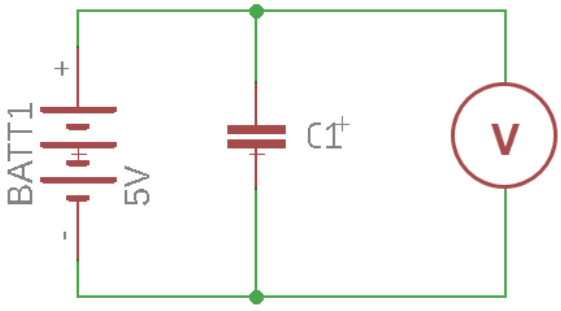


Figure 37: A simple capacitor circuit

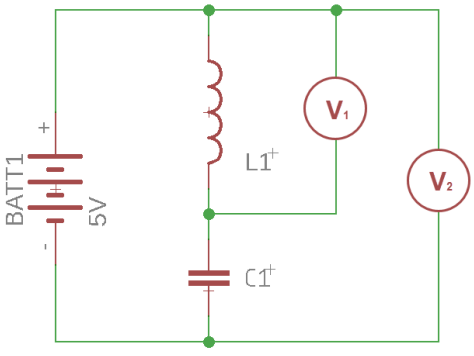


Figure 38: Series LC circuit

[2] Keep in mind that the sound gain, frequency and duration remain constant. This effect is present due to a nature of a LC circuit present in the module, the principle of which will be explained later.

Now, let us consider what will happen with the voltage registered by a voltmeter *V* in figure 37. As the capacitor C1 is discharged, there would be a potential difference between a battery and the capacitor. The current begins to flow, and the voltage increases to a certain point of inflexion. However, after that, when the amount of charge on the capacitor plates increases to a similar value as on the battery, the potential difference decreases to zero – the capacitor is now fully charged and reaches maximum voltage. This process is visualised in Figure 39.

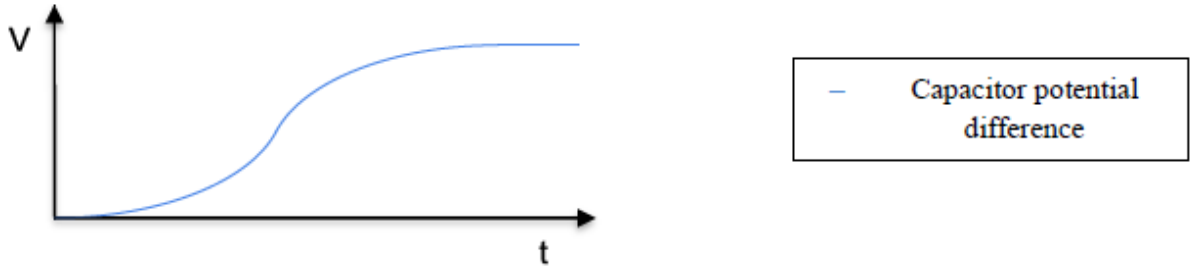


Figure 39: Potential difference between the capacitor C1

A circuit visible in figure 38 should act similarly, except for one major difference. In the second part of the graph above, at the point of inflexion, there is no potential difference between the capacitor and the battery. Therefore, the current should no longer flow. However, when the capacitor was charging, the current flow through a coil L1 which still stores energy in its magnetic B-field. Therefore, an additional potential will cause the magnetic field to decrease and an even greater increase in a capacitor charge. For now, a possible V-t graph presenting this phenomenon is presented in figure 40. A red line shows a predicted value of the potential difference, while the blue one acts as a reference to the previous figure.

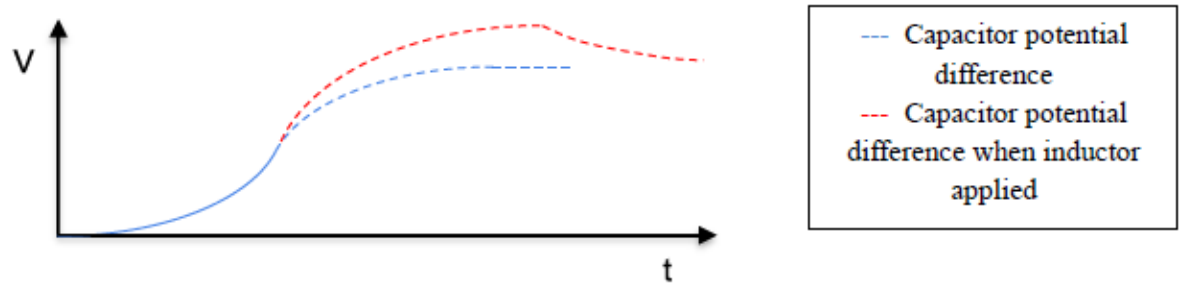


Figure 40: Predicted potential difference in an arbitrary series LC circuit

Signal analysis shown justifies the initial stage of the reading from the beginning of this sub-section, when the sound is initially recognised. The maximum potential difference later analysed and processed by an internal ADC of the ATmega328P. It is to check whether the sound gain exceeded the pre-programmed voltage threshold.

7.3.2 Test 1 – How frequency affects the output voltage of the Grove Loudness Sensor

Test goal: Measure sound sensor responsiveness depending on the input frequency of the source sound.

Independent variable:

- Frequency

Dependent variable:

- Peak voltage

Controlled variables:

- Distance from source: 30cm
- Output device – Suvi's MacBook Air
- Sine wave as output waveform
- Environment (humidity, temperature)

Procedure:

1. Play sound of specific frequency for 2 seconds from output device
2. Record the peak voltage as measured by a logic analyser
3. Change played frequency

Figure 41: Setup of test 1 – How frequency affects the output voltage of the Grove Loudness Sensor

Results:

Input freq. (Hz)	Quiet room level (V)	Peak voltage 1 (V)	Peak voltage 2 (V)	Average peak voltage (V)
500	0.020	0.215	0.345	0.235
1000	0.020	0.116	0.195	0.156
1500	0.020	0.547	0.420	0.484
2000	0.020	0.650	0.453	0.552
2500	0.020	0.718	0.501	0.609
3000	0.020	0.491	0.447	0.469
3500	0.020	0.243	0.231	0.237
4000	0.020	0.347	0.416	0.382

Table 27: Results of first set of tests, testing how frequency affects output voltage of the Grove Loudness Sensor – testing a wider range of frequencies

Input freq. (Hz)	Peak voltage 1 (V)	Peak voltage 2 (V)	Peak voltage 3 (V)	Average peak voltage (V)
2400	0.604	0.762	0.667	0.678
2500	0.642	0.773	0.686	0.700
2600	0.895	0.715	0.738	0.783

Table 28: Results of second set of tests, testing how frequency affects output voltage of the Grove Loudness Sensor – testing a narrower range of frequencies above and below 2500Hz

Findings:

It seems that a frequency of 2600 Hz would be the most appropriate test sound for the loudness sensor. However, this deviates from the datasheet, which indicates that the operating range of the sensor would be between 50-2000 Hz [7]. The test results seem odd due to background research and therefore further testing was required. However, for next few tests, we will be using a 2600 Hz wave for testing purposes.

7.3.3 Test 2 – How distance affects the output voltage of the Grove Loudness Sensor

Test goal: Measure sound sensor responsiveness depending on distance from the input source.

Hypothesis: Increasing the distance between sound source and receiver will decrease the peak voltage outputted by the sensor.

Independent variable:

- Distance from source (cm)

Dependent variable:

- Peak voltage (V)

Controlled variables:

- Frequency: 2600 Hz
- Output device – Suvi's MacBook Air
- Sine wave as output waveform
- Environment (humidity, temperature)

Procedure:

1. Play sound of specific frequency for 2 seconds from output device
2. Record the peak voltage as measured by a logic analyser
3. Take three data points per distance
4. Increase distance

Results:

Distance (cm)	Peak voltage 1 (V)	Peak voltage 2 (V)	Peak voltage 3 (V)	Average peak voltage (V)
30	0.873	0.702	0.687	0.754
100	0.443	0.471	0.278	0.397
200	0.316	0.344	0.457	0.372

Table 29: Results on how distance affects output voltage of the Grove Loudness Sensor

Findings:

These tests results show that, as could be expected, the voltage outputted by the loudness sensor seems to decrease as the distance increases. More data could be acquired, but it is unnecessary for the scope of this project to spend copious amounts of time on this test. It supports the notion that increasing distance from a sound source to the receiver decreases the amplitude at which the sound is received at, as energy is lost during the time of propagation. However, the results do not give a clear trend as to how and running further experiments and analysis in the matter are out of the scope of this project.

7.3.4 Test 3 – How the waveform shape affects the output voltage of the Grove Loudness Sensor

Test goal: Determine how the shape of a waveform affects the output voltage of the Grove Loudness sensor.

Independent variable:

- Shape of waveform

Dependent variable:

- Peak voltage (V)

Controlled variables:

- Frequency – 2600 Hz
- Distance – 30 cm
- Sound source: Suvi's MacBook Air
- Environment (humidity, temperature)

Procedure:

1. Setup is similar to that shown in Figure r. within Test 1 (page num)
2. Play sound of specific frequency for 2 seconds from output device
3. Record the peak voltage as measured by a Saleae™ logic analyser
4. Take three data points per waveform
5. Change waveform

Results:

Waveform	Peak voltage 1 (V)	Peak voltage 2 (V)	Peak voltage 3 (V)	Average (V)
Sine	0.409	0.336	0.485	0.410
Square	0.438	0.411	0.416	0.422
Sawtooth	0.355	0.349	0.467	0.390
Triangle	0.279	0.378	0.265	0.307

Table 30: Results on how the shape of a waveform affects output voltage of the Grove Loudness Sensor

Findings:

This test shows that a square wave produces a greater output voltage at the same distance and loudness than any of the other waveforms. Why this is the case could be further investigated but does not fit into the scope of this project and therefore will not be dived into due to limited time. It could be interesting to research why and how different waveforms affect the functionality of electronics in the future. For the purposes of this project, this test shows that it would be worth using a square wave when testing the functionality of the car.

7.3.5 Test 4 – Testing whether the honking function works

Test goal: Test whether car will honk at an object in front of it when an object is detected in front of it.

Independent variable:

Code uploaded to car kit

Dependent variable:

Honking occurs

Controlled variables:

- Environment (humidity, temperature)
- Curvature of line (straight)
- Hardware in use
- Object used
- Initial distance from object (50cm)

Procedure:

1. Place car at starting mark
2. Turn the car on
3. Observe the car, see whether the buzzer turns on when an obstacle is detected.
4. Repeat 10 times.

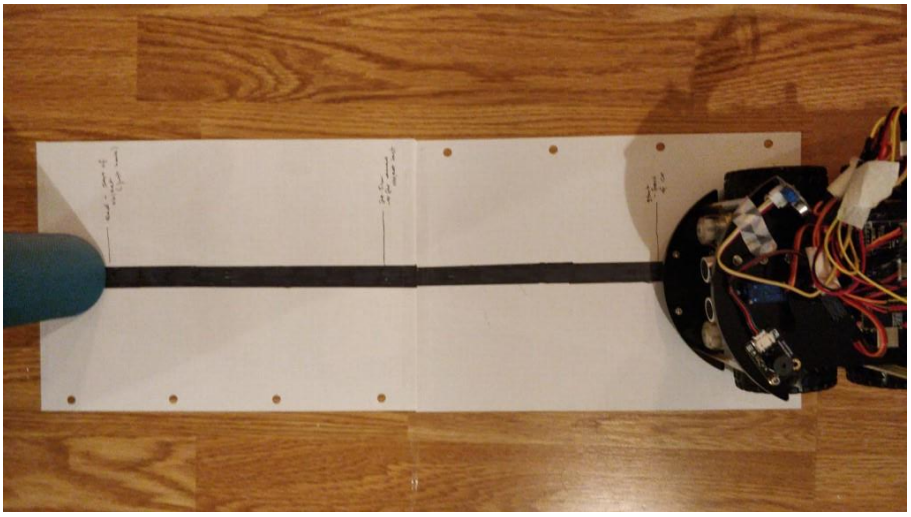


Figure 42: Setup used for tests 4 and 5

Results and findings:

The functionality worked 10/10 times. The program for emitting sound when detecting obstacles is fully functional. The model would move towards the obstacle, stop when it detected the obstacle and emit a sound from the buzzer for half a second. I initially had the buzzer set to output sound for one second but decided to decrease the time for the sake of user experience, as the buzzer is quite loud, and a longer output sound is unnecessary for the sake of this project. Changing the length and creating sequences of sounds outputted by the model would be rather straightforward to implement via software changed if this was desired.

7.3.6 Test 5 – Testing whether waiting after a honk works

Test goal: Test whether car will honk at an object in front of it when an object is detected in front of it, wait for 5 seconds to let the obstacle move out of its way and continue following the line as normal when the obstacle no longer is there.

Independent variable:

- Code uploaded to car kit

Dependent variable:

- Honking occurs

Controlled variables:

- Environment (humidity, temperature)
- Curvature of line (straight)
- Hardware in use
- Object used
- Initial distance from object (50cm)

Procedure:

1. Same setup as shown in Figure f above in test number 4.
2. Place car at starting mark.
3. Turn the car on.
4. Observe the car, see whether the buzzer turns on for 0.5 seconds when an obstacle is detected.
5. Once buzzer turns on, remove the obstacle from the path
6. Observe whether car continues tracking line.
7. Repeat 10 times.

Results:

The experiment was successful 6/10 times on the 7th of March 2021.

Findings:

In the cases, where it didn't work properly, the car stopped, beeped at the obstacle and when the obstacle was removed within 5 sec, the car no longer did anything. This could be due to motors freezing, the surface being unideal for the tired, etc. My hypothesis is that this might be due to issues with the line tracking function or then the structure of the program as a whole, since the car has still had trouble with the line tracking. I tested the line tracking on its own quickly and it was a bit erratic. It is also possible that the battery level could be impacting functionality and therefore it should be ensured that the model is fully charged before next set of tests. Further investigation on how the battery level affects functionality could be interesting and worth looking into.

Further notes: 30th of March 2021

Later tests on the same functionality on the 30th of March yielded a 100% success rate, as the line tracking and integrated design of the whole system had been improved upon. This supports the notion made on the 7th of march that the issues that caused 40% failure in this functionality was due to issues in other parts of the integrate software, as these had now been improved upon.

7.3.7 Test 6 – Determining what threshold is appropriate when implementing phase 2

Test goal: Determine what threshold value would be appropriate for coding the emergency mode trigger. Determine how variations in background noise levels affects what the appropriate threshold value would be.

Independent variable:

- Background noise level

Dependent variable:

- Value outputted to serial – integer number

Controlled variables:

- Environment
- Hardware setup

Procedure:

1. Gather serial data of the sound sensor for 10sec without additional noise
2. Repeat step one but with model ITSs wheels turned on and rotating.
3. Repeat with wheels on and with talking in the background.
4. Create histograms and find min, max and average values for each condition
5. Analyse – these conditions will determine what is the levels that shouldn't be detected as a sound and also determine what is the threshold that the test sound will have to exceed

Results:

The following results visually represent the values gathered from the serial monitor after data processing using Microsoft Excel. The y-axis for each of these graphs represents the number of occasions the specific data value was detected during the 10 second period, while the x-axis indicates what value (or values within a range) were detected.

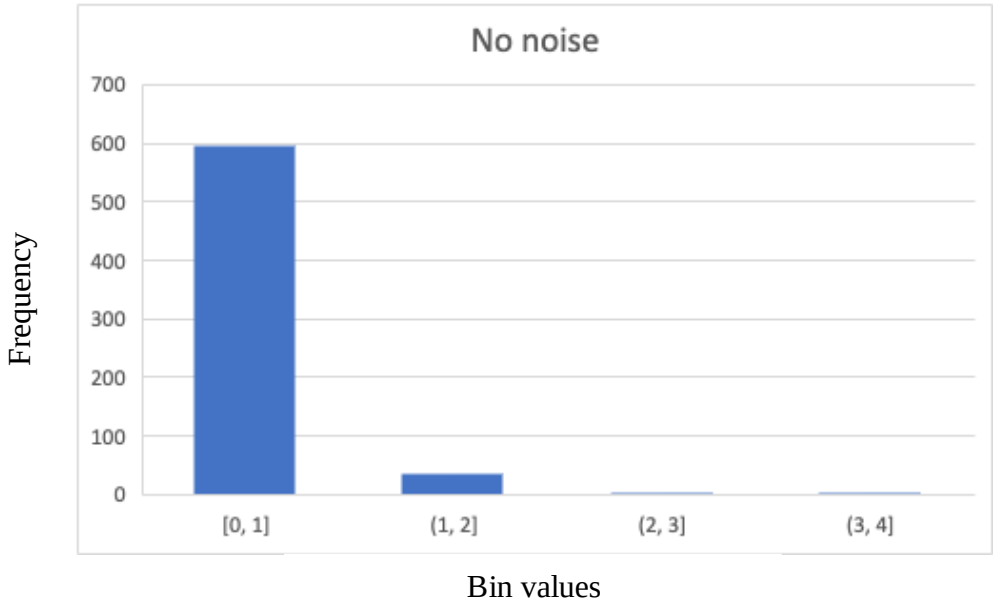


Figure 43: Frequency of different output values of the sound sensor with no external noise

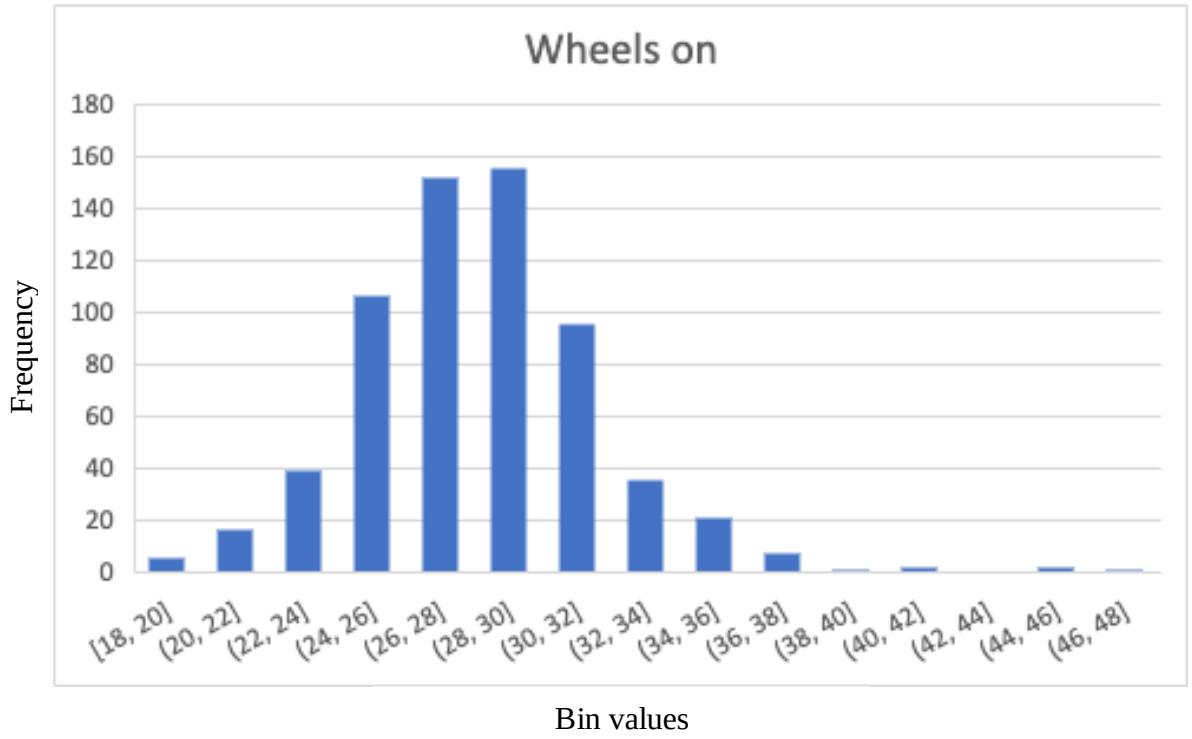


Figure 44: Frequency of different output values of the sound sensor with noise from the wheels

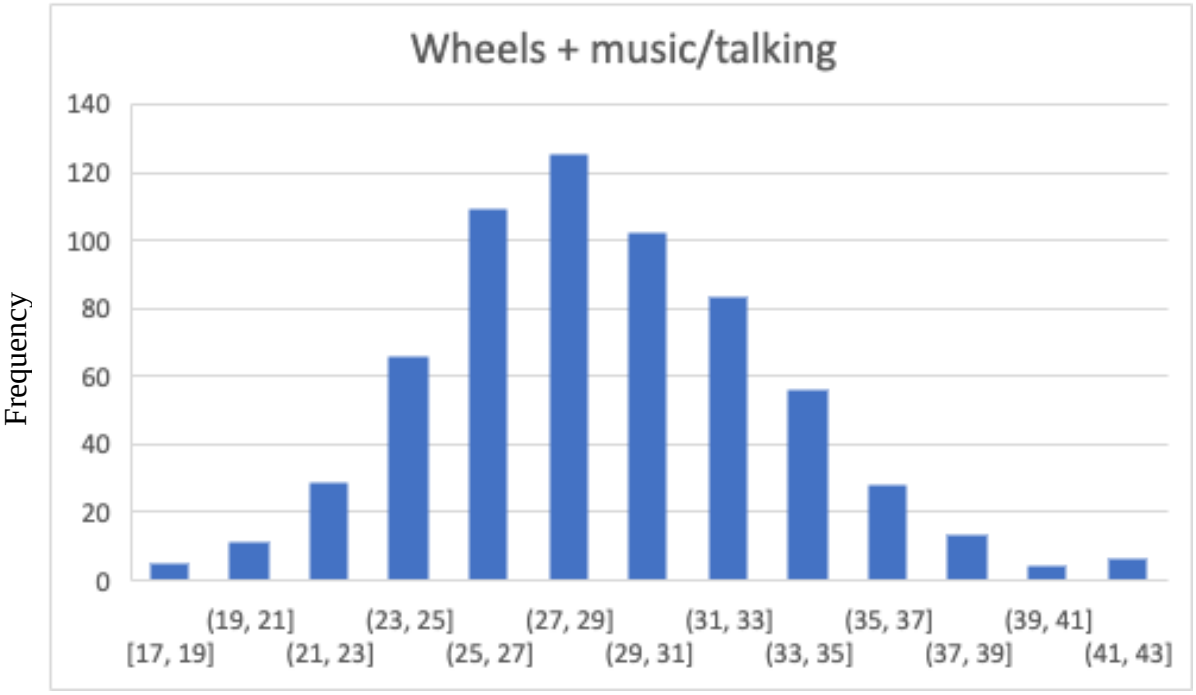


Figure 45: Frequency of different output values of the sound sensor with the wheels turned on and additional background noise

To analyse the data further, the minimum, maximum and average values for each of the three conditions were calculated. These values can be seen in the table below:

Column1	no noise	wheels on	Wheels + talk (music)
min	0	18	17
max	4	48	42
average	0.395604396	28.64364207	29.38461538

Table 31: Minimum, maximum and average serial output values detecting sound in three conditions

Findings:

It can be seen from the results of this experiment that the difference between the minimum and maximum values detected when the wheels are turned on with or without background noise are not significantly different. By looking at the graphs and table above, it can be seen that adding background noise to the condition when the wheels are turned on shifts the average higher and gives higher occurrence of the higher values, but the peak value does not increase due to the additional background noise (in this case music/talking).

According to this experiment having a value of 50 should be a good enough threshold for boundary between normal noise and what the beep will have to exceed to get a reaction. However, it should be noted that this experiment is not taking into account how wheel sound is different due to them running on a floor instead of just the air, since it was not possible to have serial connected and have the model moving simultaneously. Further testing must be conducted to determine what threshold value is appropriate in the real situation the model ITS will be in, where it moves on an actual surface, as we need to ensure that the ITS is able to fully function without reacting to noise it itself creates. Otherwise, the ITS will be stopping more frequently than it needs to, disrupting user experience and the journey.

7.3.8 Test 7 – Extra tests on threshold values when the car is moving on tracks

Test goal: Figure out an appropriate threshold value for the audio unit code.
Hypothesis: As in previous tests, 30 was the average for external background noise conditions and typical peak vales in previous tests were below 50, an assumption is made that values over 50 should suffice as a threshold value.

Independent variable:

- Threshold value in implemented code

Dependent variable:

- Displacement of car in cm

Controlled variables:

- Environment (humidity, temperature)
- Hardware setup
- Track used – straight track with white background and black line with total length of 85cm (custom made, similar to that seen in figure xxx in Test 4, but extended)

Procedure:

1. Place the model ITS on a straight black track with total length 85cm to see how far it moved for each threshold value.
2. Take eight measures of displacement per threshold value condition. Increase threshold value with an increment of 10, starting from 30 and ending at 100.

Results:

Threshold	disp. 1	disp. 2	disp. 3	disp. 4	disp. 5	disp. 6	disp. 7	disp. 8	Average
30	4.2	1.8	1.1	1.2	1.1	3.5	0.7	2.7	2.0375
40	1.6	5.3	1.7	0.5	5.2	0.3	2.6	0.5	2.2125
50	0.9	0.7	3	10.4	0.5	14.2	1.7	35.5	8.3625
60	56	85	2.7	12.2	22	31	36.4	42.1	35.925
70	52.5	65.9	71.5	85	85	68	85	85	74.7375
80	85	85	85	63.5	85	85	40	5.6	66.7625
90	85	85	85	85	85	85	85	85	85
100	85	85	85	85	85	85	85	85	85

Table 32: Displacement (in cm) of the model ITS in each threshold value condition, 8 sets of data per condition

Findings:

The maximum possible displacement is 85, but it marks times that the ITS completed the test track and started going past it as well. The hypothesis was partially correct, values of over 50 were required, but the value required for full functionality was higher than expected. Results show that a value over 80 is required. Therefore, a value of 85 will be used. This experiment was a better indicator for determining an appropriate threshold value, as it took into account the actual noise created when the car moves on a surface.

7.3.9 Test 8 – Determining the best way to test audio input detection

Test goal: Determine the best test sound to use when wanting to trigger audio unit functionality.

Hypothesis: Based on previous tests, the 2600Hz square wave should be a rather good test sound. However, will compare it with other sound sources that would be more convenient to create when testing, as playing the test sound would require an external sound source to be set up.

Independent variable:

- Sound used – sound detected by the loudness sensor

Dependent variable:

- Whether or not the car reacts to it

Controlled variables:

- Environment
- Code used – threshold value is set as 85 so that sensor doesn't detect sound from car moving as danger
- 0 means no reaction, 1 means reaction happened (stopped for 5 sec)

Procedure:

- Let car move on a straight track, not paying attention to the other functions currently
- Different sounds for the car to react to:
    1. A clap
    2. Knock the floor
    3. The 2600Hz square wave test sound played off of a phone
    4. 500Hz sound played on max volume off a computer
    5. Hitting two hard objects together
- The second part of this experiment took the two best test sounds and conducted more tests on them to determine which one would be the most appropriate for testing purposes.

Results Part 1

Test number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**Success (%)**
Clap	0	0	0	0	1	1	1	1	0	1	0	0	0	38.46
Knock	0	0	1	1	1	1	1	1	1	1	0	1	1	76.92
2600Hz square wave played from phone	0	0	0	0	0	1	0	0	0	0	0	0	0	7.69
500Hz square wave played at max volume off computer	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00
Hitting two hard objects together – hairbrush and elephant decoration	0	0	1	0	0	1	0	1	0	1	0	0	0	30.76

Table 33: Pass/fail and success rate of testing of each test sound type

Findings: Part 1

Knocking on the floor seems to be the best option currently to test the sound. Unfortunately, we do not have an actual car horn (or any kind of horn) to test with to have a more real-life applicable experiment. Knocking gives most consistent results, which seems odd considering the preliminary tests that we did would have indicated that a 2600Hz square wave would cause the best results.

Surprisingly, the 2600Hz square wave did not yield good results. It only worked one time, which was unexpected, especially since the readings from the loudness sensor had originally been rather high. However, the values recorded in the first 3 tests did not see what integer values were outputted from the sensor nor did they test the values outputted when detecting a clap or knock on the floor, so the hypothesis was constructed with limited background knowledge. Further testing into which one of the two best test sounds to use was conducted in part 1 was carried out in part 2.

Results Part 2

Test number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**Success (%)**
Clap	1	1	0	1	1	1	1	0	1	0	0	1	1	1	71.42
Knock	0	0	1	1	1	1	1	0	0	0	1	0	1	0	50.00

Table 34: Pass/fail and success rate of testing of clap VS knock sound type

Findings: Part 2

Clapping seems to be more consistent with results now. Therefore, use that for testing purposes, if there is extra time later on in project, this can be investigated further. However, for now we must focus on the integration of the line tracking and obstacle avoidance codes, so will leave this for the time being. A benefit of both sounds is that they are easy to generate and especially the clap is something that is easy to create and works very well for the scope of this project. However, for any future implementations, a better system should be developed.

For future audio units, would be better to have a sensor that differentiates between different frequencies, this way, can analyse the frequencies of things in the background noise, and code in a specific frequency that if goes above a threshold value, then the car would react – like a predetermined frequency for all car horns, which could be standardised.

7.4 Integrated design

Outcome of the integrated design testing has been consolidated and tabulated below using MS Excel™. For each section of a track a colour indication shows whether the ITS was capable of progressing. These tests were carried out using the integrated code rather than separated functions, as previous tests had been. Graphical representation and its further analysis can be found in the following sections of this document.

7.4.1 Test 1 – Full tracks testing with integrated design

Test goal: To determine whether version 9 of the code for the integrated design yields a successful outcome.

Independent variable:

- N/A

Dependent variable:

- N/A

Controlled variables:

- Environment
- Hardware setup

Procedure:

1. Place the car at the starting point of the track.
2. Run the system.
3. Monitor the behaviour and note which line tracking attempts were unsuccessful for a particular segment.
4. Repeat the test at least ten times (five times per colour).
5. Tabulate the results.

Figure 46: Electronics Lab Short Circuit track

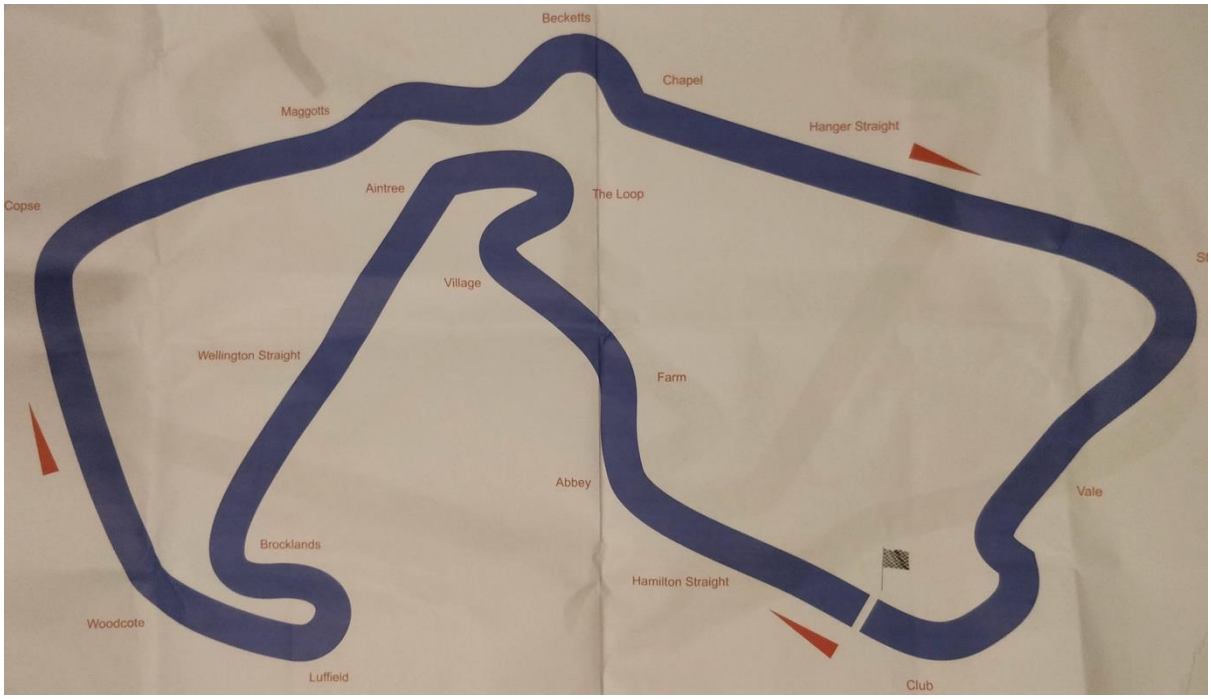


Figure 47: Silverstone track

TEST NUMBER	TEST *TRACK*	Arduino straight	Current spi*ke*	Pot *turn*	Inductor loop	Voltage dr*op*	Thermistor turn	Diode lane	Transistor bend	Signal peak	Pi *curve*	% w*orking*
1	EE Lab Blue	1	1	1	1	1	1	1	1	1	1	100
2	EE Lab Blue	1	1	1	1	1	1	1	1	1	1	100
3	EE Lab Blue	1	1	1	1	1	1	1	1	1	1	100
4	EE Lab Blue	1	0	0	1	1	1	1	1	1	0	70
5	EE Lab Blue	1	0	0	1	1	1	1	1	1	0	70
6	EE Lab Black	1	1	0	1	1	0	1	1	1	1	80
7	EE Lab Black	1	1	1	1	1	1	1	1	1	1	100
8	EE Lab Black	1	0	1	0	0	0	1	1	1	1	60
9	EE Lab Black	1	0	1	1	0	1	1	1	1	1	80
10	EE Lab Black	1	0	1	1	0	1	1	1	0	1	70

Table 35: Electronics Lab track testing results

TEST NUMBER	TEST TRACK	Abbey	The Loop	Aintree	Wellington	Luffield	Copse	Becketts	Hanger	Stowe	Club	% working
1	Silverstone Black	1	1	0	1	1	1	1	1	0	1	80
2	Silverstone Black	1	1	0	1	1	1	1	1	0	0	70
3	Silverstone Black	1	1	0	1	1	0	1	1	1	0	70
4	Silverstone Black	1	1	1	1	0	0	1	1	0	0	60
5	Silverstone Black	1	1	0	1	1	0	1	1	0	1	70
6	Silverstone Blue	1	1	0	1	1	1	1	1	1	1	90
7	Silverstone Blue	1	1	1	1	0	0	1	1	1	0	70
8	Silverstone Blue	1	0	0	1	1	1	1	1	1	1	80
9	Silverstone Blue	1	0	1	1	1	0	1	1	1	1	80
10	Silverstone Blue	1	0	0	1	1	1	1	1	1	1	80

Table 36: Silverstone track testing results

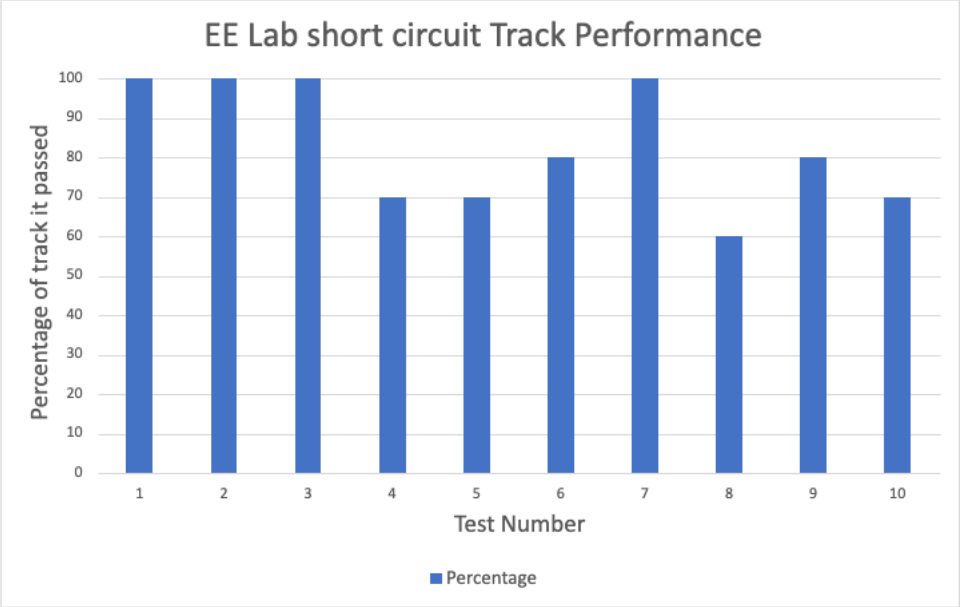


Figure 48: EE Lab – Track Performance

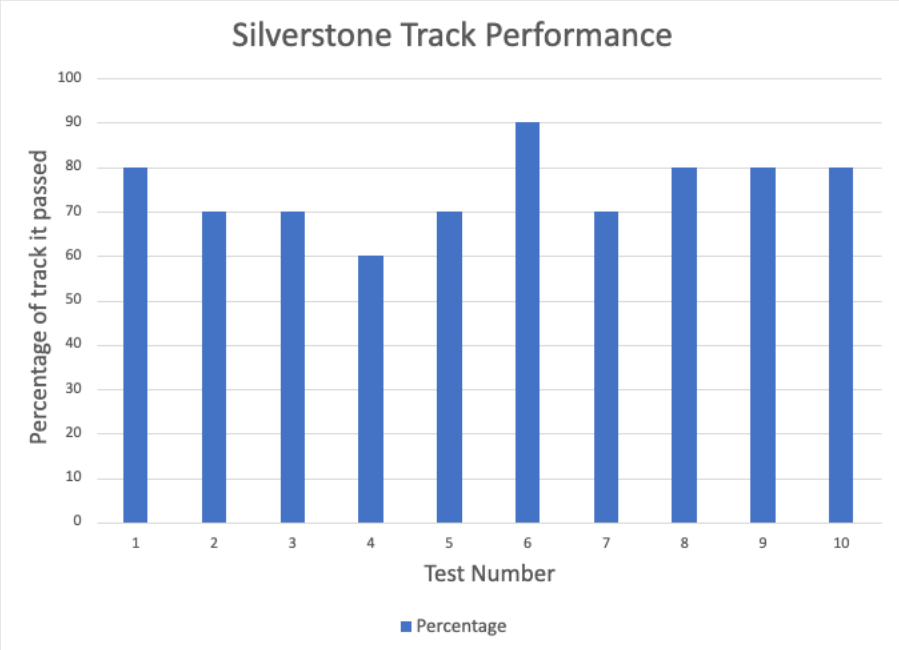


Figure 49: Silverstone – Track Performance

As visible, a significant number of successful attempts have been achieved. Note that in all of the cases the probability of success is not determined by the colour of the track, but the sharpness of the turn. Hence, the current spike for example has a substantially lower passing rate in respect to the others.

The final track we conducted testing on was a custom-made red track, so we could confirm that the colour of the track would not matter for our ITS to function as intended. Figure XXX below shows the custom red track with labels of each section that was assessed when determining the percentage of functionality of the ITS. We incorporated different kinds of curves for the red track to ensure that multiple cases were considered with the red tracks, similar as to the cases in the EE Lab and Silverstone tracks. The percentage of success for each run was determined by whether the ITS passed each of the sections of the track. 1 indicates a successful run of the section while 0 indicates a failure.

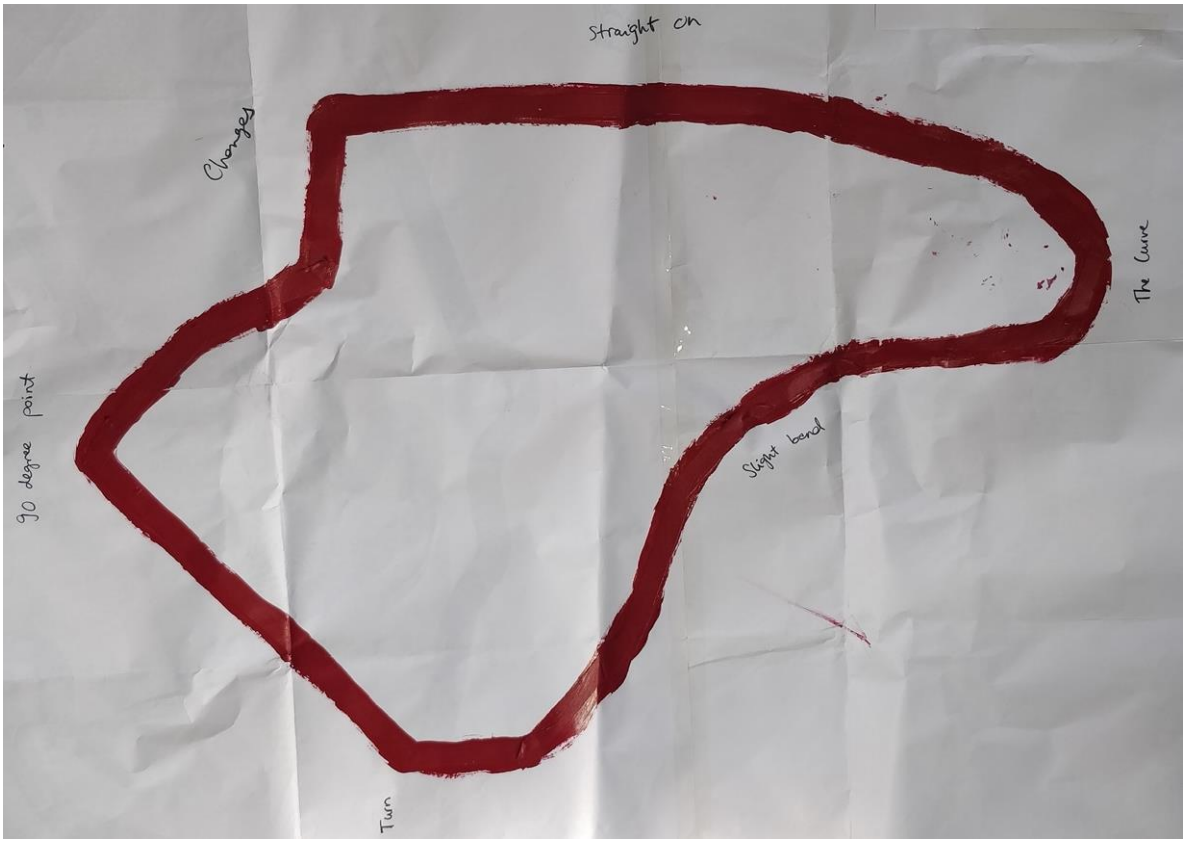


Figure 50: Custom-made "Texas Instruments"™ red track made for testing purposes

TEST NUMBER	Straight on	Changes	90-degree point	Turn	Slight bend	The curve	% working
1	1	1	1	1	1	1	100
2	1	1	1	1	1	1	100
3	1	1	1	1	1	0	83.3
4	1	1	1	1	1	1	100
5	1	1	1	1	1	1	100
6	1	1	1	1	1	1	100
7	1	1	1	1	1	1	100
8	1	1	0	1	1	1	83.3
9	1	1	1	1	1	1	100
10	1	0	1	1	1	1	83.3

Table 37: Custom red track testing results from 30.3.2021

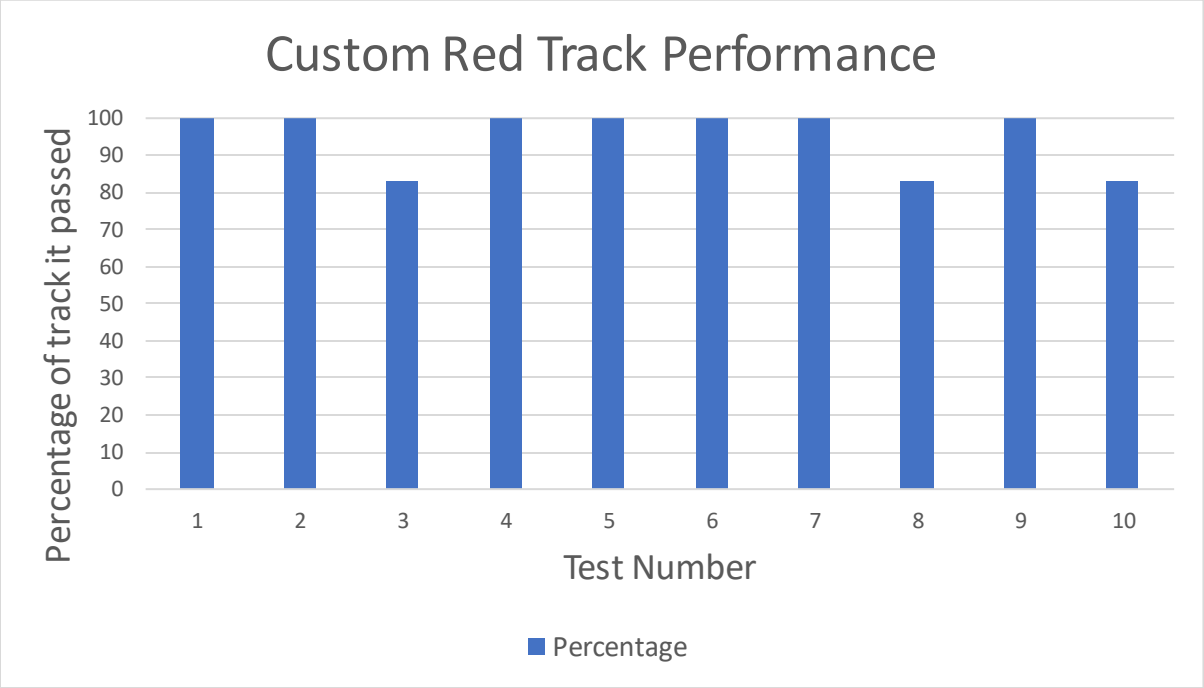


Figure 51: Custom Red track Performance from 30.3.2021

At the final stage of our project, we achieved an average success rate of 95% for the custom colour track. The three tests when the car was unable to fully pass through the track was due to sliding effect due to the lack of friction between the wheels and the surface, which cause the ITS to veer off tracks.

7.5 Total Progress

A crucial step in the project was figuring out a way to quantify the progress we had. Simply having a pass-fail system for whether the cat could get through tracks did not seem a sufficient enough way to determine how functional the car was in respect to our aims. We wanted to develop a way to keep track of the progress we made and how each step we took was affecting how well we were fulfilling the project aims as a whole. We came up with a system where each type of functionality had its own weight and within it, different conditions that we wanted the model to be able to handle had a weight. The way this was weighted is as below:

- Line detection: 35% of the project functionality as a whole
    - 5 tracks – weight is 20% of line tracking functionality per track
        - 2 Black tracks (EE Lab Short Circuit and Silverstone)
        - 2 Blue tracks (EE Lab Short Circuit and Silverstone)
        - 1 red track (custom made)

- Obstacle detection and avoidance 35% of the project functionality as a whole
    - 6 different things considered – 16.6% of line obstacle detection and avoidance functionality each
        - Returns to line when the object is on a straight line (180 case)
        - Returns to line when the object is on a slight turn (225 case)
        - Returns to line when the object is on a 90-degree turn (270 case)
        - Returns to line when the object is on a sharp turn (315 case)

- Audio unit 30% of the project functionality as a whole
    - 4 different things considered – 25% of audio unit functionality each
        - Honks at an object in front of it
        - Gives time for object to move, if it has moved, it continues along line
        - Detect and reacts to one honk – stops for 5 seconds
        - Can detect multiple honks and react to it

The following tables show our progress throughout the project in each one of the three categories and in total.

	EE blue	EE black	SS Blue	SS Black	Custom	Total %
Week 1	0	0	0	0	0	0
Week 2	0	0	0	0	0	0
Week 3	0	0	0	0	0	0
Week 4	0	0	0	0	0	0
Week 5	0	70	40	40	0	30
Week 6	0	70	40	40	0	30
Week 7	70	70	40	40	70	58
Week 8	83	83	75	75	83	79.8
Week 9	83	83	75	75	83	79.8
Week 10	83	83	75	75	95	81.2

Table 38: Line tracking progress

	Detects	Avoids	180 case	225 case	270 case	315 case	Total %
Week 1	0	0	0	0	0	0	0
Week 2	0	0	0	0	0	0	0
Week 3	90	0	60	0	0	0	25
Week 4	90	0	60	0	0	0	25
Week 5	90	0	60	0	0	0	25
Week 6	90	0	60	0	0	0	25
Week 7	90	60	60	60	0	0	45
Week 8	90	70	60	60	50	40	61.7
Week 9	90	70	60	60	50	40	61.7
Week 10	90	70	60	60	50	40	61.7

Table 39: Obstacle detection progress

	Emit honk	Wait for object	Detect 1	Detect many	Total %
Week 1	0	0	0	0	0
Week 2	0	0	0	0	0
Week 3	0	0	0	0	0
Week 4	0	0	0	0	0
Week 5	0	0	0	0	0
Week 6	100	100	0	0	50
Week 7	100	100	0	0	50
Week 8	100	100	100	0	75
Week 9	100	100	100	0	75
Week 10	100	100	100	0	75

Table 40: Audio Unit progress

	Total functionality %
Week 1	0
Week 2	0
Week 3	8.33
Week 4	8.33
Week 5	18.33
Week 6	35
Week 7	51
Week 8	72.16
Week 9	72.16
Week 10	72.62

Table 41: Total functionality progress

The graph below takes is a visual representation of the data from the tables above and provides a concise and easily interpretable description of our progress throughout the weeks.

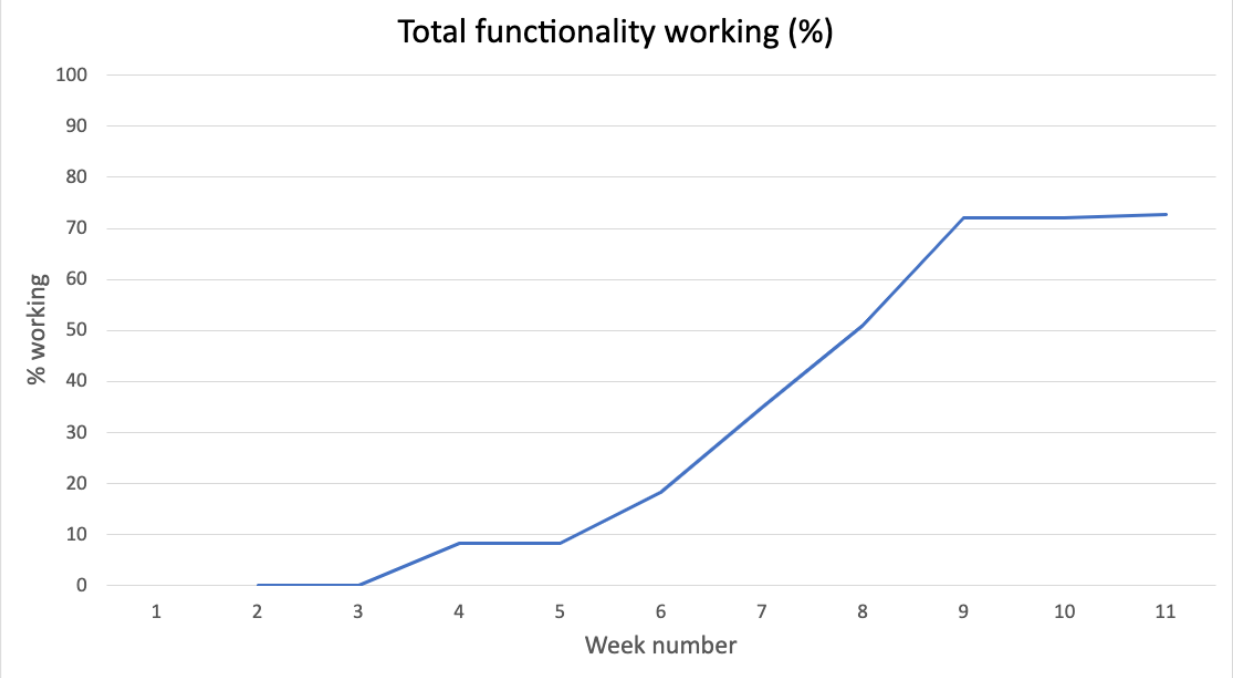


Figure 52: Total achieved functionality throughout the weeks

7.6 The sudden malfunction on 4th of April

In week 9 we had achieved a functioning integration of the software for the model ITS. We proceeded by conducting extensive tests to gather more quantitative data on the percentage of functionality of the model ITS achieved. After a day of testing without issue, the model ITS was placed to the side. The following day when testing was continued, the car was no longer working as intended. It no longer was able to follow the line and had trouble moving forward. The colour testing in the beginning of the program to configure to the track the ITS was placed on no longer worked. The group convened for a day on the 5th of April to troubleshoot and figure out what the issue was.

We noticed that for some reason, one time when the bottom of the ITS was knocked on, the car functioned without issue for about one minute before the issues resurfacing. We tried to explain this behaviour and concluded most likely we were dealing with a hardware issue. This seemed reasonable, since the software that had been used had not had any issues and had run multiple consecutive working tests. We started the debugging process by analysing what things could be causing the issue. After this we proceeded to eliminate possible issues one at a time.

Possible reasons for malfunction for which testing was carried out:

- Faulty wires
- Faulty components or short circuits
    - Motors
    - Loudness sensor
    - Arduino Uno board
    - TCS3200 colour sensor
- Software issues

7.6.1 Test 1: Wiring and Connections

Test goal: Determine whether the issue is due to faulty wiring.
Hypothesis: The malfunction is due to broken wires in the model.

Procedure:

1. Remove wire from its position.
2. Test wire connectivity using continuity test.
3. Repeat for all wires in the circuit.
4. Attach wire back to the circuit.
5. Test the connection between the pins of a sensor and pins of the Arduino Uno to ensure that a connection is established between the two entities.

Findings:

We began by testing all the wires of the model ITS using a continuity test. We found a couple wires that had slight connectivity issues. This can be expected when using jumper cables, as they are less robust than other wire types. We replaced wires that did not meet our standards with new ones that were tested for full conductivity before use. After the wiring had been extensively checked, the same issues remained.

7.6.2 Test 2: Motors

Test goal: Determine whether the issue is due to the motors malfunctioning.
Hypothesis: The malfunction is due to the motors malfunctioning, which would cause the ITS to move in a stunted manner.

Procedure:

1. Upload a program to the model that is meant to run the wheels in different directions.
2. Hold the car in hand and turn on.
3. Observe whether the wheels turn in both directions as intended.

Findings:

The motors work as intended. The issue is not related to motors.

7.6.3 Test 3: Loudness Sensor

Test goal: Determine whether the issue is due to issues with the loudness sensor.
Hypothesis: A short circuit could be caused by the lack of insulation between the loudness sensor and the metal rod it is connected to, which would cause the malfunction. (This is supported by the fact that in prior testing over the weeks, when the ITS has struggled to function before, adjusting the position of the loudness sensor affected how well the model functioned.)

Procedure:

1. Add insulation between the loudness sensor and the metal rod.
2. Test whether the model ITS works as initially intended.

Findings:

Adding insulation did not affect functionality. The possibility of it being a connectivity issue with the loudness sensor (wiring) had been eliminated in the first stage of debugging (Test 1).

7.6.4 Test 4: Arduino Uno

Test goal: Determine whether the issue is due to issues in the Arduino Uno board.
Hypothesis: Damage to the Arduino Uno board is causing the malfunction.

Procedure:

1. Examine the Arduino Uno board for any external damage or damage caused by short circuits.

Findings:

No damage visible on the Arduino Uno board. Replacing the Arduino Uno board with another one (from a car kit that had been kept as a backup for "spare" parts) did not affect results either.

7.6.5 Test 5: TCS3200 colour sensor

Test goal: Determine whether the issue is due to issues with the TCS3200 colour sensor.
Hypothesis: Damage to the TSC3200 is causing the malfunction.

Procedure:

1. Rerun test program for colour sensor as described by "Test 5 – Analysing Analogue Readings from New Colour Sensor TCS3200".
2. Analyse values read by sensor.
3. Analyse functionality of colour sensor using a logic analyser.

Findings:

Sensor seems to be working as intended. However, for some reason the tests we carried out when debugging showed values from different colour surfaces that were much lower than the preliminary tests we had carried out on the sensor. The values were at much lower ranges than previously. Especially the blue channel values were much lower and were often very similar to those attained on a white surface, regardless of what colour surface the sensor was placed on. We could get the car to work somewhat as intended by hardcoding the values for desired colours, but this was a step back from the functionality we had had before and therefore we continued debugging.

After closer inspection of the ITS, we finally found that the colour sensor had slightly tilted in a way that affected the readings it took from the surface the car was on. We secured the sensor so that it would remain parallel to the surface. We also found that by blocking two of the LEDs on the colour sensor, the output values of the TCS3200 had better margins of difference between colours, allowing for the configuration of the car to its surroundings to work once again. We were very happy to have recovered full functionality of the car.

8 Evaluation and Discussion

Within the 10 weeks of active work time that we had for designing and building our model of an ITS, we managed to achieve most of our aims. Our model can track any colour line, avoid obstacles and successfully return to a line in most cases, output sound to communicate intent with its surroundings and detect audio signals and react in an appropriate manner. We have successfully fulfilled most of the user requirements. A few of these are not fully functional, as the model is unable to return to a line after avoiding an obstacle in all cases at all times. This is the only fundamental user requirement we failed to meet completely but have met partially. We met all the supplementary user requirements, although these could still be further expanded upon, as will be discussed in later sections.

We had foreseen some of the constraints and challenges we faced along the way and were prepared to deal with them accordingly. The sudden malfunction we faced on the 4th of April 2021 was a situation that we had considered possible in our risk register, but not highly probable. However, as we had made our risk register during the beginning of the project, we were mentally prepared for when it happened and knew how to move forward. We applied step by step analysis and debugging to eliminate possible hardware and software issues. We found that the error was caused by a tilt of the colour sensor and the overbearing LEDs causing the margins between output values to be minimal, which we corrected to get the full functionality we had aimed for.

While working towards our aims and internal project milestones, we fulfilled all the operational requirements we had set for ourselves. Our final operational modes, program flow and function sets developed over time, but the basis of them remained constant. We developed a software that utilises a multitude of functions and can simultaneously operate in several ways. Note that the model ITS can detect sound and obstacles when detecting a line, even when on a curve due to the custom delay function we created (as can be seen in Appendix A).

8.1.1 Hardware

Even though the hardware structure was given as a complete kit, we have still managed to implement additional functionalities by adding external modules, and therefore enhance the general performance of the system. By placing longer metal shafts we have made the electronic framework safer by storing it in-between the black planes and improved positioning of the ultrasonic module. The TCS3200, in conjunction with the sound sensor, guarantee reliability and consistency in line tracking. Now, the car is capable of succeeding on all three test tracks with a minimal external input. Thereby, all functional requirements have been fulfilled; the scalable hardware car model can be further adapted and developed to fit into a real-life environment.

8.1.2 Line tracking

Line tracking by itself stands at a 75% success rate for the Silverstone tracks, 83% on the Electronics Lab: Short Circuit tracks and 95% on the custom red tracks. Throughout the whole timeline of the project, we have repeatedly prioritised greater accuracy

over smooth execution. This is evident in our choice of an oscillating movement to allow for accurate line tracking over the smoother implementation (see Line Tracking Test-2). The smoother implementation allowed the car to speed through straight sections of the track, stop when it arrived at a turn and then turn right or left accordingly. However, due to inertia working on the wheels of the car, it would often accelerate through turns and move a significant amount forward before detecting that it had gone off track. This made the car miss several turns and reduced the accuracy of the implementation overall. As previously stated, this smooth journey implementation allowed for a 30% success rate on the EE Lab: Short Circuit tracks. This was well below our desired threshold. Here, user experience has been compromised for finer accuracy of line tracking provided by our oscillating motion. It is, however, important to note that considerable jitter is produced due to this implementation and is one of the limitations of the current version of our program.

The current version's strongest property is its ability to automatically configure to the track colour and environment it is in. This is done by tailoring the program at the very beginning automatically, i.e. red, blue and green frequency values are recorded at the start and readings taken continuously for the rest of the journey are compared to those recorded original values. A match with those values is considered as a presence of a line and a mismatch is considered as an absence of a line. This requires the sensor and hence the car to be placed on top of the line, as opposed to the white surface, at the beginning of the program to allow the configuration to take place. This leads to there being no limit to the colours it can detect since the colour does not matter as long as values are different to the background the track is on, e.g. a white surface. This added functionality would not be available if the frequency values were hard coded instead. This configuration also allows the same version of the code to function in any environment the car may be placed in, since variation in values between environments does not affect the program.

As discussed in section 4.5 background research in tracking roads, the general research focus has been on tracking lanes via front mounted cameras and computer vision algorithms, rather than sensors underneath the car that would allow the car to detect lane boundaries as it moves over them or lines that may be etched onto roads. In this project, the tracks are significantly narrower than the prototype itself, and therefore the track is treated as a line to be followed (via various sensors directly underneath the scaled prototype), rather than a lane that the car is strictly required to stay inside of. However, self-driving cars are currently expected to assimilate into existing roads and traffic conditions alongside regular cars and roads are not expected to be modified to accommodate self-driving cars. Hence line following technology adopted by our prototype may not be scalable to full-scale vehicles and existing roads as there are no narrow lines to follow. More popular methods for detecting lanes such as computer vision algorithms may be required in order to scale this project up.

8.1.3 Obstacle detection and avoidance

Currently, the model can detect and avoid obstacles at a decent success rate. However, obstacle detection and avoidance are not fully functional. This is mainly due to the car to track size ratio, which is disproportionate -where the car is much bigger than the track. Therefore, the car

struggles to detect the path at times as the minimum speed required for the motors to work successfully without freezing is too high for the car to be able to move in smaller increments.

Additionally, friction affects all the calculations used to determine the necessary angles to implement the roundabout method. This means that the current model can only accurately implement the roundabout method on a bespoke surface – which is the yoga mat. Therefore, the current code can only be implemented to surfaces that provide the same friction as yoga mats which means the code needs to be constantly altered to fit the surface which imposes restrictions on the scalability of the ITS.

Finally, the current ultrasonic sensor (HC-SR04) can only detect obstacles that are directly in front of its path and are big enough (mainly height wise) to cover the whole range of detection for the sensor. This means the following obstacle placements cannot be detected by the ultrasonic sensor:

- obstacles that are at an angle in respect to the sensor
- obstacles that are too small for the sensor to detect from the current placement
- obstacles that partially cover the field of vision of the sensor

Subsequently, these factors reduce the accuracy of the implementation of obstacle detection and avoidance making the car partially successfully autonomous.

8.1.4 Audio Unit

One of the aims that was not achieved during the project was the ability to detect two consecutive loud sounds and react to them in an appropriate way. However due to time constraints and an unforeseen malfunction of the ITS as a whole close to the end of the project meant that we decided as a group to focus on getting the primary functionality of the car to work rather than implementing new sections of code that would cause the car to react differently to two consecutive peaks than one peak. This means that an entire quarter of the desired functionality for the audio unit was not achieved. This decreased the total functionality percentage by 8.75%, which is substantial. If this is not considered, we would have achieved 81.35% functionality in total.

8.1.5 ITS model performance

Our model ITS would currently classify as a Level 4 automated ITS, as it is able to have control of all driving functionality in most cases. However, there are some circumstances when it required human input to manage the situation it was in, for example occasionally on sharper turns, we would help it by alerting it of danger via clapping. In the future, the functionality of it detecting sounds should only be used to alert it of e.g. cars driving towards it at fast speeds that could crash into it, etc.

When completing this project as a whole, we focused mainly on our specific aims, on accuracy and consistency, trying to build a reliable system that could replicate its performance in a variety of environments. While we did this, we often made decisions that caused our system to be less user friendly, making decisions that would damage user experience if scaled to the real

world. We did however take the UK traffic rules into account when creating our designs, ensuring that the regulatory requirements of our designs were fulfilled. As we were developing the model, we ensured that all hardware could be adjusted and reassigned pins throughout the process, ensuring the maintainability of our design. The software is very easy to adapt. Furthermore, our design can adapt to its own surroundings, which makes it more scalable and maintainable in that respect.

We did not have time to further investigate the possibility of implementing any security measures, which would be a requirement for real world applications. By focusing on the functional requirements and aims of the project, we paid less attention to the non-functional requirements, which would be essential to build upon for real world application of our model. In some cases, these decisions were made due to the time pressure we had, and we have envisioned ways to improve upon our designs to develop a model that would be more scalable in the real world.

8.2 Future improvements

8.2.1 Hardware

Undoubtedly, one of the most severe problem we had to face was related to an insufficient friction between the wheels and track surface. To remedy sliding effects caused by that factor, we have fixed the surface to the floor, and placed a softer, planar material underneath. However, this has not eliminated the problem entirely. We believe that by having better wheels the sliding effect would be minimised when rotating and searching for a line.

Due to a fact that now only one colour sensor is present instead of three line tracking IR diodes, it is significantly harder for a car to understand at which side it is currently positioned, and where to direct itself at the nearest turn. To simplify our algorithm, a custom PCB module could have been designed to integrate three colour sensors, similarly to three IR diodes. Thanks to that, it would be possible to detect the line and its surroundings simultaneously, thereby increasing the performance at sharp turns even further.

8.2.2 Line tracking

One of the few drawbacks of the new colour sensor TCS3200 is the intensity of the four LEDs. The white light from the LEDs can often be overbearing for the photodiodes reading the values and interferes with the values received from the sensor, thus causing malfunction at times. The severity of this interference from lateral light required us to block out two of the four LEDs at a certain point in the project, allowing an immediate better performance of the sensor. Being able to control the brightness of the LEDs via the program or an automatic configuration is an aspect we would have explored in the future with more time as this would have boosted performance of the TCS3200 colour sensor.

A limitation of our current version of the code, due to its automatic configuration capabilities, is the requirement for the car to be placed on top of the line at the beginning of the program. One of the aspects of our code to focus on and improve would be this requirement and attempts

to remove it. A higher number of sensors would also allow configuration values to be read from different angles and not just a single point. This issue could thus be dealt via both hardware and software approaches.

Lastly, the primary limitation of the current version of line tracking code is the compromise of user experience for higher accuracy of line tracking. Improving the user experience by possibly implementing a smoother algorithm that does not cause as much turbulence but maintains the level of accuracy would be the main goal if we were to continue development of the line tracking procedure. This would be done by trial and error attempting various implementations of how the car actually moves to stay on track, via the code. Since the main reason for switching to this method was the poor performance of the smooth implementation when handling turns on the track, focusing on the turns would be a good starting point when changing the program. Improving accuracy at turns might remove the requirement to use an oscillating movement entirely and is worth exploring.

8.2.3 Obstacle avoidance and detection

Obstacle detection and avoidance could be further improved by replacing the current sensor with a sensor that has a wider range of motion "sight". This would enable the car to be able to detect obstacle of varied sizes positioned at different heights compared to the sensor and the car.

This consequently improve obstacle avoidance.
Currently, the car can detect obstacles while moving, however it requires itself to be stationary to enable the sensor to be able to emit and receive the ultrasonic waves. This weakens the car's capability to be autonomous as once it detects the obstacle the car itself becomes an obstruction on the road.

Therefore, having a sensor that can detect and calculate the distance of the obstacle while the car is in motion would make the car more efficient and adhere better to road safety.
Finally, to make the car fully autonomous in avoiding obstacles the car needs to have multiple sensors placed on different part of the car which can detect the various environmental factors around the car. This would improve the car's spatial awareness and it would prevent it from causing collisions while reversing, turning, or parking.

8.2.4 Audio unit

The audio unit could be improved upon by implementing further software that allows for multi-peak detection of loud sounds. In this way, a different quantity of loud sounds that surpass the peak level of 85 could trigger different kinds of reactions. A "language" of sorts could be implemented as a way for the ITS and its surroundings to communicate with one another. This "language" or communication protocol could consist of different length peak values combined with different quantity of consecutive peak values triggering different types on functionality.

The `millis()` Arduino function could be used to implement the analysis of how long a peak value is detected for and that could trigger different functionality. By adding more combinations of different length and number of peak values detected together, you could have a specific sequence that the car knows to emit at another moving entity in its surroundings, for example a pedestrian, in a specific scenario and similarly how to react when it receives the similar sequence back.

An essential improvement for the audio unit is for it to only react to specific stimuli rather than all loud noises around it. Loudness is not necessarily a particularly good indicator for danger and if scaled to the real world, this would possibly cause massive problems, as any loud noise would trigger the car to stop. While it is important for a sound to be loud for a human to be alerted to it, an autonomous car can have a more refined approach. If for example a microphone or other sensor was attached to the model instead of the loudness sensor, it would be possible to implement a design, by designing appropriate software that allows for the car to only react to sounds within a certain frequency range. This way, only specific stimuli would trigger the car functionality rather than all stimuli above one threshold value. A similar "language" to that mentioned before could also be implemented, making the system even more robust.

While adding more combinations gives more scenarios for the communications protocol to be applicable in, it must be kept simple enough for it to be scalable and usable by the population at large for it to be useful in the grand scheme of things. This must be considered when designing such a system in the future. Having a communication protocol that is based on sound, but not human language will also be more globally applicable.

8.2.5 Additional improvements

Despite the general improvements described above, a better coding approach could have been taken into consideration to make the algorithm asynchronous through an effective use of interrupts. To do so, an embedded C program can be written in the future instead of the Arduino one. In addition, an AVR-GCC compiler does indicate a greater number of syntax and compilation errors which makes the debugging process significantly easier. To sum up, by changing the software design approach it would be possible to implement additional functionalities to a greater extent, while keeping a number of polling operations at a minimum.

9 Conclusion

During the 11-week period, the team designed and built model of a level 4 ITS that can communicate intent with its environment. During the project, the team was able to successfully write a program that integrated all program functions to enable simultaneous use of all desired functionality, create a system that can handle any colour line and any environment by configuring to its surroundings at the beginning of its journey. The model achieved line tracking success rates of 83% on the EE Short Circuit tracks, 75% on the Silverstone tracks and 95% on the custom red tracks. A success rate of 62% was achieved for obstacle detection and avoidance in addition to achieving 75% of all desired functionality for audio detection and reaction. The total achieved functionality of the current ITS model is 73%.

Essential design choices made in the project included changing the original line tracking module with a TCS3200 Colour Sensor and creating a custom delay function that allowed the system to check for obstacles and sounds while in line tracking mode. This was especially useful when in the ITS was in the process of getting past sharp corners, as it allowed for us to warn the ITS when it was slipping off the tracks, which simulates a situation where the ITS would be going “off road”.

10 Bibliography

[1] World Health Organization. *Road Traffic Injuries.* World Health Organization. Last updated 7 Feb. 2020, Available: URL www.who.int/news-room/fact-sheets/detail/road-traffic-injuries [Accessed 6 Feb. 2021]

[2] Muqolli, E. (2019). *Communicating intent in Autonomous vehicles.* ASU Digital Repository. Available: URL https://core.ac.uk/reader/200249849 [Accessed 6 Feb. 2021]

[3] Czech Piotr, Turoń Katarzyna, Barcik Jacek. (2018*). Autonomous vehicles: basic issues.* "Scientific Journal of Silesian University of Technology. Series Transport" (Vol. 100 (2018), s. 15-22), doi 10.20858/sjsutst.2018.100.2 [Accessed 8 Apr. 2021]

[4] Shreya Sule, Kritak Gupta,Viraj Desai (2014). *Autonomous Cars: The Future of Roadways.* International Journal of Students Research in Technology and Management. Available: https://core.ac.uk/reader/268006177 [Accessed 8 Apr. 2021]

[5] National Highway Traffic Safety Administration (NHTSA). *Automated vehicles for safety.* Available at: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. [Accessed 8 Apr. 2021]

[6] Seth H. Horowitz. (2012). *The Universal Sense: How Hearing Shapes the Mind.* 1st edition. Bloomsbury. USA. ISBN: 978-1608198832

[7] Seeed studio "Grove – Loudness sensor," 101020063, Release date: 9.20.2015. Available: https://www.mouser.co.uk/datasheet/2/744/Seeed_101020063-1217445.pdf

[8] Department of Transportation. *The Highway Code – General rules, techniques and advice for all drivers and riders (103 to 158).* Published: 1.10.2015. Updated: 23.3.2021. Available at: https://www.gov.uk/guidance/the-highway-code/general-rules-techniques-and-advice-for-all-drivers-and-riders-103-to-158 [Accessed 7 Feb. 2021]

[9] Department of Transportation. *The Highway Code – Using the road (159 to 203).* Published: 1.10.2015. Updated: 23.3.2021. Available at: https://www.gov.uk/guidance/the-highway-code/using-the-road-159-to-203 [Accessed 7 Feb. 2021]

[10] R. Rajamani. Vehicle dynamics And control. Mechanical engineering series. Springer Science, 2006. [Accessed 13 Apr. 2021]

[11] S. Lapapong, V. Gupta, E. Callejas, and S. Brennan. Fidelity of using scaled vehicles for chassis dynamic studies. Vehicle System Dynamics, 47(11):1401–1437, 2009. [Accessed 13 Apr. 2021]

[12] Song, S., 2015. Towards Autonomous Driving at the Limit of Friction. Postgraduate. University of Waterloo. [Accessed 13 Apr. 2021]

[13] W. Farag and Z. Saleh, "Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems," *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Sakhier, Bahrain, 2018, pp. 1-8, doi: 10.1109/3ICT.2018.8855797. [Accessed 13 Apr. 2021]

[14] Paul V. C. Hough. Method and means for recognizing complex patterns. December 18 1962. URL https://www.google.com/patents/US3069654. US Patent 3,069,654. [Accessed 13 Apr. 2021]

[15] Nguyen, T., 2017. Evaluation of Lane Detection Algorithms based on an Embedded Platform. Postgraduate. Technische Universität Chemnitz. [Accessed 13 Apr. 2021]

[16] Mohamed Aly. Real time detection of lane markers in urban streets. IEEE Intelligent Vehicles Symposium, pages 7–12, June 2008. [Accessed 13 Apr. 2021]

[17] A. Borkar, M. Hayes, M. T. Smith, and S. Pankanti. A layered approach to robust lane detection at night. IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems, pages 51–57, March 2009. [Accessed 13 Apr. 2021]

[18] Vincent Voisin, Manuel Avila, Bruno Emile, Stephane Begot, and JeanChristophe Bardet. Road markings detection and tracking using Hough Transform and Kalman Filter. Proceedings of the 7th International Conference on Advanced Concepts for Intelligent Vision Systems, pages 76–83, 2005. [Accessed 13 Apr. 2021]

[19] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using B-Snake. Image and Vision Computing, 22(4):269 – 280, 2004. [Accessed 13 Apr. 2021]

[20] Texas Advanced Optoelectric Solutions® "TCS3200, TCS3210 Programmable Colour to Light Frequency Converter" TAOS099, July 2009. Available at: https://www.mouser.com/catalog/specsheets/TCS3200-E11.pdf [Accessed 13 Apr. 2021]

[21] Cytron Technologies "Product User's Manual – HCSR04 Ultrasonic Sensor", May 2013. Available at: http://web.eece.maine.edu/~zhu/book/lab/HC-SR04%20User%20Manual.pdf [Accessed 13 Apr. 2021]

[22] STMicroelectronics "L298 Jenuary 2000 DUAL FULL-BRIDGE DRIVER", January 2000. Available at: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf [Accessed 13 Apr. 2021]

[23] "SERVO MOTOR SG90 DATA SHEET". Available at: https://content.instructables.com/ORIG/FA2/O1SS/J7ARLNBW/FA2O1SSJ7ARLNBW.pdf [Accessed 13 Apr. 2021]

[24] Ayesha Iqbal, 9th Sept. 2020, Obstacle Detection and Track Detection in Autonomous Cars, Autonomous Vehicle and Smart Traffic, IntechOpen, Available at: https://www.intechopen.com/books/autonomous-vehicle-and-smart-traffic/obstacle-detection-and-track-detection-in-autonomous-cars [accessed 15 Apr. 2021]

[25] Vincent E. Roback, Larry B. Petway, Glenn D. Hines, Farzin Amzajerdian, Robert A. Reisse and Diego F. Pierrottet, 2013, Lidar Sensors for Autonomous Landing and Hazard Avoidance, NASA Technical Reports Server, Available at: http://hdl.handle.net/2060/20140002742 [Accessed 15 Apr. 2021]

[26] Seth Lambert, 5th March 2020, LiDAR systems: costs, integration, and major manufacturers, Mes Insights, Available at: https://www.mes-insights.com/lidar-systems-costs-integration-and-major-manufacturers-a-908358/ [Accessed 15 Apr. 2021]

[27] Autopilot, Tesla, Available at: https://www.tesla.com/en_GB/autopilot [Accessed 15 Apr. 2021]

[28] Kameron Rummel, 2019, The Fundamentals of Radar with Applications to Autonomous Vehicles, Liberty University Digital Commons, Available at: https://digitalcommons.liberty.edu/cgi/viewcontent.cgi?article=1933&context=honors [Accessed 15 Apr. 2020]

[29] Samuel Gibbs, 7th Sept. 2015, Lidars on top of self-driving car, the Guardian, Available at: https://www.theguardian.com/technology/2015/sep/07/hackers-trick-self-driving-cars-lidar-sensor [Accessed 15 Apr. 2021]

[30] Elec Freaks "Ultrasonic Ranging Module", HC-SR04, Available at: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf [Accessed 15 Apr. 2021]

[31] Digital Buzzer Module SKU DFR0032, Available at: https://wiki.dfrobot.com/Digital_Buzzer_Module__SKU__DFR0032_[Accessed 15 Apr. 2021]

[32] Vishay Semiconductors "Reflective Optical Sensor with Transistor Output", TCRT5000, TCRT5000L, Available at: https://www.vishay.com/docs/83760/tcrt5000.pdf [Accessed 15 Apr. 2021]

[33] ELEGOO Official. 2021. Arduino Kits User Support. [online] Available at: https://www.elegoo.com/pages/arduino-kits-support-files [Accessed 15 April 2021].

11 Appendix A – Program

```
// Group 9 D&B: Suvi, Nasmin, Nafia, Michal

#include <Servo.h>
#include <stdio.h>
#include "HardwareSerial.h"
#include <ArduinoJson.h>      //Use ArduinoJson Libraries

Servo myservo;                //create servo object to control servo

int Echo = A4;
int Trig = A5;
int rightDistance = 0, leftDistance = 0, middleDistance = 0;
int buzz = A1;
int sound = A0;               //define pin for audio input use

//Line Tracking IO define
int S2 = 2;
int S3 = 4;
int LED = 12;
int sensorOut = 10;

//define value for sound sensor input
int val;                      //to store value of input
int ss_flag = 0;              //start/stop flag - 0 means no movement, 1
                              means start moving
int danger = 85;              //value for threshold to detect a danger sound

int greenFrequency;           // Stores frequency read by the photodiodes
                              green component of colour
int blueFrequency;            // Stores frequency read by the photodiodes
                              blue component of colour
int redFrequency;               // Stores frequency read by the photodiodes
                               for red component of colour
int track_colour_green;       //stored green colour for track line
int track_colour_blue;        //stored blue colour for track line
int track_colour_red;         //stored red colour for track line

int rot_l;              //stored value for how much rotation to left required
int rot_r;              //stored value for how much rotation to left required
int inc;                //stored value for rotation increment

 //set pins
#define ENA 5
#define ENB 6
#define IN1 7
#define IN2 8
#define IN3 9
#define IN4 11

//set car speed
#define MOV_SPEED 150
#define TURN_SPEED 160
#define thirty_six 320        // angle 36 at speed 160
#define f_del 300             // forward delay
#define seventy_two 600       // angle 72 at speed 160
typedef unsigned char u8;     //Change the name of unsigned char to u8
```

```
// Sound and obstacle check enabled when using c_delay
void c_delay(u16 deldeldel) {
  for(int i=0; i<(deldeldel/5); i++) {
    delay(5);
    check_obstacle();
    if(check_sound()){
          emergency_mode();
          //reset values for line tracking
          rot_l = 25;
          rot_r = 27;
          inc = 1;
       }
   }
}

// Color testing function
int colour_test() {
  // select green channel
  digitalWrite(S2, HIGH);
  digitalWrite(S3, HIGH);
  delay(2);
  greenFrequency = pulseIn(sensorOut, LOW);   // Get colour input
  if (((track_colour_green - 5 ) <= greenFrequency) && (greenFrequency <=
(track_colour_green + 5))) {
    //select red channel
    digitalWrite(S2, LOW);
    digitalWrite(S3, LOW);
    delay(2);
    redFrequency = pulseIn(sensorOut, LOW);   // Get colour input
    if  (((track_colour_red  -  5  ) <= redFrequency)  &&  (redFrequency  <=
(track_colour_red + 5))){
      //select blue channel
      digitalWrite(S2, LOW);
      digitalWrite(S3, HIGH);
      delay(2);
      blueFrequency = pulseIn(sensorOut, LOW);   // Get colour input
      if (((track_colour_blue - 5 ) <= blueFrequency) && (blueFrequency <=
(track_colour_blue + 5))){
          return 1; //all three conditions met = line colour is detected
        }
      }
    }
  else { return 0;}  // We are not on the line
}


void BlinkLedEnable() {
  digitalWrite(LED, HIGH);
  delay(150);
  digitalWrite(LED, LOW);
  delay(150);
  digitalWrite(LED, HIGH);
  delay(150);
  digitalWrite(LED, LOW);
  delay(150);
  digitalWrite(LED, HIGH);
  delay(150);
  digitalWrite(LED, LOW);
  delay(150);
  digitalWrite(LED, HIGH);
  delay(150);
```

```
}

void fwd_left(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void fwd_right(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
}

void forward(u8 car_speed){
  int direction = 0;
    if(colour_test()) {
      if(direction = 0){
        fwd_left(MOV_SPEED);
        c_delay(50);
        } else if (direction = 1){
          fwd_right(MOV_SPEED);
          c_delay(50);
          }
    }
    else if(!colour_test()) {
      if(direction = 0){
        fwd_right(MOV_SPEED);
        c_delay(50);
        direction = 1;
        } else if (direction = 1){
          fwd_left(MOV_SPEED);
          c_delay(50);
          direction = 0;
          }
    }
}

void straight(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void back(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
```

```
void left(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void right(u8 car_speed){
  analogWrite(ENA, car_speed);
  analogWrite(ENB, car_speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}

void stop(){
   digitalWrite(ENA, LOW);
   digitalWrite(ENB, LOW);
}


//Ultrasonic distance measurement Sub function
int Distance_test() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(20);
  digitalWrite(Trig, LOW);
  float Fdistance = pulseIn(Echo, HIGH);
  Fdistance= Fdistance / 58;  //conversion to cm
  return (int)Fdistance;
}

void line_tracking(void) {
  int rot_l = 25;
  int rot_r = 27;
  int i = 1;
    if(colour_test()) {
      forward(MOV_SPEED);
    } else if(!colour_test()){
      while(!colour_test()) {
        check_obstacle();     //check for obstacle
        check_sound();        //check for sound
        left(TURN_SPEED);
        c_delay(rot_l);
        stop();
        c_delay(20);
        if (colour_test()) break;
        right(TURN_SPEED);
        c_delay(rot_r);
        stop();
        c_delay(20);
        if (colour_test()) break;
        rot_l = rot_l + 25 * i;
        rot_r = rot_r + 25 * i;
        i = i + 1;
      }
    }
}
```

```
void honk(void){
  //send output to buzzer
  digitalWrite(buzz, HIGH);
  // buzz for 0.5 sec
  delay(500);
  //stop outputting to buzzer
  digitalWrite(buzz, LOW);
  delay(20);
}


void emergency_mode(){
  stop();                       // one honk detected - stop moving
  //val = analogRead(0);        // check if honk is still happening
  delay(5000);                  //one honk, meaning 5 sec stop
  // if obstacle is detected
  middleDistance = Distance_test();
  if(middleDistance <= 30) {
    obstacle_avoidance();
    }
  /*second honk detected
  if(val > danger){
    stop();
    } */
}

void check_obstacle(){
   // if obstacle is detected - in mm
  if(middleDistance <= 30) {
    stop();                     //obstacle detected
    honk();
    //wait 2sec, if obstacle moves out of the way
    delay(2000);
    middleDistance = Distance_test();
    if(middleDistance <= 30) {
    obstacle_avoidance();
      }
    }
}

void check_sound(){
  val = analogRead(0);          //check sound sensor input
  if(val > danger){
    emergency_mode();
   }
}


void obstacle_avoidance(void) {

    myservo.write(90);          //setservo position according to scaled value
    delay(500);
    middleDistance = Distance_test();
    if(middleDistance <= 30) {
      stop();
      delay(500);
      myservo.write(45);
      delay(500);
      rightDistance = Distance_test();
      delay(500);
```

```
  myservo.write(90);
  delay(500);
  myservo.write(135);
  delay(500);
  leftDistance = Distance_test();
  delay(500);
  myservo.write(90);
  delay(500);

  if(rightDistance > leftDistance) {

    left(TURN_SPEED);
    delay(seventy_two);

    for(int checkPoint=0; checkPoint<10;checkPoint++){
      for(int i=0; i<10;i++){
        straight(MOV_SPEED);
        delay(f_del/10);
        colour_test();
        if (colour_test()==1){
          return;
        }// end if
      }// end for loop i
      right(TURN_SPEED);
      delay(thirty_six);
    }                       //END FOR LOOP
    stop();
  }                         //END DISTANCE COMPARISON RIGHT>LEFT

  if(rightDistance < leftDistance) {

    right(TURN_SPEED);
    delay(seventy_two);

    for(int checkPoint=0; checkPoint<10;checkPoint++){
      for(int i=0; i<10;i++){
        straight(MOV_SPEED);
        delay(f_del/10);
        colour_test();
        if (colour_test()==1){
          return;
        }// end if
      }// end for loop i
      left(TURN_SPEED);
      delay(thirty_six);

    }                       //END FOR LOOP
    stop();
  }                         //END DISTANCE COMPARISON RIGHT<LEFT

  else if((rightDistance <= 30) || (leftDistance <= 30)) {
    back(MOV_SPEED);
    delay(180);             //turn, track line
  }                         //END THE BLOCKED ROAD

  else {
                            //nothing
  }
}                           //END OF THE OBSTACLE STATEMENT

else {
                            //end the obstacle avoidance function
```

```
    }
}


void setup(){
  Serial.begin(9600);
  //configure servo motor - pin, pulse width corresponding to minimum 0
degree angle,  pulse width corresponding to maximum 180 degree angle
  myservo.attach(3,700,2400);
  // ultrasonic module initialization
  pinMode(Echo, INPUT);
  pinMode(Trig, OUTPUT);

  //Infrared tracking module port configuration
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(LED, OUTPUT);


// Setting the sensorOut as an input
  pinMode(sensorOut, INPUT);

  //Motor-driven port configuration
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);

  //audio port configuration
  pinMode(buzz, OUTPUT);
  pinMode(sound, INPUT);
  //for sound sensor
  pinMode(A0, INPUT);

   // Begins serial communication
  Serial.begin(9600);

  BlinkLedEnable();
  digitalWrite(S2, HIGH);
  digitalWrite(S3, HIGH);
  delay(10);
  track_colour_green = pulseIn(sensorOut, LOW); //check colour of track and
save to variable - this way outside of tracks shouldn't be detected as track
  digitalWrite(S2, LOW);
  digitalWrite(S3, LOW);
  delay(10);
  track_colour_red = pulseIn(sensorOut, LOW); //check colour of track and
save to variable - this way outside of tracks shouldn't be detected as track
  digitalWrite(S2, LOW);
  digitalWrite(S3, HIGH);
  delay(10);
  track_colour_blue = pulseIn(sensorOut, LOW); //check colour of track and
save to variable - this way outside of tracks shouldn't be detected as track

  stop();
}


void loop(void) {
```

```
  val = analogRead(0); //check sound sensor input
  middleDistance = Distance_test();
  line_tracking();
  myservo.write(90);  //setservo position according to scaled value
  check_obstacle();
  check_sound();

}


/* *** END OF PROGRAM *** */
```