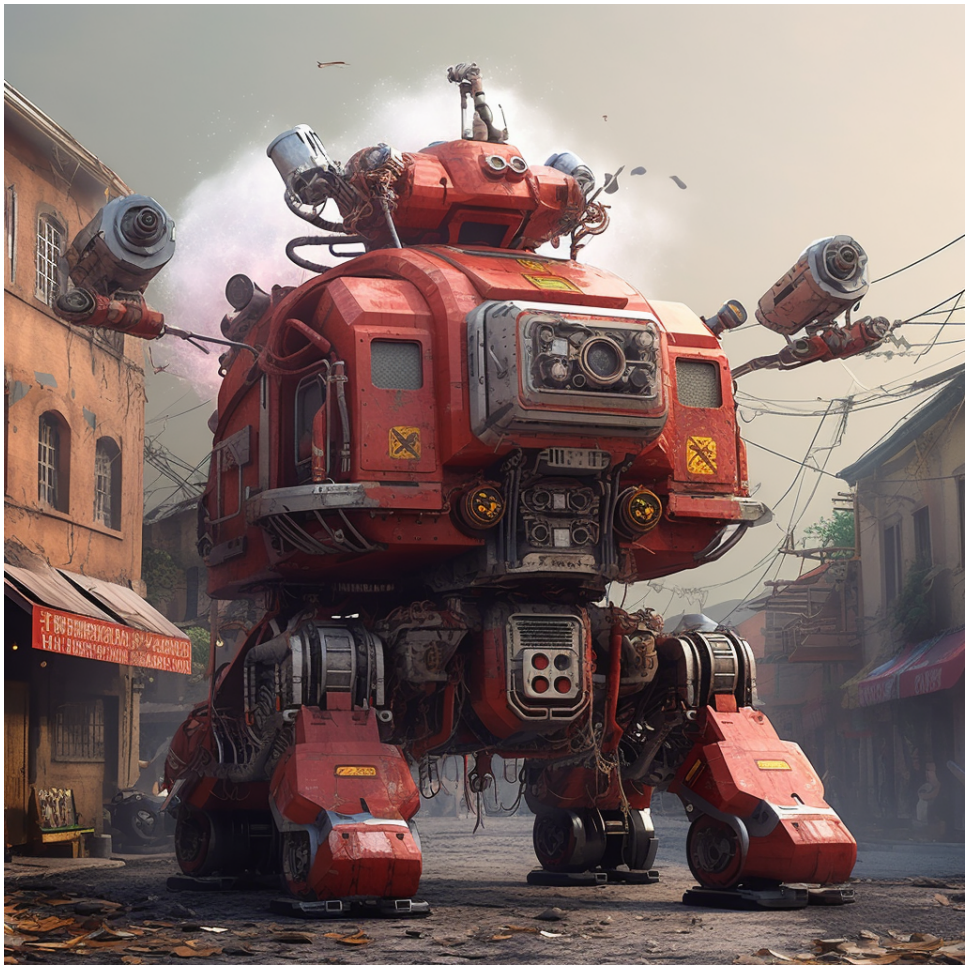
Project 2
MECHENG 706
Group 11



Bhumik Patel - Student 1
Peter Thompson - Student 2
Fraser Eade - Student 3
Trevor Heinemann - Student 4

Contents

1 Fire Sensing/Extinguishing System - Bhumik Patel

This section presents the methodologies employed in the design and implementation of the fire sensing and extinguishing system for the mecanum wheeled robot, along with the control techniques utilized throughout the process. The primary objective involves the detection of two 12V lamps, subsequent navigation towards them, and activation of a DC fan, thereby extinguishing the "flame".

1.1 Fire Sensing

To enhance fire detection capabilities, a configuration utilizing four phototransistors was implemented, specifically wired to detect white light. Among these phototransistors, two were designated as long-range sensors for extended reach, while the remaining two were designated as short-range sensors. This arrangement ensures optimal visibility and transparency as the robot navigates around obstacles on the table. The phototransistors were strategically positioned at a uniform height of approximately 15cm above the ground. This positioning enables them to overlook obstacles and effectively sense the simulated "fire" from a considerable distance. The long-range sensors were carefully calibrated to cover an effective range of 500-2000mm, employing a 98kΩ resistor, while the short-range sensors were calibrated to operate within 100-700mm using a 6.8kΩ resistor. This comprehensive calibration setup enables the fire sensing system to effectively monitor the entire table area.

The phototransistors are arranged in a uniform manner alongside the fan, as depicted in Figure 5. To ensure accurate calibration, the sensors were tested at 100mm intervals using various resistor values. However, the most optimal results were obtained with the selected resistor values. The calibration curves for these values are illustrated in Figures 2 and 3. A power function is employed to convert the raw readings from the phototransistors into corresponding distances. Given their characteristics, the phototransistors can be polled every 10ms through a timer interrupt, while the remaining sensors can be queried at different intervals. This allows the robot to swiftly monitor its environment and gather crucial information for the control system. The wiring arrangement for these sensors is depicted in Figure 1.

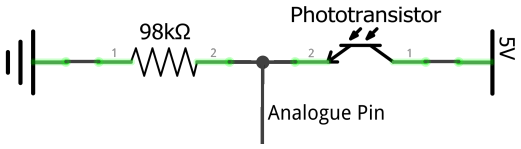


Figure 1: Phototransistor Wiring

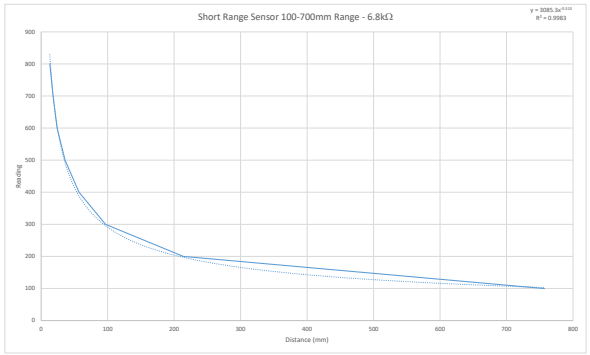


Figure 2: Short Range Calibration

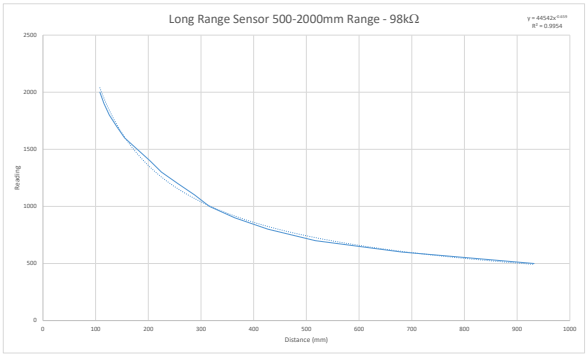


Figure 3: Long Range Calibration

1.2 Fire Extinguishing

In addition to the phototransistors, a DC fan was incorporated into the fire extinguishing system to effectively extinguish the simulated fire. The phototransistors played a vital role in providing accurate distance measurements to ensure precise positioning of the robot relative to the "fire". The positioning of the fan can be observed in Figure 5. Initially, all these components were mounted on a servo motor, but a decision was made to eliminate its usage in order to simplify the system. The wiring configuration for this setup is illustrated in Figure 4. To power the system, the robot's onboard lithium-ion battery is employed, alongside a voltage regulator to step down the voltage to 5V. This 5V power source is then utilized in conjunction with an N-channel MOSFET, enabling control of the fan through an Arduino digital pin. When the system detects a light source, the fan is activated for a duration of 10 seconds to simulate extinguishing the "fire".

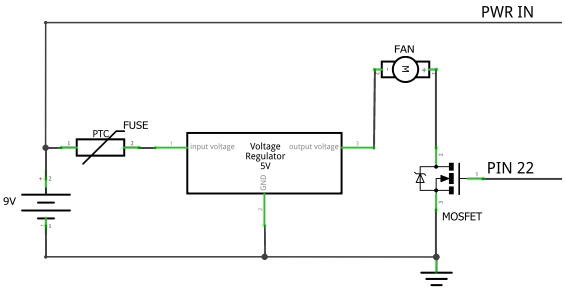


Figure 4: Fan Wiring

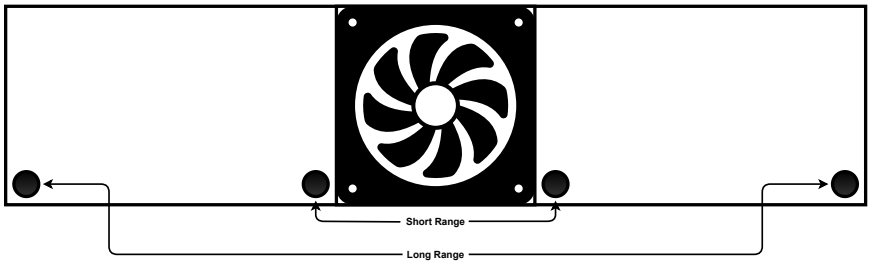


Figure 5: Sensor and Actuator Layout

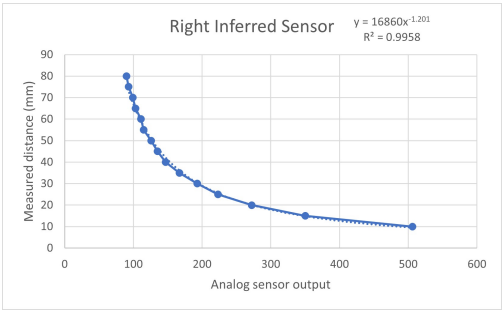
1.3 Navigation and Control

The fire sensing system is a vital subsystem of the overall system, enabling the robot to accurately detect and approach the "fire" while effectively avoiding obstacles. This functionality is achieved through the utilization of the `avoid()` function. As each sensor reading is obtained, values exceeding their effective range are set to `9998` or `9999`, indicating that they are out of range. These values play a crucial role later on in determining a merged sensor reading for both the left and right sides. The merging process is performed twice, once for the left sensors and once for the right sensors. The resulting merged values are then utilized by the `control()` function to determine the appropriate next move. Additionally, the rotation speed is dynamically adjusted within the loop to optimize the detection of the "fire".

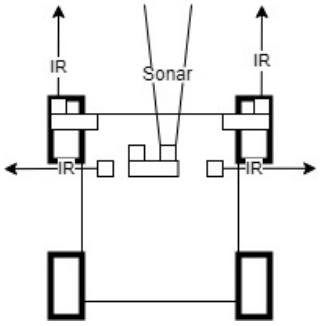
To address a previously encountered issue where the robot would oscillate while focusing on a light source, a dedicated function was developed. This function effectively detects such oscillations and employs flags to execute a series of commands in a specific order, preventing the occurrence of oscillations. Another challenge that was faced involved signal loss during robot rotation. To overcome this issue, a strategy was implemented wherein the robot utilizes strafing movements instead of rotational motions, allowing for more precise adjustments without experiencing signal loss. To align the robot with the fire source, the average reading of the two short-range sensors was calculated, and the robot's speed was reduced, resulting in successful alignment.

2 Obstacle Avoidance System - Peter Thompson

When designing the obstacle avoidance system, both the physical placement, along with the types of sensor and design of the program architecture are important aspects of the system. Starting with the physical sensor placement, the layout can be seen in figure 6b, were the ultrasonic sensor and two inferred sensors were used to detect obstacles in front of the robot, and one inferred sensor was used either side to check for obstacles before side strafing. However, in order to effectively utilise the sensors they needed to be calibrated so that the raw sensor outputs could be converted into mm for easy use. To do this each sensor was setup to take a number of measurements at 100mm increments, with their raw outputs recorded. This was then stored in an excel file to allow a regression equation to be found, as can be seen in figure 6a for the right inferred sensor. However the there was still a lot of variability in the sensor readings, with the occasional large spike in the readings. To resolve this a 5 element moving average filter was used to smooth out all the results. Given the use of behavioral control, the software implementation of the obstacle avoidance



(a) Right Inferred Sensor Calibration



(b) Sensor Placement

Figure 6: Sensor design and initialisation

was contained in a single function called avoid(), that would take all the current sensor readings and output an action, in the form of a command that would instruct the robot to move forward, backward, strafe right, turn CW, etc. until the function was called again by the control function and the command updated. The avoid function logic can be viewed in figure 7, and can further be broken up into three primary sections, 1. edge case detection (yellow) 2. avoiding obstacles in front (green), and 3. avoiding obstacles to the side (purple).

1. Edge Case Detection (Yellow) Given the primarily reactive nature of the control, there exist a number of situations where the robot can get stuck moving back and forth repeatedly. An example of this is when the the robot faced a wall, it would see an object in front of it and strafe left to avoid it, resulting in the robot getting stuck on the wall, and then eventually in the corner. To avoid such situations as this, a "getUnstuck()" function was created that could be triggered in edge cases such as this, to override the usual control output of the avoid function, to reverse for approximately 1 second, before turning CW for half a second and resuming the usual function control. The way this was triggered and was acheived was primarily through the use of two counters; fCounter, which counts the number of loop cycles the Avoid() function detects an object in front of the robot in a row until 1 second has passed, after-which triggering getUnstuck() and resetting the counter; and getUnstuckCounter, which once getUnstuck() has been called, increments each cycle, and continues to intercept the usual Avoid() logic and call getUnstuck() until the specified duration has passed, at which point it resets and allows the usual logic to continue. The other

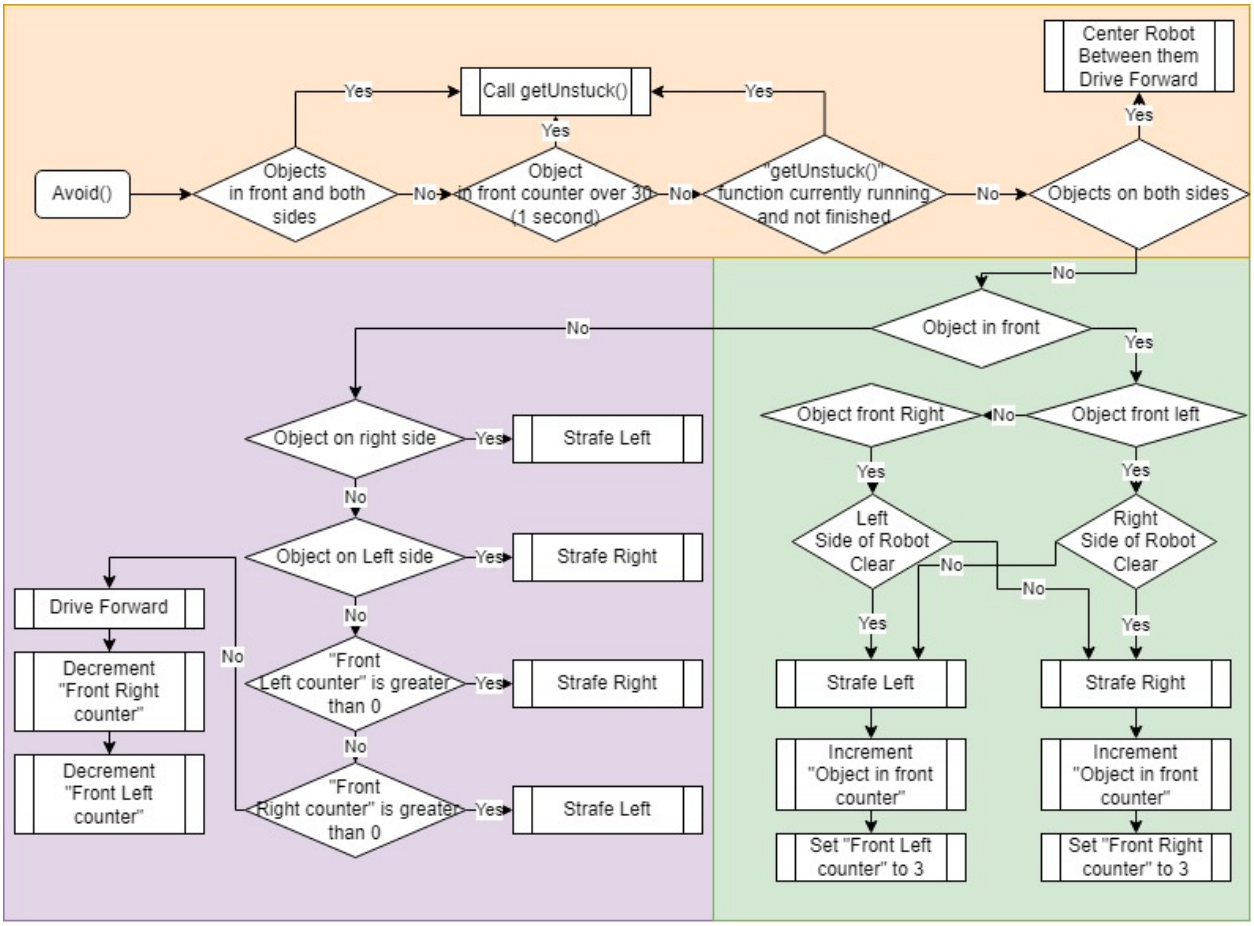


Figure 7: Flowchart of avoid() function logic

two edge cases are in the event obstacles are detected on both sides and in front, in which case getUnstuck() is called again. Or in the event that the robot is traveling between two objects, in which case the robot will center itself between them and drive forward.

2. Avoiding Obstacles in Front (Green) The section of the logic is comparatively simple, if an object is detected in front of the robot, check if its on the front left or right side, and strafe in the opposite direction, and if an obstacle is in the way of the intended strafe, strafe in the opposite direction. Additionally a front left or front right counter is set to 3 each time an object is detected by either the front left or right sensor, which instructs the avoid function to continue strafing in the intended direction for an additional 3 cycles of avoid, to ensure the robot is cleared of all obstacles before proceeding forward.

3. Avoiding Obstacles to the Side (Purple) The logic in this section is very simple, and can be boiled down to, decrement the front left and right counters, and if an obstacle is detected on either side or if the front left or right counter is still greater than 0, strafe in the opposite direction. If none of these conditions are met, a default instruction is set to drive forward.

The combination of these sensors and this logic results in a relatively simple but effect obstacle avoidance system that allows the robot the complete it intended task, with minium collisions.

3 Student 3: Fraser Eade: Behaviour Control

The behavioural control for the firefighting robot is the reactive control version of behaviour based control. The function makes use of several variables set up before the function begins. These variables check for when extinguish has been running for a certain period of time, if the short range photo transistors are reading a value closer than threshold, and if the distance sensors are indicating there is an object within threshold distance. The distance sensors include an ultrasound sensor and 4 infrared sensors, 2 short range and 2 long range. Additionally important to the code is the fact that these thresholds are set in another part of the body of the code and certain thresholds are modified in the behavioural control. The machine has four states called finished, extinguishing, avoid and seek. The order of priority they are called in is finished with highest priority, extinguishing with second highest, avoid third highest and seek with the lowest priority. See

3.1 Finished State

A summary of the entire finite state table can be seen in**??**. One of the variables that is set up is an integer that counts up once the extinguish function has been active for a long enough time and the programme exits extinguish. In other words this variable tracks how many times extinguish has been called and successfully completed. When this variable reaches two or in other words when the robot has extinguished two fires then the machine enters the finished state. In this state the motors are stopped and the programme stops running as the robot has completed its tasks.

3.2 Extinguish State

Extinguish can only be entered when the finished state does not have its conditions met. The programme checks if the average value of the two short range phototransistors is below the threshold value. If it is below the threshold value then the function sets the threshold value to be slightly higher. This made the programme a bit more stable and less prone to going in and out of extinguish as slight changes to light quality made the programme exit the state. In extinguish the control loop calls the function extinguish which stops the motors from moving and turns the fan on. Also inside of this programme is a check for when the extinguish function has been running for 3 seconds. This indicates the programme is in extinguish stably and changes a boolean to indicate that the programme has been extinguishing. This boolean is utilised in avoid to keep track of how many fires have been extinguished.

3.3 Avoid State

Avoid can only be entered when the extinguish state and finished state do not have their conditions met. The avoid function requires the distance sensors, namely the ultrasound sensor or any of the 4 infrared sensors to have detected an obstacle within the threshold for that particular sensor. This indicates there is an obstacle close to the robot and the robot must move away from the obstacle. This function increments the counter to keep track of how many times the robot has extinguished a fire if the boolean set to true during the extinguish state is true. In other words if avoid is activated after extinguishing a fire the variable showing how many times a fire has been extinguished will increase. The avoid function also sets the photo transistors sensing threshold to be lower once more for the short range photo transistors to prevent incorrect seeking. The state also stops the fan from spinning and calls the avoid function which will move the robot out of the way of the obstacle.

3.4 Seek State

Seek is the lowest priority state and as such can only be entered if none of the other states have their conditions satisfied. The seek state sets the threshold for light detection to enter the extinguish state back to its normal value to make sure it does not remain the threshold set during extinguish. Seek also increments the extinguishing count and sets the boolean to keep track of whether or not the programme has been extinguishing back to false. Seek also turns off the fan and calls the pointSimple function which will search for the light source (the fire) and drive towards it.

3.5 Overall Architecture

One can therefore see that all together this function enters search state to find the fire, then when encountering an obstacle enters the avoid state to move away from or past the obstacle. Then once close enough to the fire and pointing at it the programme enters the extinguish state to put out the fire. Finally when both fires are put out the programme enters the finished state and ends. Additionally every loop in the control loop checks if the light source is within 250 millimetres of either of the two short range sensors, if it is the speed the robot moves at is reduced to allow the robot to be more accurate. If it is not within 250 millimetres speed is set to normal.

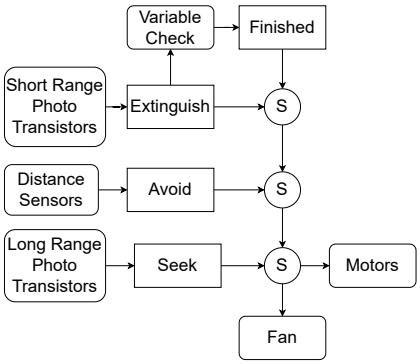


Figure 8: Behaviour Based Control Diagram

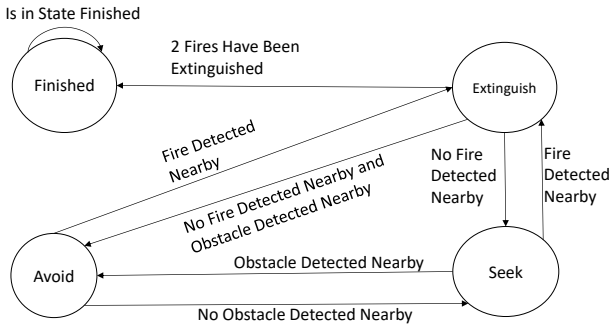


Figure 9: Finite State Machine Diagram

4 System Integration - Trevor Heinemann

The system integration concerns the actual implementation of the required and described functionality, in the realms of both hardware and software. From prior experiences and initial testing, it was rapidly determined that the primary design space for the project would be software; the available hardware sensing and actuation capabilities were limited and hardware iteration is naturally slower. In contrast, the available memory and microcontroller processing capabilites far exceeded the necessary requirements for our control schemes, which required almost no optimization to implement.

In order to seamlessly bridge the gap between hardware and software, much work was done in software to process and modulate sensor inputs and actuator outputs to account for sensor and actuator inconsistencies. Additional consideration was required to ensure that a viable pinout configuration could be created to prevent the microcontroller I/O from bottlenecking the system.

4.1 Software System Implementation

As noted in prior sections, the overarching control technique for our robot was one of reactive control, coupling our sensor inputs to actuator outputs with only a very brief time between in which all logical control decisions were made. This control scheme allows for a robust system with fast reactions, but has the drawback of a simplified memory and no precognition leading to potential issues with edge cases and behavioral loops. Ultimately we found this control methodology to be ideal for the application, as the tasks themselves were fairly simple but required immediate reaction to dynamic conditions.

This decision to utilize reactive control and couple our actuators to our sensing naturally means that behavior change can only occur at the rate of our slowest sensor polling rate. We found that while our phototransistors could be polled at sampling periods as low as 10 ms, other sensors such as the ultrasonic could only be reliable polled at around 600 ms. Accordingly, we structured our code such that every 600ms it would complete a series of logic checks implemented by way of a finite state machine to determine the correct behavioural output based on the sensory readings. If an obstacle was detected by the ultrasonic or infrared sensors, we would enter our avoidance algorithm (detailed in section 3), if we saw no nearby obstacles or light source we would enter the light seeking algorithm (detailed in section 1) and so on. This approach limited the speed of logical state changes in the robot to the polling rate of our slowest sensor, but higher polling rates were still used for some sensors, in order to help offset the delay created by our moving average filter.

Once the core control logic was able to determine the appropriate state for the system, an individual function for each intended behavior would be called, but notably each behavioral state function had to be designed in such a way that it would function correctly by being called continuously. This was done because allowing each function to trap the system within its own logic, by using a 'while' loop for instance, would prevent the system from reacting dynamically to changing conditions. However this also necessitated the creation of new memory variable for any information that would need to be stored for more than the cycle time of 600ms. Examples of these memory variables included the number of fires extinguished, in order to stop the robot after two fires, or the number of times in a row the avoid function was called, in order to call a different function if the robot appeared to be stuck on a particular series of obstacles. As detailed, this approach meant that each possible edge case of ways the robot could get stuck required its own memory variable or set of memory variables to detect, and its own behavioral state to solve, which fortunately was made feasible by the relatively simple environment and limited possible sensor inputs.

Once each subfunction or behavioral state had determined the correct course of action, it

would return the desired actuator output in the form of a direction of travel. These actions were implemented through an enumerated type, and converted into motor output through use of a tuned processing function designed to account for motor inaccuracies and to provide linear actuation and movement. Notably, the nature of the mecanum wheels was utilized to implement diagonal and curving travel directions in addition to the standard four, which in conjunction with the logic states had the effect of creating a pseudo piecewise P controller where the robot was able to continuously correct its direction of travel.

4.2 Hardware System Implementation

Unlike the software design, the physical hardware configuration of our robot was designed to be as simple and therefore robust as possible. The initial design of mostly front facing sensors (as detailed in section 2) was found to be satisfactory, and only slight iterations were required. Specifically, the front facing obstacle detection IR sensors needed to be moved to the width of the wheelbase to avoid clipping obstacles, and the servo motor below the fan and phototransistor array (detailed in section 1) was deemed unnecessary. The servo motor mount was initially designed in order to more quickly locate the light sources, but we found a moving sensor array coupled with the inherent inaccuracy of the phototransistors to be unreliable and ultimately unnecessary.

Our electrical design was also moderately simple, connecting each of the sensors and actuators to the necessary Arduino pins, with no shortage of analogue input pins for our sensors. The specific circuits for our various sensors are detailed in section 1. Our electrical design also necessitated the use of a breadboard in order to facilitate rapid iteration and adjustment, though we did sometimes encounter errors with wires coming loose. Additionally, on the day of testing, the voltage regulator for our fan began to cause issues, and was ultimately deemed to be unnecessary and was not used for the final runs.

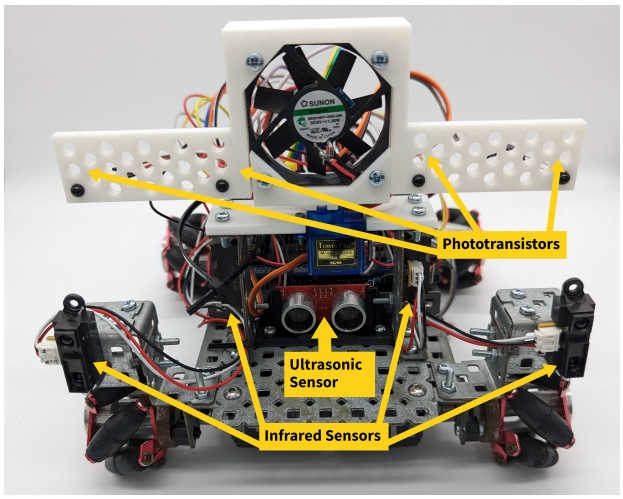


Figure 10: Frontal Sensor Array

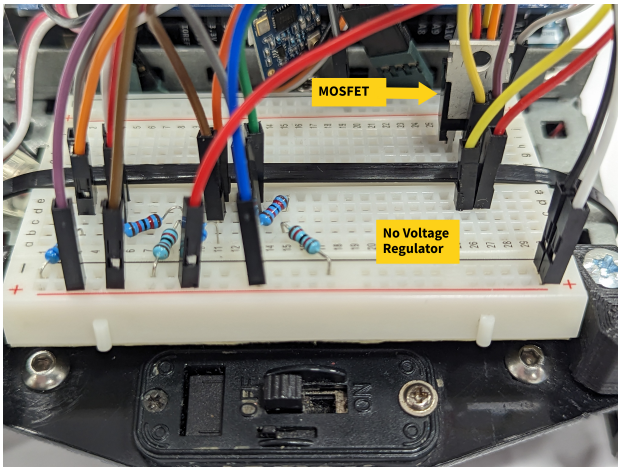


Figure 11: Breadboard Configuration

5 Project Management

The project required multiple skills from the members of this group including time management, task allocation and scheduling alongside the technical skills of building and coding this robot. There were several transferable skills from ENGGEN 303, some of which were used and some of which may have helped had they been used. This project made us of Agile product development techniques with cycles of 3 weeks and utilised Jira to help manage the project. Each student made their contributions in certain areas.

5.1 303 Application

At the beginning of the project, the first step was to decide what type of sensor configuration to use, for both the distance sensors and photo transistors. To do this an initial round of brainstorming was done where a number of different configurations were presented, and the pros and cons of each layout was compared and considered. Some of the early ideas included arranging the forward facing distance sensors at various angles, and using the servo motor to scan the arena for fires, and other ideas such as mapping out the course to ensure we never backtracked on previously explored areas to ensure an optimised fire-seeking behaviour. After this round of brainstorming was done, each of the ideas were compared, and it was decided to pursue mapping the course through the use of extended kalman filters, along with fuzzy logic control, and three forward and 2 side facing sensors.

5.2 Agile Techniques

The project was organized to run in 3 week cycles where a prototype would be created, evaluated and improved upon although these time frames were not able to be achieved exactly there were still two sprints enacted. The first prototype for this system was to create a seek function which utilised a map of the area created by scanning the surroundings. This map would be used to avoid obstacles and for the robot to find its way to the fire. Ultimately it was found that the programme was difficult to make work and unreliable when executing so the next prototype simplified the control scheme into a behaviour based control system. This system utilised a control loop which would put the system into its four main states. These four main states were the avoid, finished, extinguishing and seeking states mentioned previously and this overall design was ultimately utilised in the final demonstration. The second sprint then had minor improvements added to refine the design. The avoid function was remodelled to make use of counters to keep the robot moving completely out of the way and was simplified to use functions to check if there were objects in the way. These functions measured the sensors and made the code more readable and easier to debug. Each sprint improved the reliability and performance of the system and Agile product development was put to good use in this project.

5.3 Project Management Software

To facilitate the management of the team project, the adoption of a project management software was deemed necessary. Jira Software was selected as the chosen tool, primarily due to its comprehensive planning and road mapping capabilities. This software enables teams to proactively outline and organize tasks, subsequently assigning them to respective team members for execution. Furthermore, Jira Software's compatibility with agile project management methodologies proved crucial in the effective handling of a project characterized by its dynamic and iterative nature. The Gantt chart utilized throughout the project can be seen in Figure 12.

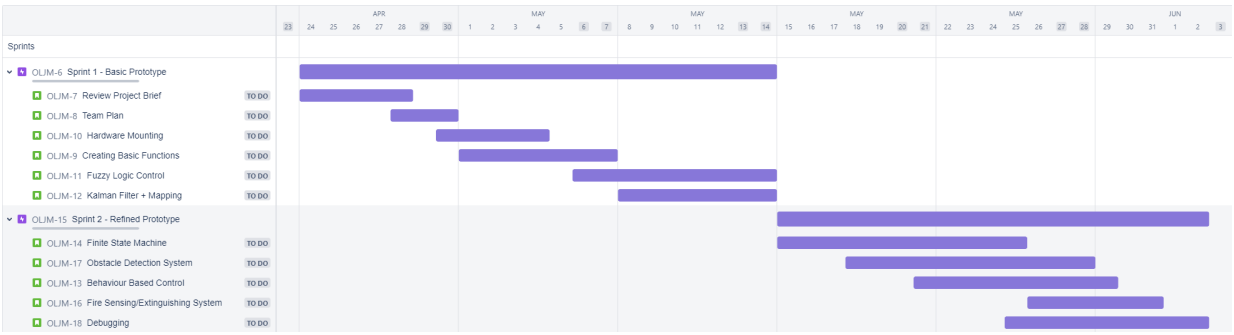


Figure 12: Jira Gantt Chart

5.4 Individual Contributions

Fraser: Contribution to coding mostly, writing the majority of the control loop and working on the avoid and seeking functions to fix edge cases and debug as well as testing of the robot.

Bhumik: Integration and wiring of fire sensing and extinguishing system (fan and photo transistors). Optimizing timer interrupt function for better obstacle detection and work on control function.

Peter: Writing the timer-interrupt based sensor polling code, writing the obstacle avoidance and fire seeking functions. Testing the robot and debugging edge cases that occurred in each of the functions.

Trevor: