MECHENG 706

Project 2 – Behaviour Control Robotics

Group 4

Eloise Beattie | Oliver Reedy | Otto Walker | Luke Hynds

UNIVERSITY OF AUCKLAND
Waipapa Taumata Rau
NEW ZEALAND

ENGINEERING
DEPARTMENT OF MECHANICAL AND MECHATRONICS ENGINEERING

Contents

1 Fire Sensing & Extinguishing System

One of the key subsystems of the firefighting robot was the fire detection and extinguishing system. This subsystem consists of several key aspects. First is the phototransistor array, consisting of three phototransistors. This is mounted onto the second subsystem, a servo motor, which allows the array to be pointed in different directions for the purpose of searching for the fire. Third, there is the alignment verification phototransistor, which allows the robot to check if the fire is within extinguishing range. Lastly, there is the fan, used for extinguishing the fire when in front of the robot.

1.1 Infrared Phototransistor Array Design

The core component of the fire detection system in the robot is the infrared phototransistor array. This array consists of three IR phototransistors, each capable of detecting infrared radiation emitted by flames, or in the case of the demonstration, a small incandescent light.

1.1.1 Switch Mode Phototransistors

The phototransistors were used in switch mode, giving a Boolean response on whether a fire is detected in the direction it's pointing, or not. Switch mode phototransistors were chosen for their simplicity and allow the robot to gain an insight into which direction the fire is in, quite reliably. The setup of the phototransistors is elaborated on in Section 4.

1.1.2 Phototransistor Alignment

An important design decision for the robot was how the array of phototransistors was to be mounted. It was decided that mounting the 3 phototransistors 60 degrees apart, on top of a servo controlled rotating platform was the best option, as shown in Figure 1.1. As the servo has 180 degrees of movement, there would only be a small blind spot behind the robot, which is scanned by rotating the robot around its wheelbase. This was decided to be the best traded off between accuracy and speed. The phototransistors are mounted with blinders, which can also be seen in Figure 1.1, that allow them to have a narrow field of view, and subject to less noise. This allows for detection using the unfiltered output.

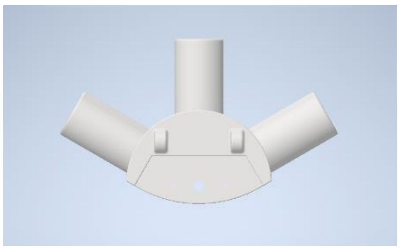


Figure 1.1: Phototransistor Array Mount

1.1.3 Verification Phototransistor

Only three of the four available phototransistors were used in the search array. This design was implemented as it was decided that the phototransistors would be used in switch mode as mentioned above. This meant that no distance data was available for location of the fires. To mitigate this, a verification phototransistor was implemented, which pointed downwards and would give a logical response if the robot was positioned point directly at a fire. If there wasn't to be a response from the verification phototransistor, the robot would know there was an obstacle in front of it.

1.2 Servo Control of Array

As mentioned previously, the phototransistor array is mounted on a servo-controlled platform, enabling dynamic adjustment of the array's orientation, while the robot is driving. The servo control allows the robot to actively scan its surroundings until it detects a fire. This dynamic control considerably increases the speed and accuracy of fire detection. In normal operation, while no fire has been detected, the robot will cruise around, constantly scanning for fires. Once a fire is detected, the robot will turn and face the fire straight on, and drive towards it.

The scanning stops and only the centre phototransistor is used for positioning while detection is maintained.

1.2.1 Spinning Sequence

To further enhance the coverage and detection accuracy, the robot employs a spinning sequence for the phototransistor array. As mentioned previously, the array is mounted on a rotating platform that rotates back on forward covering a 300-degree span. At the beginning of the run, the robot also spins around its wheelbase, allowing for full 360-degree detection. This spinning sequence allows the robot to detect fires in all directions, eliminating blind spots and ensuring comprehensive monitoring. The spinning mechanism is synchronised with the gyroscopic data of the spinning robot.

1.2.2 Rainbow Search

If a fire was detected, and its position is lost before the fire is extinguished, the robot utilises a "Rainbow Search" algorithm to prioritise areas with the highest likelihood of fire presence. This algorithm scans the locations directly in front of the robot more carefully first, and then increases the search angle. This allows for quick re-detection in most cases.

1.3 Fan Control

The extinguishing system of the robot is based on a fan, capable of directing concentrated airflow towards the detected fire. Once it is confirmed that the fire is directly in front of the robot, the fan is engaged until the fire is extinguished.

1.4 Adjustable Mounting

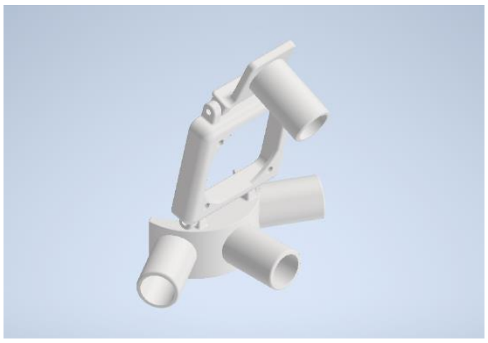


Figure 1.2: Adjustable Mounting Structure

The phototransistor array, fan and verification phototransistor are all mounted on an adjustable mount, allowing for precise positioning and alignment. This adjustable feature allowed for accurate tuning of angles, including that of the fan, and most importantly the verification phototransistor. The design of the adjustable mount can be seen in figure 1.2. By fine-tuning the angle and height of the array, we can maximize the effectiveness of the fire detection system, ensuring that the sensors can accurately detect the lights from various distances and angles. This was a critical design feature for the tuning of this subsystem.

1.5 Finite State Machine

The fire detection logic in our robot is setup using a finite state machine. The states of the FSM are defined by whether a fire is detected or not, or whether a fire has previously been detected. The FSM used for fire detection can be seen in figure 1.3, and a more detailed explanation of how finite state machines are used and promised within the system is covered in section 3.

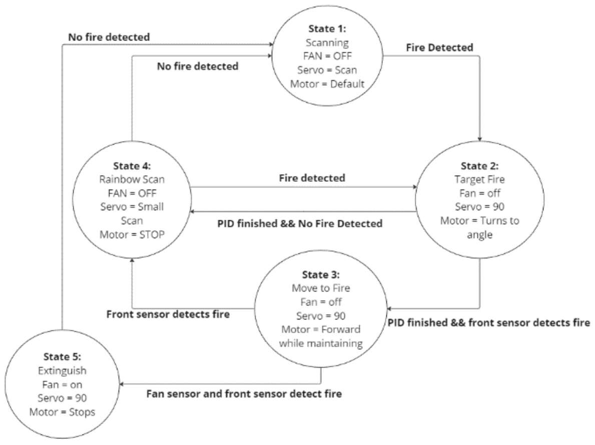


Figure 1.3: Fire Detection Finite State Machine

2 Obstacle Avoidance Subsystem

Obstacle avoidance is a critical component of our project, essential for preventing collisions and ensuring optimal performance. The system is equipped with five range sensors, including a sonar, two long-range infrared sensors, and two short-range infrared sensors. The design focuses on strategic sensor placement and locomotion behaviours to ensure the robot navigates without collisions.

2.1 Design Requirements and Assumptions

Collisions with obstacles such as walls significantly reduce the operational efficiency of the robot. It is imperative to develop a reliable method for detecting and avoiding obstacles to maintain high performance levels.

The scenarios anticipated involve the robot operating in environments with static and dynamic obstacles. The primary requirement is the reliable detection of obstacles within the robot's path to facilitate real-time avoidance. Given that the robot's movement is largely forward or rotational, it was assumed that obstacles would primarily be encountered at the front of the robot.

2.2 Sensor Placement

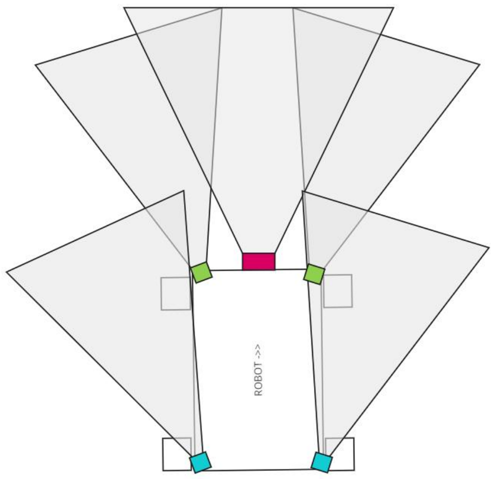


Figure 2.1: Sensor Placement

To ensure comprehensive coverage, the sensor configuration was meticulously planned. The sonar and two short-range IR sensors were mounted on the front to span all angles, ensuring that any obstacle directly in the robot’s path would be detected. The two long-range IR sensors were strategically placed to cover potential blind spots that could occur during lateral movements. This setup is depicted in Figure 2.1, where the sonar is depicted by the pink, short range IR sensors by green and long range by blue. It is clear in the figure where problem areas lie such as the blind spots between the short-range IR sensors and the sonar. The impacts of this blind spot had to be minimised to avoid collision. This was accomplished by tuning two parameters, the distance threshold and the angle of sensor placement. The distance threshold was what confirmed whether an obstacle was in the robot’s path, mentioned in more detail below, while the sensor angle is that angle the short-range IR sensors make with the centre line of the robot. Tuning of this parameter involved the prototyping of various mounts, with adjustments made in 5° increments until the optimal configuration was achieved. The development of the 3D mounts is further detailed in Part 4 – System Integration.

2.3 Sensor Calibration

The sensors were configured to provide a Boolean input: 'True' indicated the presence of an obstacle within a predetermined proximity (distance threshold), and 'False' indicated a clear path. This setup allowed for efficient debugging and tuning of the sensor ranges. Calibration was a crucial step where the voltage readings from the sensors were converted into distance

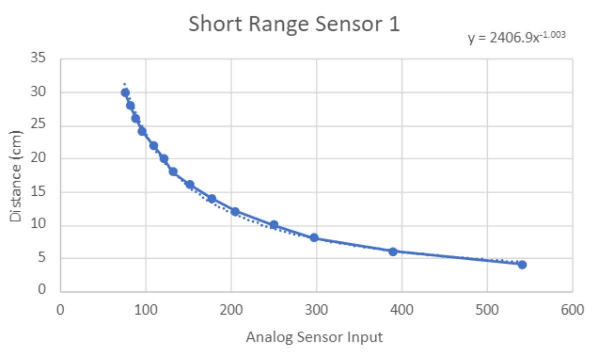


Figure 2.2: Sensor Calibration

measurements, allowing for a consistent meaningful distance threshold opposed to an obscure voltage threshold determined independently for each sensor. This process is shown in Figure 2.2. It was determined unnecessary for post processing and filtering of the sensor readings due to the sensor readings being in switch mode, where sensor noise only impacts the system when hovering around the distance threshold which does not occur for periods long enough to impact the robot's response, and therefore means any filtering would be redundant.

2.4 Obstacle Avoidance Logic

The obstacle avoidance logic in our robot is orchestrated using a finite state machine (FSM), which simplifies its implementation. As illustrated in Figure 3, sensor locations are designated as FM, FR, FL, BR, and BL, corresponding to Front-Middle, Front-Right, Front-Left, Back-Right, and Back-Left. Given the operational premise that the robot primarily moves forward, the FSM is designed to transition from obstacle avoidance back to default when the front three sensors are clear; otherwise, it indicates the presence of an obstacle. For general circular obstacles, the robot typically sidesteps. However, if sideways movement is blocked—as detected by the rear sensors—or if all front sensors are obstructed, the robot assumes a wall is present, opting to turn away rather than shift parallel to the obstruction. This strategy provides a robust response to a variety of obstacle configurations.

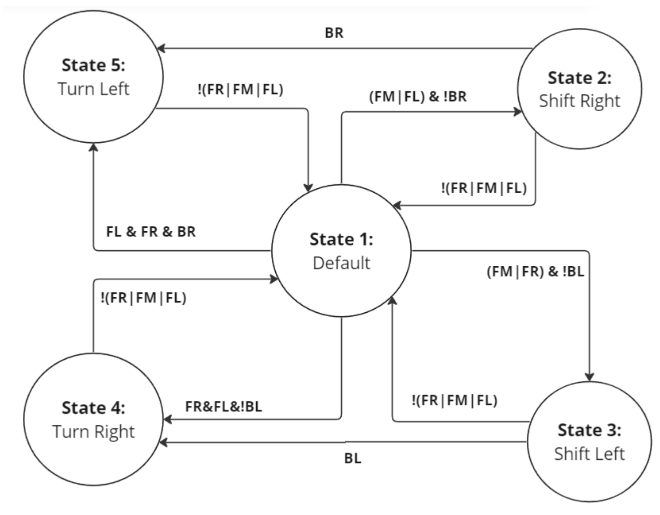


Figure 2.3: Obstacle Avoidance FSM

2.5 Results and Discussion

The obstacle avoidance system demonstrated high efficacy, successfully navigating around obstacles in about 90% of test scenarios, including during the robot demonstration. The primary challenge impacting system performance involved sensor range limitations. At large distances, sensors occasionally triggered false positives, causing the robot to manoeuvre around non-existent obstacles. More critically, obstacles positioned too close to the sensors resulted in false negatives, particularly problematic when the robot made sharp turns near an obstacle. These issues underscore the need for enhancements in future designs. Future improvements will focus on refining sensor placement and avoidance logic to more effectively handle these edge cases. This iterative approach is expected to further increase the reliability and accuracy of the obstacle detection system.

3 Behaviour Control

3.1 Code Architecture

To preface how the differing commands of the robot were decided, the overarching code architecture should be described for context. The main loop involves a Finite State Machine responsible for initializing the robot, running the robot, and stopping the robot once both fires are extinguished. Initialization involves zeroing the gyroscope and acquiring initial values for all the sensors. The final stopped state simply halts the motors and exits any form of sensor reading and printing. The bulk of the code and logic lies in the Running state.

Whilst the robot is in the Running state the following loop is undergone. A function responsible for updating all the sensors is called, updating the internal variables holding the values. A Boolean for whether a PID controller responsible for spinning the robot is updated for use in some of the inner FSMs. These sensor variables, along with specific required Booleans and pointers, are passed into the functions responsible for the obstacle avoidance, and fire detection and extinguishing. As discussed in sections 1 and 2, these functions are responsible for determining their own independent motor movements, fan commands, and servo movements.

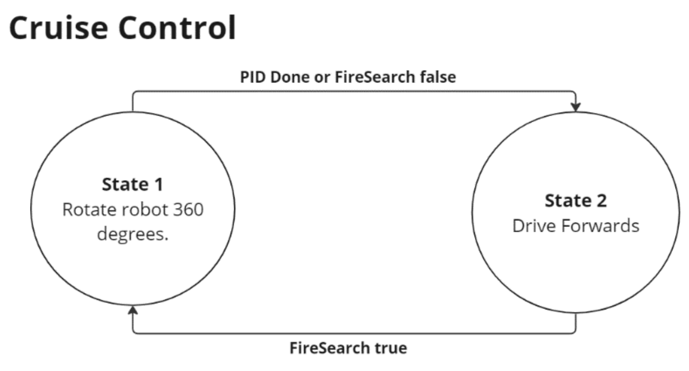


Figure 3.1: Cruise Control FSM Flow Chart

An additional function labelled cruise control was also created. Like fire detection/extinguishing and obstacle avoidance, this function is an FSM which outputs its own desired motor movements. Its purpose is to spin the robot upon entering the Running state or extinguishing a fire and drive forwards when a fire has been detected. The functions for Cruise Control, Obstacle Avoidance, and Fire Detection/Extinguishing were all contained within their own class labelled behaviour_control.

Returning to the main running loop, after running each function the arbitrate function is called, which contains the logic for behaviour prioritisation. Finally, the commands determined from the arbitrate are used passed into polled servo control, fan control, and motor control functions. Some additional logic is contained here which saves the previous state of both the servo and the fan. If the fan is determined to have switched from the FAN_ON state to the FAN_OFF state, an additional fire is determined to have been extinguished. Once reaching 2 extinguished fires, the robot will switch from the Running State to the Stopped State. Otherwise. If the servo is determined to have gone from any other state into the STOP_SCAN state, a temporary block is placed on the phototransistors preventing them from reading high.

3.2 Arbitrate

The arbitrate function was essential for determining the correct actuator commands at any given time. The cruise function is the first, lowest priority, and ever-present command. The flag for cruise is set to always be true, meaning if no command was provided by the fire sensing or obstacle avoidance, then the robot will either be driving forwards, or performing a full spin to search for a fire. The next lowest priority is the fire search.

The next, second highest priority is the obstacle avoidance, with the caveat that if the cruise control is attempting to spin, then the obstacle avoidance will be skipped. The reasoning for this was that it was told to us that the robot would be placed in a position where a full spin would be possible without hitting any obstacles, and we wanted to ensure that the robot would complete a spin and find the fire as quickly as possible without wasting time driving to avoid obstacles.

Finally, and most crucially, was an additional and highest priority fire extinguish check. The reason for this was to ensure that robot would not attempt to avoid an obstacle when that obstacle potentially contained the fire. A check for if the fire_flag was high, the front sonar was reading less than 20cm, and that either the fan command was FAN_ON, the fire_command was STOP, or the fire_command was TARGET_FIRE. If all these conditions were met, then it was determined that the robot was likely facing the fire, and instead of attempting to avoid it, it should instead prioritize the commands of the fire sensing and extinguishing, which should be to stop and turn the fan on.

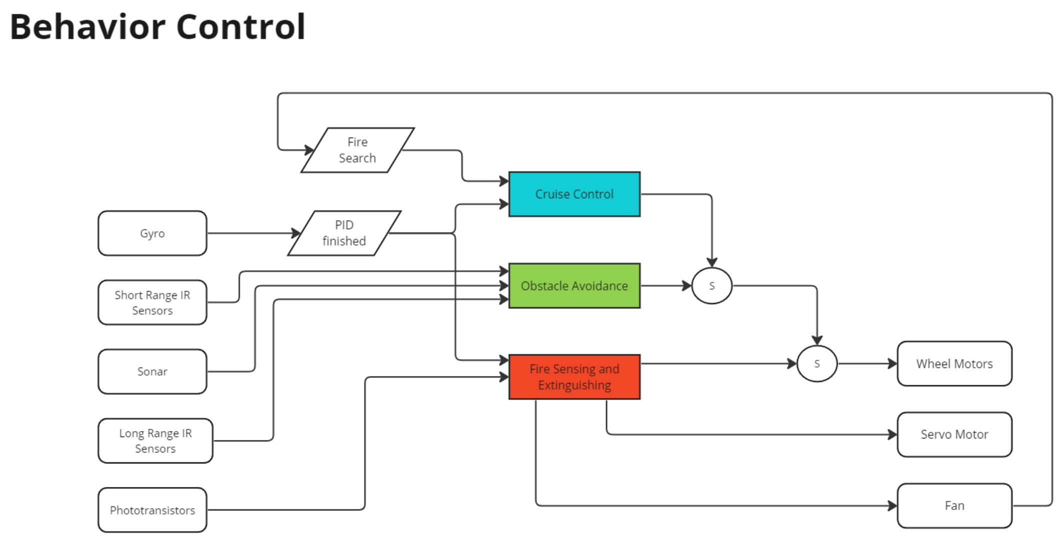


Figure 3.2: Sensor to actuator control logic

3.3 Commands

As only the fire function had servo and fan commands, these were always performed regardless of the fire_flag status. However, each flag would overwrite a global motion motor command as the arbitrate function was checked, with the cruise command being defaulted to.

Each command was saved as an Enum, unique to the relevant actuator. The fan command consisted of a FAN_ON and FAN_OFF command. The servo contained a SCAN command, a BABY_SCAN, and STOP_SCAN/EXTINGUISH. Although the last two were the same action of having the servo be stationary, the use of 2 names was useful for debugging. The motion commands consisted of a range of basic actions, containing movement and turning in every direction. The behaviour control saved the actions for each actuator in their respective Enum state for use in the functions to control the actuators, with a switch case to distinguish between the actions and apply correct signals to the pins.

4 System Integration

4.1 Sensor polling

The sensors used for position data in the project were four infrared based Distance Measuring Sensor Units, two GP2Y0A41SK0F with a range of 4-30cm, and two GP2Y0A21YK0F with a range of 10-80cm, an ultrasonic ranging module HC-SR04 with a range of 2-400cm, and an ADXRS642 gyroscope with a range of up to 250 degrees/sec. The four Infrared sensors and the sonar were polled at 50ms each, the sonar separately to the IRs. For fire detection, four SFH 300 FA phototransistors were used, polled at 5ms.

4.2 Electronics

The fan required a higher current to run than could be provided by the Arduino, so it was powered directly from the batteries (fig. 4.1.1).

All sensors connected directly to the Arduino board, with the exception of the phototransistors, which required a simple voltage divider circuit (fig. 4.1.2).

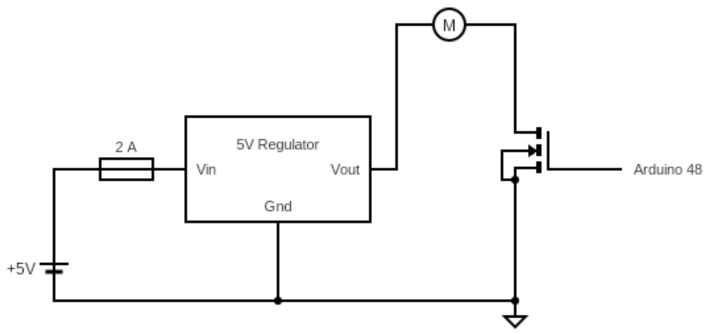


Figure 4.1.1: Fan Circuit Diagram

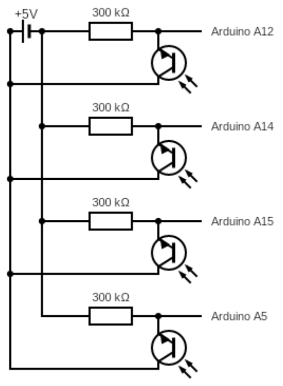


Figure 4.1.2: Phototransistor Circuit Diagram

All pins were connected to the Arduino shield (fig. 4.2) which then interfaces with the Arduino itself. Pins used were (A5, A12, A14, A15) for reading phototransistor outputs, (48) for controlling the fan, (A7, A8, A9, A10) for reading IR outputs, (40, 41) for connecting to the ultrasonic sensor, (A11) for reading gyroscope output, (10, 11) for Bluetooth connection, and (46, 47, 50, 51) for controlling the mecanum wheels.

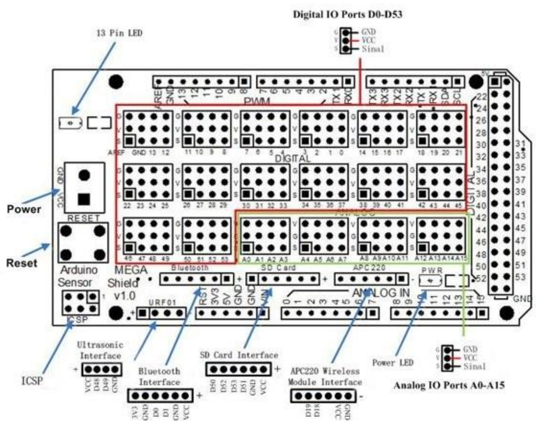


Figure 4.2: Pin Map

4.3 Actuator Control

The fan operates on a simple on-off principle in which the function fanControl takes a simple on or off command and subsequently writes high or low to the fan pin.

The servo utilises the arduino Servo library, allowing for the .write() command which operates on a Servo type object, taking an angle in degrees as input and turning the servo to that angle.

The mecanum wheels also employ the Servo library, using .writeMicroseconds() to send custom PWM signals to the motor pins. To allow the wheels to operate in conjunction with each other, a speed and direction is sent through the kinematics of the omnidirectional wheels, then the resulting velocity for each wheel is saturated and sent through writeMicroseconds() to each separate wheel.

4.4 3D Modelling and 3D Printing

All sensor mounts were 3D printed, as were the mounts for the servo and the breadboard. The final robot design contained 15 separate 3D printed parts, all designed as best as possible for manufacture, and all connecting via nut and bolt to the given mounting holes in the metal robot body. The ideal angles of the sensor mounts were determined experimentally, where weak versions of each likely angle were printed together then trialed, with the optimal angle, when found, being reprinted with a smaller nozzle size and at a slower rate to make a stronger product less prone to bending and deforming. This was done to change angles instead of variable mounting points as it was found to be more resistant to bumping, and the larger holes found in the variable prints were too much of a weak point. Screw holes were printed with zero tolerances, allowing the thread of the screws to catch the plastic and create a more secure mount.

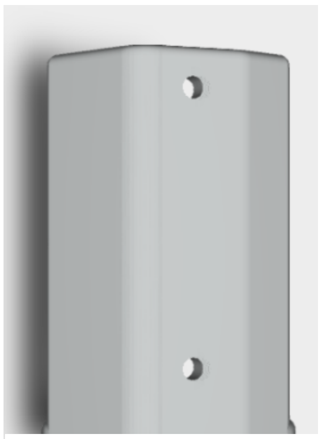


Figure 4.3: Front IR Sensor Mount

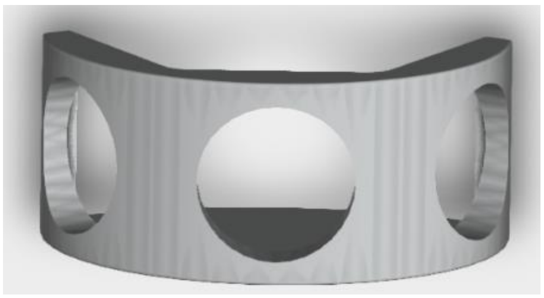


Figure 4.4: Phototransistor Array Mount

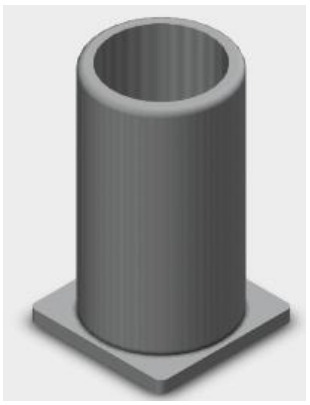


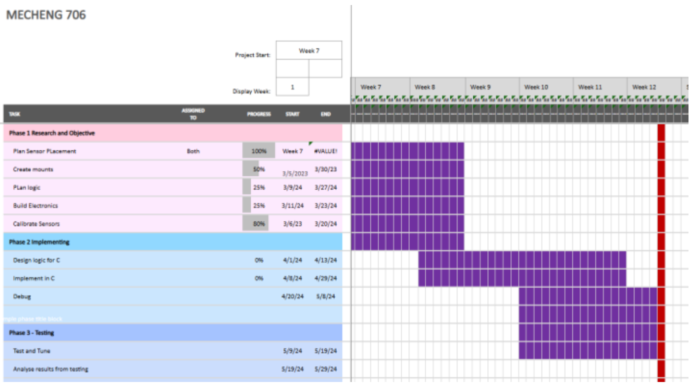
Figure 4.5: Phototransistor Focusing Tube

5 Project Management

5.1 Project Management Skills

During the development of our autonomous fire sensing robot, we employed several key project management skills learned in ENGGEN 303. One of the primary skills was the creation of a detailed Work Breakdown Structure (WBS), which allowed us to break the project into smaller, manageable tasks. This structure ensured clear task delegation and facilitated better tracking of progress across various stages of the project. Each team member was assigned specific responsibilities, ensuring accountability and clarity in our collaborative efforts.

Another important project management skill was effective time management. We developed a project schedule using a Gantt chart, which provided a visual representation of our tasks and deadlines. This helped us to try and stay on track working towards the deadline.



MECHENG 706

Project Start: Week 7

Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Research and Objective				
Plan Sensor PLacement	Both	100%	Week 7	#VALUE!
Create mounts		50%	3/5/2023	3/30/23
PLan logic		25%	3/9/24	3/27/24
Build Electronics		25%	3/11/24	3/23/24
Calibrate Sensors		80%	3/6/23	3/20/24
Phase 2 Implementing				
Design logic for C		0%	4/1/24	4/13/24
Implement in C		0%	4/8/24	4/29/24
Debug			4/20/24	5/8/24
Phase 3 - Testing				
Test and Tune			5/9/24	5/19/24
Analyse results from testing			5/19/24	5/29/24

Figure 5.1: Gantt Chart

5.2 Agile Project Development

Adopting Agile project development principles significantly enhanced our project management process. This approach allowed us to respond flexibly to changes in design and incorporate feedback from our learnings along the way. Each week we discussed the current sprint we were working on, and what the next steps moving forward were. This allowed us to discuss progress and address any obstacles. This environment meant our team was on the same page and team members could share updates, seek assistance, and adjust plans as necessary, where other people made critical changes.

The iterative nature of Agile development enabled us to integrate feedback continuously, ensuring that our design improved, and progress wasn't lost when changes were made by other team members. By breaking the project into manageable sprints, we could test and refine each component before moving on to the next. This approach not only improved the overall quality of our robot but also ensured that our robot's subsystems worked at each checkpoint allowing them to be combined at the end.

5.3 Software Used

To manage our project efficiently, we utilized several software tools. Miro was instrumental for task breakdown and tracking critical changes to the design. Its intuitive interface allowed us to create sections for different project phases, and a central place for all project information.

Figure 5: Miro Workflow

For version control and collaborative development, we relied on GitHub. This platform facilitated seamless code sharing and version management, allowing multiple team members to work on different aspects of the project simultaneously without conflicts.

5.4 Contributions

Eloise Beattie	Worked on behaviour control logic, including coding and debugging of class. Wrote the Obstacle Avoidance subsection of report. Assisted in proof reading report.
Luke Hynds	Wrote Fire Sensing/Extinguishing and project management, 3D modelled structure additions and integrated electronic components. Assisted in debugging code and report editing.
Otto Walker	Wrote System Integration subsection of report. Worked on 3D design of robot structure additions.
Oliver Reedy	Worked on behaviour control logic, coding and debugging. Wrote the Behaviour Control subsection of report. Assisted in proof reading report.