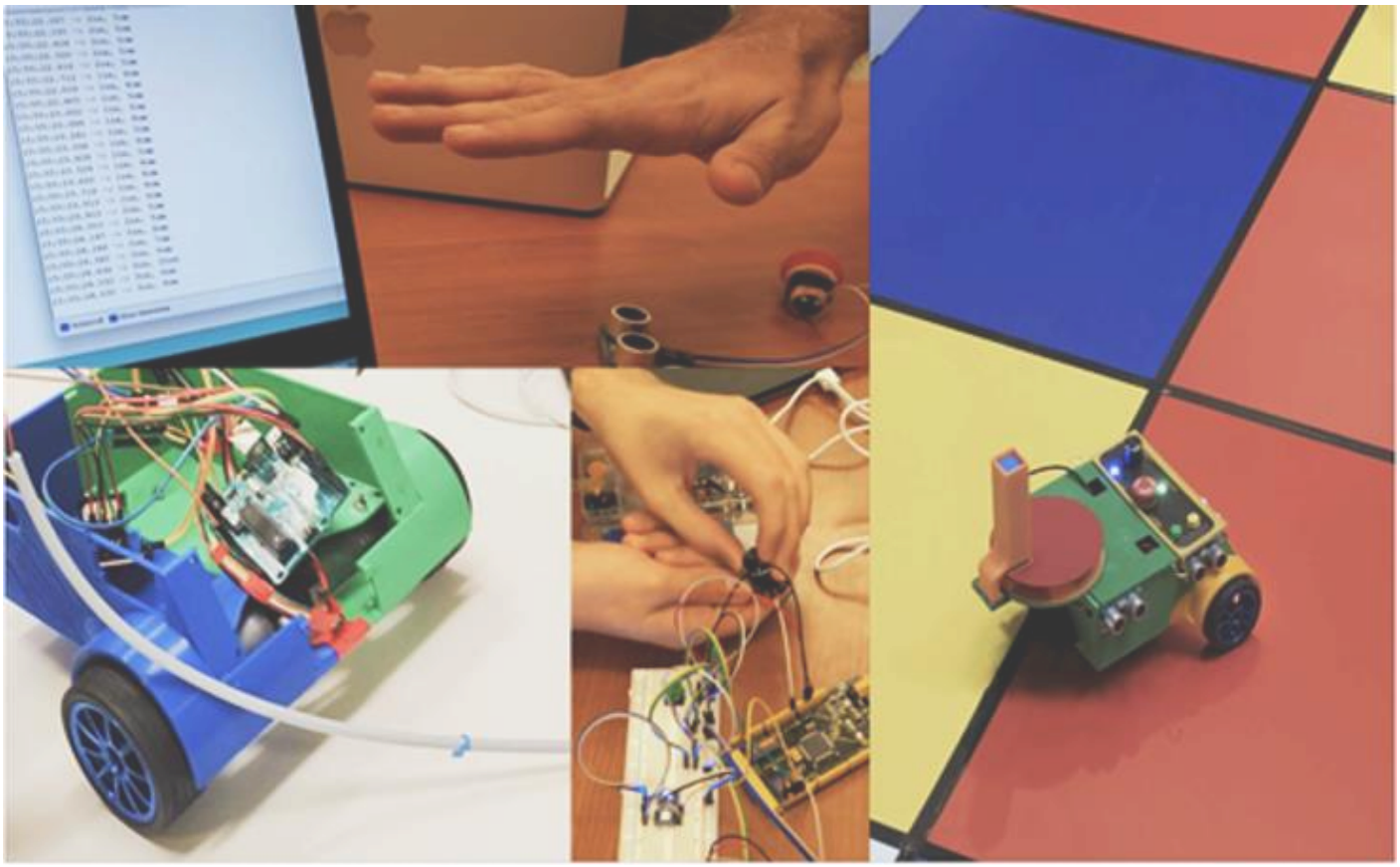
ME – 588 Final Project Report



Team – 1 Submission

Members:

James Baxter
Prithvi Prasad
Anudeep Sharma
Yidan Ding
Matthew Konishi

Abstract

The main aim of the project was to build a robot which could autonomously traverse a designated playing field, and place an object (Mole-Whacker) on each square of a designated color. The autonomous navigation system uses sensors and encoders (odometry) to traverse a predetermined path, and a wall-following mechanism to estimate vehicle position and perform course correction. This robot used Finite State Machine logic, which provided the necessary intelligence to move around the playing field, and drop Mole-Whackers on the appropriate squares. Close-loop system with PID control was attempted to ensure that the robot followed the designated course, while the FSM logic enabled the robot to decide which actions to take, which would allow it to accomplish the goal of placing one Mole-Wacker on each square of a designated color. Complications resulting from faulty equipment resulted in inconsistent performance of the robot. As such, the robot was able to complete the Final Check-Off, but was changed at the last minute in an attempt to improve consistency. Two new motors were substituted in to the robot, which resulted in decreased performance due to large amounts of internal friction present in one of the motors.

Introduction

Autonomous Guided Vehicles (AGV) are replacing many tedious and repetitive tasks such as mail delivery, warehouse forklifting, on-road navigation, and more. These are completely re-programmable and can be customized to perform one or more tasks by simply uploading a different software program. However, it becomes important to consider all possible scenarios and external events that the AGV might face and prepare it for the same while programming the AGV. This project is a demonstration of developing a AGV from ground up that is tailored to perform a specific task, that is to drop mole-whackers at correct colored square tiles on the playing field.

Whack – A – Mole is a game in which a robot traverses a playing field and drops mole-whackers (foam balls or cubes) on squares with Moles. Target squares (where Moles are located) are identified by color. Starting in a designated start square, each robot must go around the playing field dropping mole – whackers at the designated color, and return at the starting position. The fastest robot to do so wins.

Playing Field. The playing field is comprised of 16 squares, colored red, green, and blue, with the exception of the starting square, which is white. As shown in Figure 1, the playing field is a square plywood surface (8 ft. x 8 ft). Six inch foam walls surround the playing field, and ¾ inch width lines separate each square, which are created with black electrical tape. The color distribution shown in Figure 1 was not guaranteed for the competition, and yellow squares were used in lieu of green squares.

The Mole-Whackers were foam balls or cubes. The foam balls were ¾” in diameter, and the foam cubes had a side length of ¾".

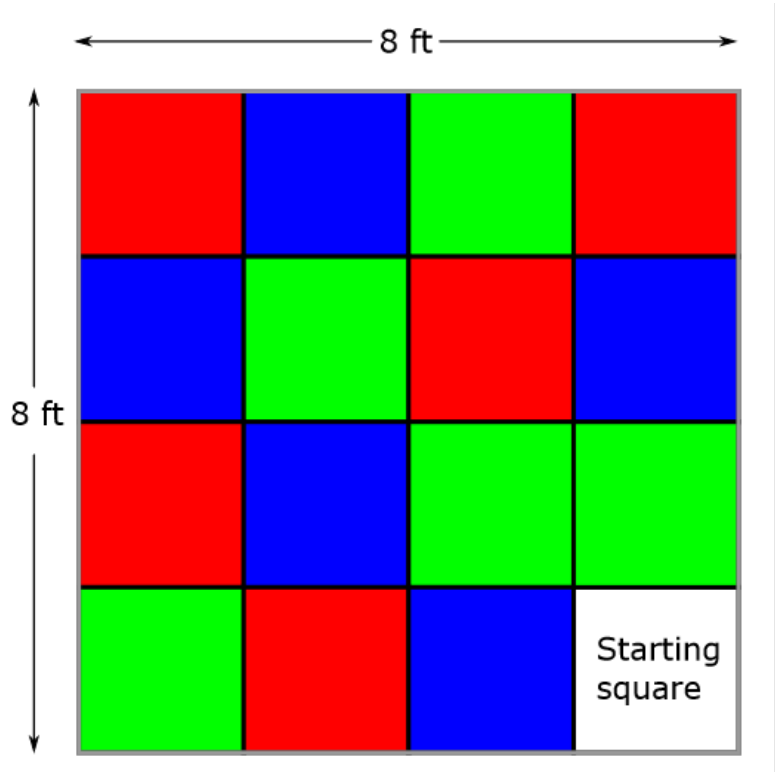


Figure 1: Top view of the playing field.

Game Description. Before the game was started, Mole-Whackers were loaded into the robot, and code was uploaded and started. The robot was to be placed within the starting square in any orientation, and the target color was input to the robot (as designated by the TA). The robot was then be started (via a button or other input), and navigated the field autonomously and place Mole-Whackers on squares of the target color. The round ended once the robot returned to the starting square, or after 2 minutes had elapsed.

General Strategy. We decided to use a fixed search path as shown in Figure 2. By following the black line between two tiles, we're able to detect and deploy Mole-Whackers on either side of the robot to increase speed. One ultrasonic sensor at the front of the robot and two placed at the right side are used to make robot follow the wall at a fixed distance and to track distance to the front wall. To drop Mole-Whackers on both sides, two color sensors are installed on both sides of the robot for detection, and one servo motor located at center is controlled to spin to the left or right to make the corresponding deployment. In the next section, we will describe our design to achieve this strategy in more detail.

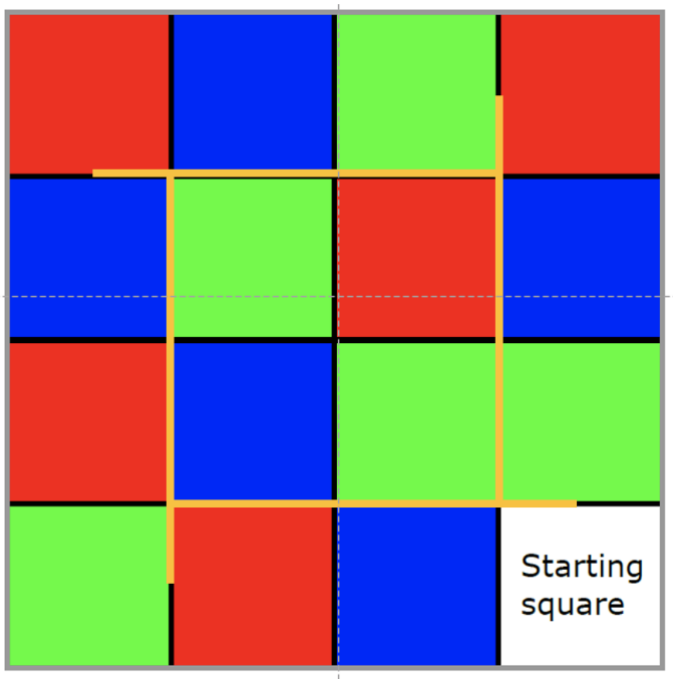


Figure 2: Predetermined fixed path

Robot Design

Before any possible testing and modification on operation could be done, the goal was to come up with a solid design which was broken up into 3 parts: Mechanical Hardware Design, Electrical Design and Code.

Mechanical Hardware Design

The design consists of a 3-wheeled robot that that would maneuver the playing field, motorized on only two front wheels. The 3rd wheel was chosen to be a castor wheel, which was left for support and smooth turning. The support of the vehicle was a single level chassis that stored the battery, Arduino, and all other electrical components of the robot.

The entire chassis of the robot was 3-D Printed to reduce the weight as much as possible. This enables faster translation speeds, reduced load on the motors, and increased battery life. 3-D printing also gave us the option of rapid prototyping in case of design failures or modifications due to mechanical errors or miscalculation of circuitry parameters.

There are few important reasons making us go with in the 3D printing – custom chassis route:

- Elimination of bread board (saves weight and prevents extra wiring mess).
- Creating a wire harness calculated based on the components' relative distances.
- Tighter packaging, thereby reducing overall dimensions. This helps in easier cornering and quicker path correction.

- All dimensions of custom parts (Appendix B4-B6) are modelled after we receive the component with suitable tolerances, making the parts direct fit into the chassis. Also taking into account the design for assembly (DFA).
- Save enormous weight by eliminating aluminum/plexiglass for chassis. Our chassis weighted at about 140grams. Which is lighter than aluminum but heavier than plexiglass. On the other hand, our model had much higher torsional rigidity and strength compared to plexiglass. Making the best possible strength to weight ratio.
- The custom mole whacker mechanism had a quick dispensing due to the very light mechanism(B5).
- A switch board was designed as per our needs, this helps in reducing clutter and quick button operation. Also eases wire management, as buttons are placed in proximity to the Arduino (B6).
- No glue/tape was used, industrial design using fixtures and nuts and bolts, this made the product more realistic. All components sit in designated slots, they do not wobble around as all degrees of motions are constrained through chassis or nuts and bolts.

The Mole-Whacker distribution was incorporated to the rear of the chassis, and consisted of a gravity-fed hopper and a rotating disc. The disc was designed such that one Mole-Whacker could fit inside of it at a time, and confining the remaining Mole-Whackers inside of the hopper while the disc was rotating. The disc would rotate on command toward one side of the robot (right or left) until it reached the dispensing slot, where the Mole-Whacker would fall through and tumble towards the target square.

These critical hardware design decisions make our robot a truly **plug and play** setup. It saved us time in trouble shooting and replacing components when necessary. The final design and CAD drawings are included in Appendix B.

Electrical Circuitry

The powering of the entire robot can be divided into 5 major sub-systems:

Drive Mechanism:

The drive mechanism was comprised of powering two motors, one for each of the front wheels. The left and right motor were driven by an L298 H-bridge, which was powered by the 12v battery. The H-bridge was controlled by signals sent from the Arduino as specified by the FSM code.

Mole-Whacker Distribution

A servo motor powered by the Arduino Mega was used to dispense mole whackers in two directions. Based on the location of the target square, the servo turned left or right to drop the mole whackers, then returned to the center.

Main power distribution

A 12 V battery powered the Arduino Mega and the motor driver. All the other components were powered by the Arduino.

Robot Display

The robot displays were powered from the Arduino Mega. The LCD display, as well as the color sensors, were connected via an I2C Multiplexer. Fig. A.4 shows the circuit arrangement of this sub-system. There are two buttons for user input, one to start the game, and one for emergency stop. A potentiometer was used to change the target color. 2 RGB LEDs were used to show the robot state (ON/OFF), as well as the target color. The LCD was used to show the current FSM state, speed, and sensor information.

Sensor conditioning

There were a variety of sensors used in the robot to derive information based on its' environment. All of them were powered via Arduino, as is shown in Fig. A.5. Two color sensors with I2C communication relate to the Arduino through I2C Multiplexer. Three ultrasonic sensors are used for distance measurement. One is located at the front and two are at the right side for wall following.

Code (Intelligence Logic):

The finite state machine was the primary means of controlling the robot's actions throughout the task. Figure 3 shows all the possible transition states of the robot, and their transitions. The robot travels the predetermined path until the correct color is detected, where it dispenses the Mole-Whacker before continuing along the path. To avoid checking a square more than once, we break down the path into four segments, as shown in Figure 4. The FSM checks each block in one segment, and then resets to the initial state once it is finished checking the entire segment.

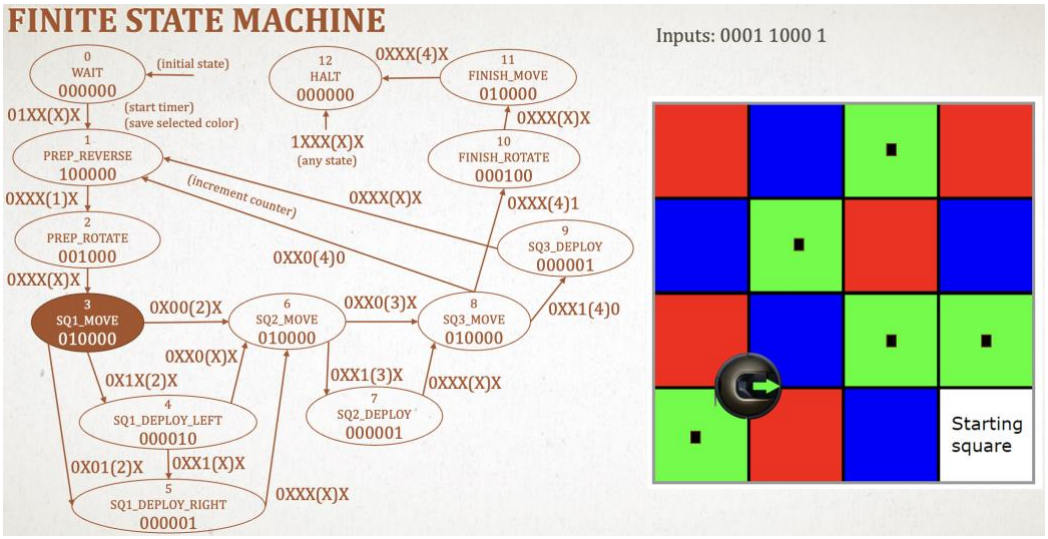


Figure 3: Final FSM Logic of the Robot Operation

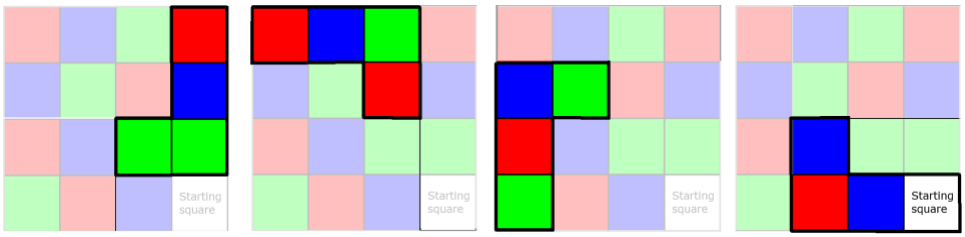


Figure 4: The playing field is subdivided into four segments for our FSM

Moving to the center of two new squares, the robot would stop and the sensors on either side would start detecting the colors. As soon as the color sensor detected the target color, the mole-whacker motor initiated the dispensing mechanism. After the servo motor rotates back to its initial position, the robot resumed driving around the path. If no target color is detected, the robot resumed driving after a predetermined time. The schematic of mole-whacker in Fig. B5 shows how the design of the Mole-Whacker dispenser dropped the cube after actuation.
To make sure the robot is able to follow the predetermined path, wall following was used to provide feedback to the robot if it drove off-course. This was done by taking sensor readings from 2 Ultrasonic sensors located on the side of the robot facing the wall, as shown in Figure B3. By using the error between the two distance sensors, we could correct the path of the robot by adjusting the effort given to the motors. This was done using a PID controller. The ultrasonic sensor located at the front is used to determine when to take turns.
After going through the last segment, the robot returns to the starting square and ends the game.

Results

While each component of the robot worked individually, we ran into several issues when combining them into one singular unit. Our initial wall-following strategy included the use of two IR sensors for detecting the distance from the side wall, and one ultrasonic sensor for detecting the distance from the forward wall. IR sensors were chosen for the side sensors as we wanted to reduce any noise-related issues, and the forward ultrasonic sensor was chosen as the rated range was greater than that of the IR sensors (due to our pathing strategy, we remained approximately 2 feet away from the side walls, while the distance to the forward wall could increase to upwards of 7.5 feet). However, we immediately found that the IR sensors had considerable noise at a distance of 2 feet, despite the claims of accurate measurement to distances up to 2.6 feet. Thus, we replaced the IR sensors with ultrasonic sensors. Unfortunately, we also found that our two ultrasonic sensors did not work well together due to interference from the ultrasonic pulses. Thus, we needed to limit the usage of these sensors, which will be discussed further in the Discussion section.

One of the most significant challenges that we faced throughout this project was to force our robot to drive straight. Originally, we believed that we had received two of the same type of motor from the collection of spare parts from previous competitions. Not only did these two motors have drastically different encoder resolutions, but the motion profiles were completely different. Several different control strategies and other methods were tried to keep our robot on course, with limited success. After trying multiple different methods, we were able to get a functioning robot by utilizing only motor encoders, which allowed us to successfully complete the final check-off. However, the robot function was still inconsistent, and so we

decided to replace the motors with a new set of motors. Unfortunately, robot performance decreased when using the new motors, as one of the two newly purchased motors had significant internal friction which caused it to stop much quicker than the other motor. As a result, our robot turned 30 to 45 degrees every time it stopped. This was too big of an issue for us to correct before the competition, resulting in a robot that struggled to stay on the designated path.

Two of the systems which worked flawlessly were the color-sensing mechanism and Mole-Whacker distribution. When our robot found itself on a designated mole location, the robot was able to correctly identify if the color matched the target color and distributed the mole in a timely manner. Our method of identifying the correct colors using a simple RGB threshold was robust enough to reject other colors while still correctly identifying the target color. If the squares were worn or more varied in color (rather than the pristine, solid colors of the current field), a more intricate color detection system may have been necessary. Similarly, our relatively simple dispenser, which dropped a Mole-Whacker through a slot located on either side of the robot, proved to be a robust solution for this task.

All of the other required functionality was included in the final robot and worked as intended. We included two LEDs on the top of the robot, which indicated the state of the robot (green for GO, and red for STOP), as well as the current target color, which was controlled by a potentiometer. An LCD screen was included on the front of the robot, which was used to output assorted information such as wall distance and the robot's current actions. A START button and an e-stop button were also included on the top of the robot, which also worked as intended.

Discussion

For the version of the robot used in the final check-off, we decided against the use of the ultrasonic sensors. Rather, we relied on the motor encoders to determine the distance traveled. While this "blind" strategy propagates errors forward throughout the task, our robot could reliably detect the color of the square and would only dispense Mole-Whackers on the correct colors. This meant that our robot would be able to perform the task correctly near the start of the path and would not misplace Mole-Whackers later in the task if it went off course. While this is a decent method for passing the final check-off, our team decided to change this design before the final competition, as it was somewhat inconsistent. This resulted in swapping out our motors for two newly purchased motors, one of which had the aforementioned problem with internal friction.

While this new motor set had similar problems as the previous set of motors (differing motion profiles), it also introduced a new problem where the robot would turn 15 to 45 degrees every time it stopped. Without some form of feedback, an error of this magnitude ruins the pathing of the robot. Thus, we incorporated one of the two ultrasonic sensors to detect the distance from the wall. While this worked when the initial trajectory was only slightly off course, it failed whenever the robot drove at a large angle relative to the wall, as the ultrasonic sensor was unable to receive accurate readings. As we could not control the angle to which the robot turned after each stop, this version of the robot performed worse than our previous version; each time the robot stopped, there was a good chance that it would turn far enough away from the wall that it would drive off course.

During the final check-off, and to a lesser extent, the final competition, the saving grace of our robot was the robot color-detection scheme and Mole-Whacker distribution system. A robust system allowed our robot to drive off course, in a somewhat random manner, without misplacing any Mole-Whackers. Any time that our robot happened across a correctly colored square, it would drop a Mole-Whacker without incident. In a typical run with our best-performing robot, it would start out along the path correctly, and slowly deviate from our predetermined path. This would result in the correct placement of Mole-Whackers on target squares early in the path, while squares later on in the path were often left unmarked, unless the robot randomly came across it after deviating from the path. Squares that were not the target color were usually left unmarked.

While the color-detection and Mole-Whacker distribution systems were the strongest point of our robot, neither one of these systems worked correctly after our first attempt at building them. For the Mole-Whacker distribution system, we encountered a tolerance issue with the whole assembly. Inside of the disk, as well as the gravity-fed hopper, the Mole-Whackers tended to stick, as the tolerances were too tight. The constraint of the 180 deg rotation range of the servo motor also makes the initial design not able to work, therefore the place of two outlets were changed in later design. The entire chassis and Mole-Whacker distribution system was adjusted and reprinted to fix this issue. The color-detection system initially malfunctioned due to having the same unchangeable I2C address from the manufacturer. At first, we tried a software-based solution to no avail. Once we purchased and integrated the Adafruit TCA9548A Multiplexer into our system, we were able to use both together with no issue.

Conclusion

In conclusion, we developed a robot that was able to compete in the Whack – A – Mole competition and that cleared all the requirements outlined in the Final Check-Off. In testing sessions and the Final Check-Off, the robot showed the ability to autonomously navigate through the playing field, recognize the color of the target squares, drop Mole-Whackers at those desired locations, and return to the home position. Changes made immediately before the final competition significantly reduced the functionality of the robot, which resulted in a poor performance during the competition. Regardless, the robot was able to complete the task, albeit inconsistently, at one point in time. It is our opinion that a third attempt at finding any two remotely similar motors would result in a nearly perfectly functioning robot, as all the other robot functionality worked well. Perhaps a different sensor setup, one on each side, for example, would also improve our robot design.

The development of this robot demonstrates integration of multiple qualities of robotic system that are independently useful, and by completing this project we have obtained significant hands-on experience in the integrated, multidisciplinary field of mechatronic design. This not only enabled us to showcase what could have been possible had hardware constraints not been there, but also demonstrated the scope of autonomous systems in times to come.

Appendix – A: Electrical Schematics

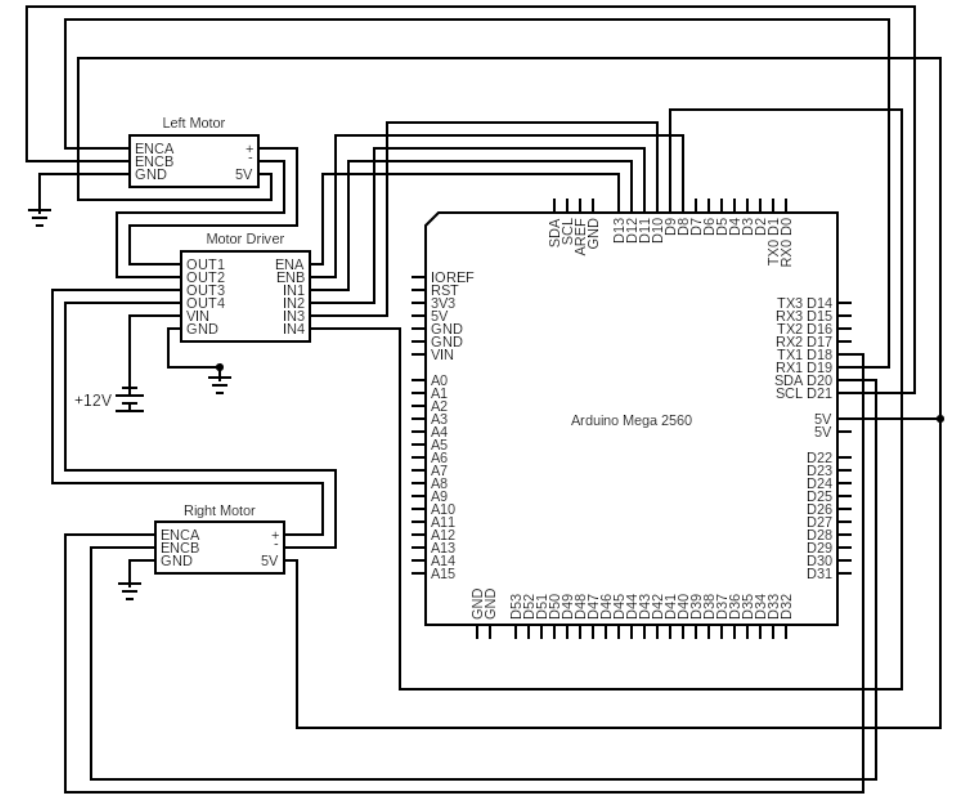


Fig. A1: Drive Mechanism

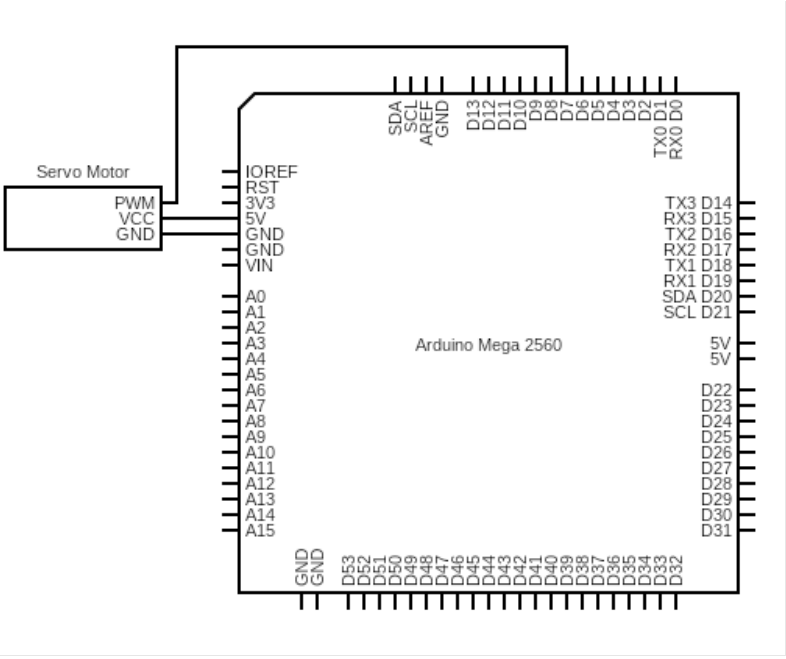


Fig. A2: Mole Distribution

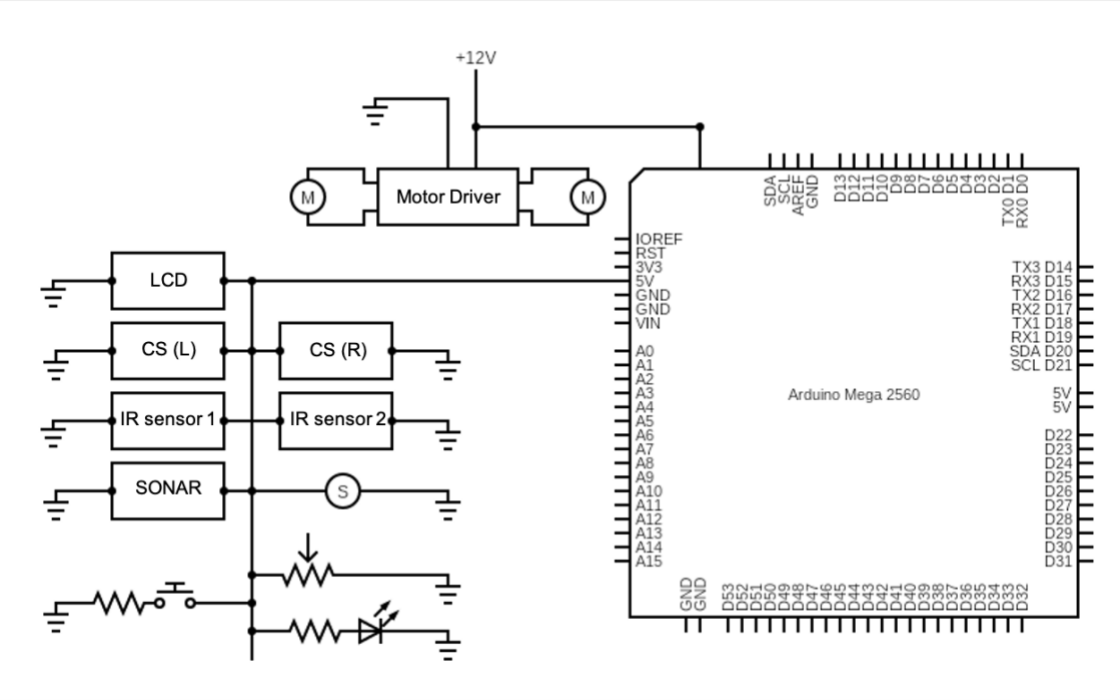


Fig. A3: Power Distribution

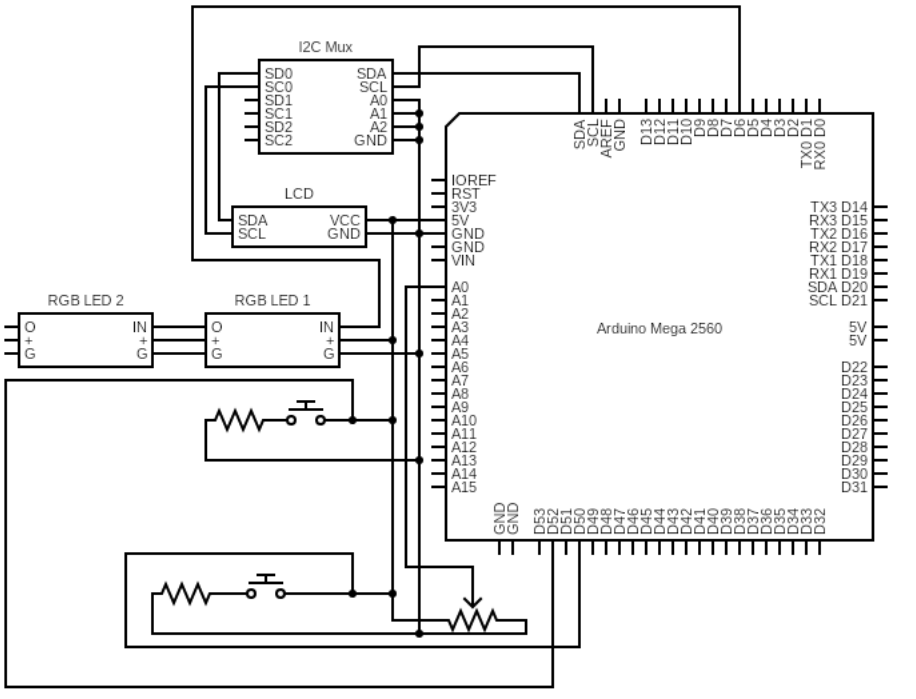


Fig. A4: Robot Display

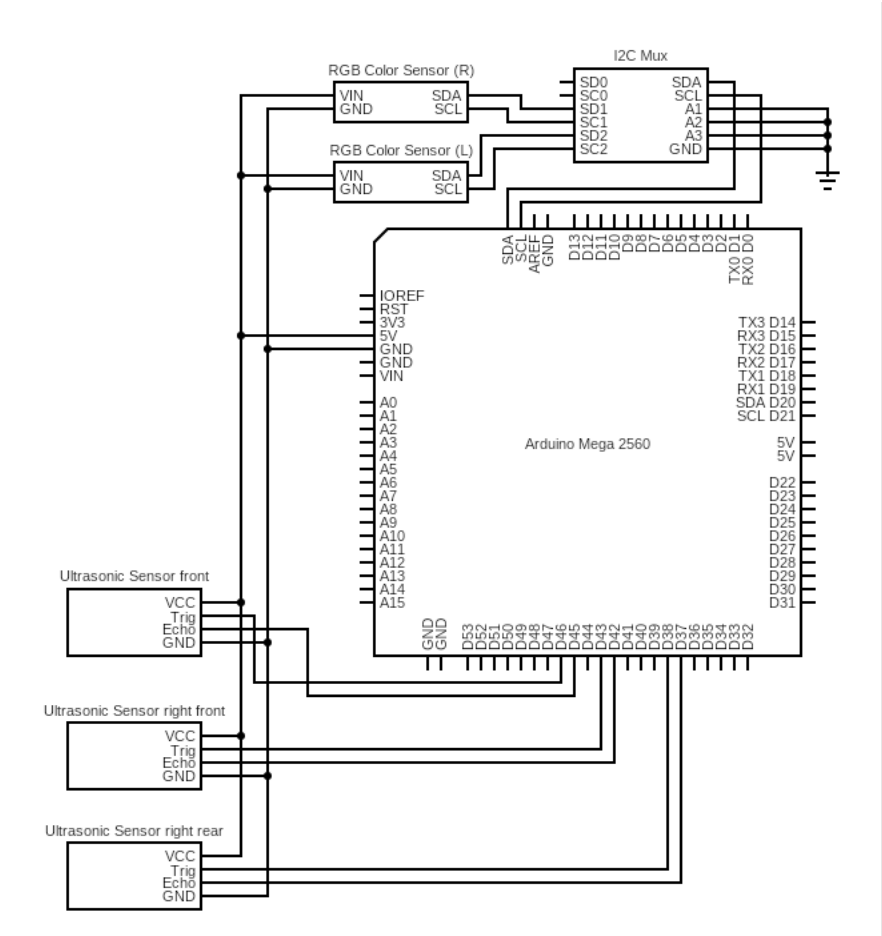


Fig. A5: Sensor Conditioning

Appendix – B: (Hardware Design)

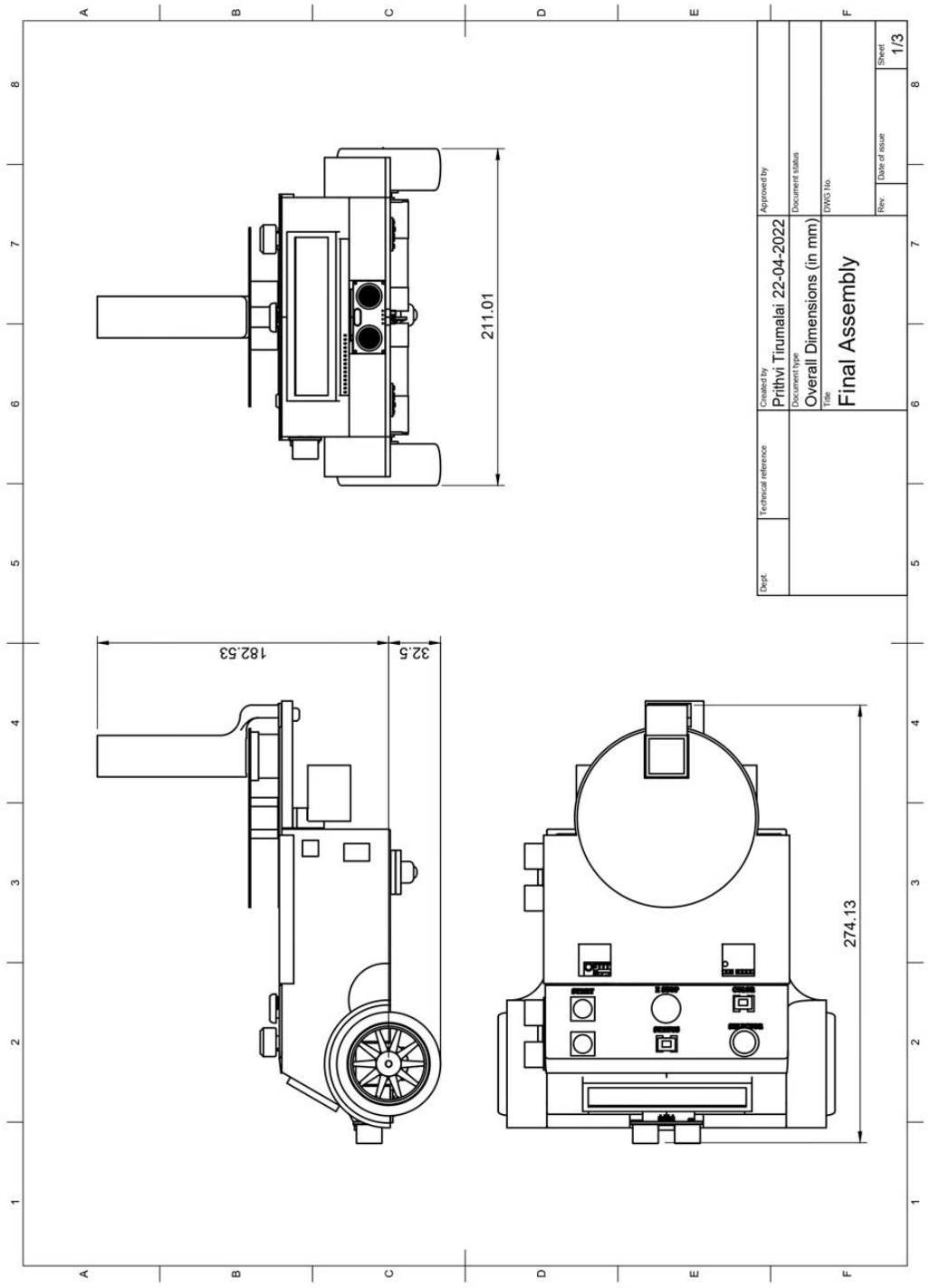


Fig. B1: Overall dimensions

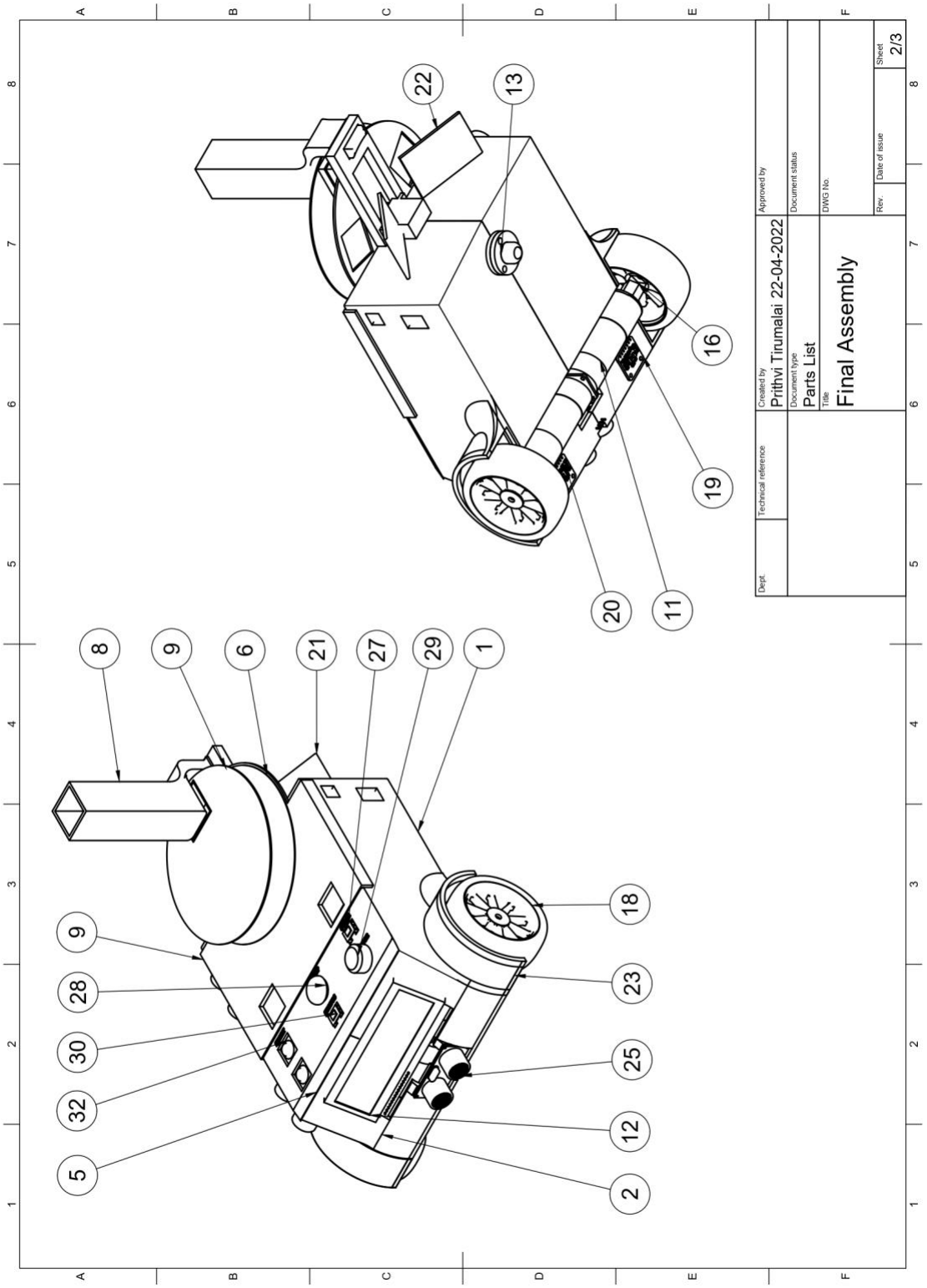
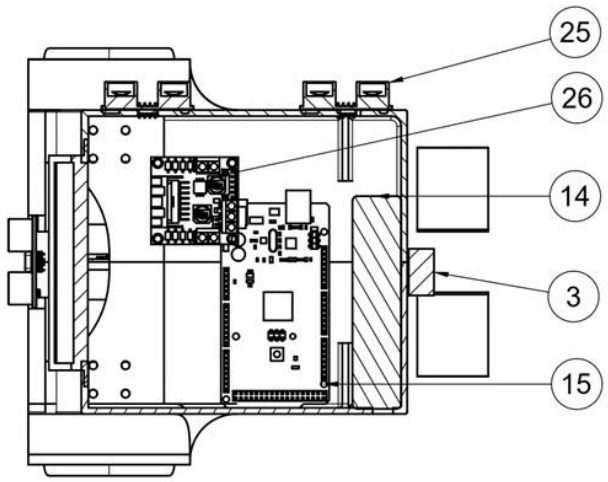


Fig. B2: Components labeled



Parts List			
Part Number	**Qty**	**Name**	**Material**
1	1	LH Frame	PLA
2	1	RH Frame	PLA
3	1	SG90 Microservo	NA
4	1	Servo Arm	Plastic
5	1	Switch Board	PLA
6	1	Mole Dropper Disc Bottom	PLA
7	1	Mole Dropper Disc Top	PLA
8	1	Mole Tower	PLA
9	1	Mole Droppper Base	PLA
10	2	Motor bracket	Aluminium
11	2	Polulu 4887 DC motor	NA

Parts List			
Part Number	**Qty**	**Name**	**Material**
12	1	16x2LCD	NA
13	1	Caster wheel	Steel
14	1	Battery	LiPo
15	1	Arduino Mega	Micro Controller
16 & 17	2	Wheel Coupling	PLA
18	1	Generic Wheel & Tire	Rubber Plastic
19 & 20	2	TCS34725_rgb_sensor	PCB
21 & 22	2	Mole Slide	PLA
23 & 24	2	Wheel Arch	PLA
25	3	Ultrasonic Sensor HC SR04	PCB
26	1	Motor Driver L298N	PCB
27 & 30	2	RGB LED	NA
28	1	E Stop	Plastic
29	1	Potentiometer	NA
31 & 32	1	Push Button	Plastic

Dept.	Technical reference	Created by	Approved by
		Prithvi Tirumalai 22-04-2022	
		Document type: Parts list	Document status
		Title: Final Assembly	DWG No.
			Rev. / Date of issue / Sheet: 3/3

Fig. B3: Internal components & Parts list

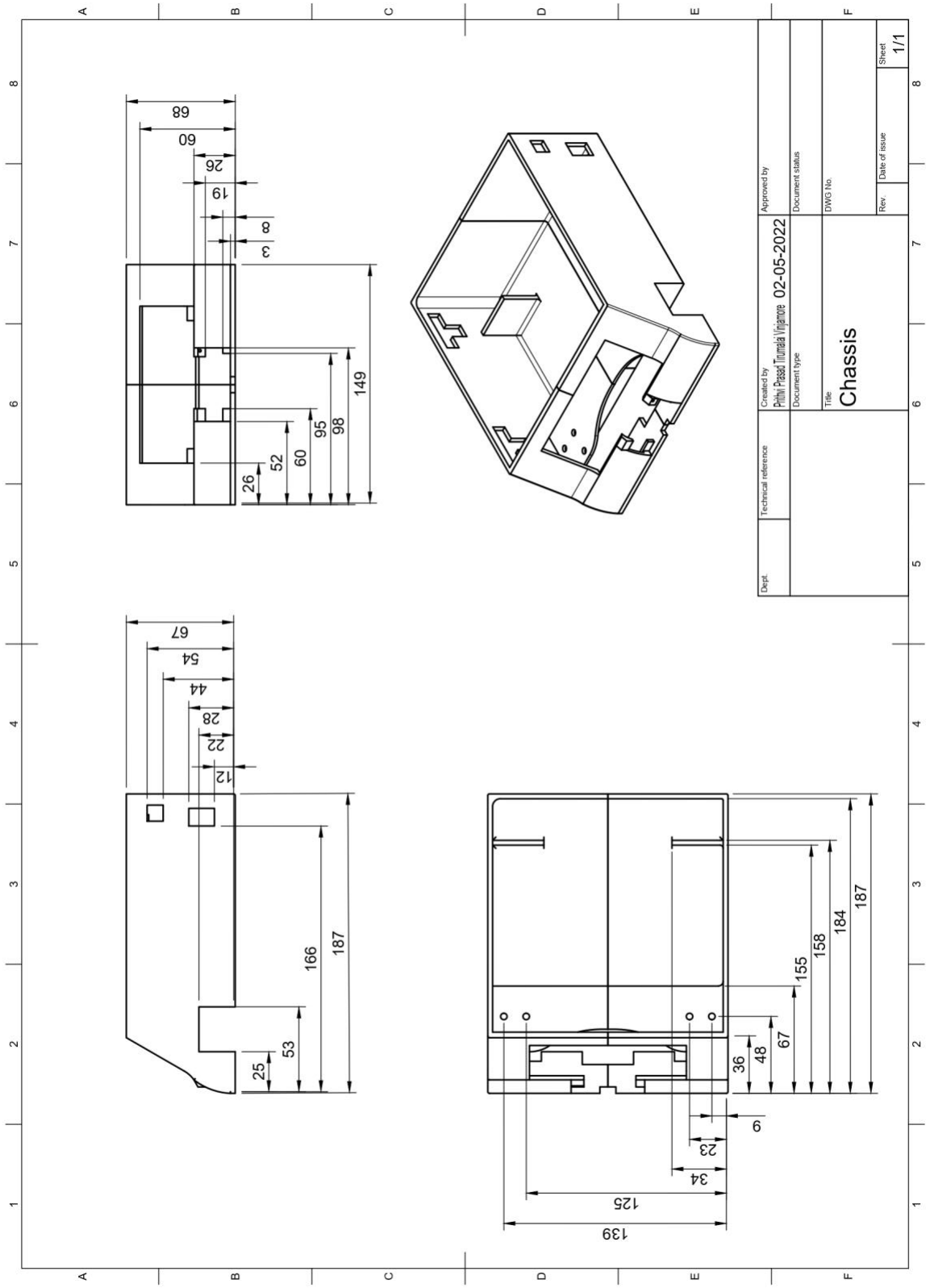


Fig. B4: Custom part - Chassis

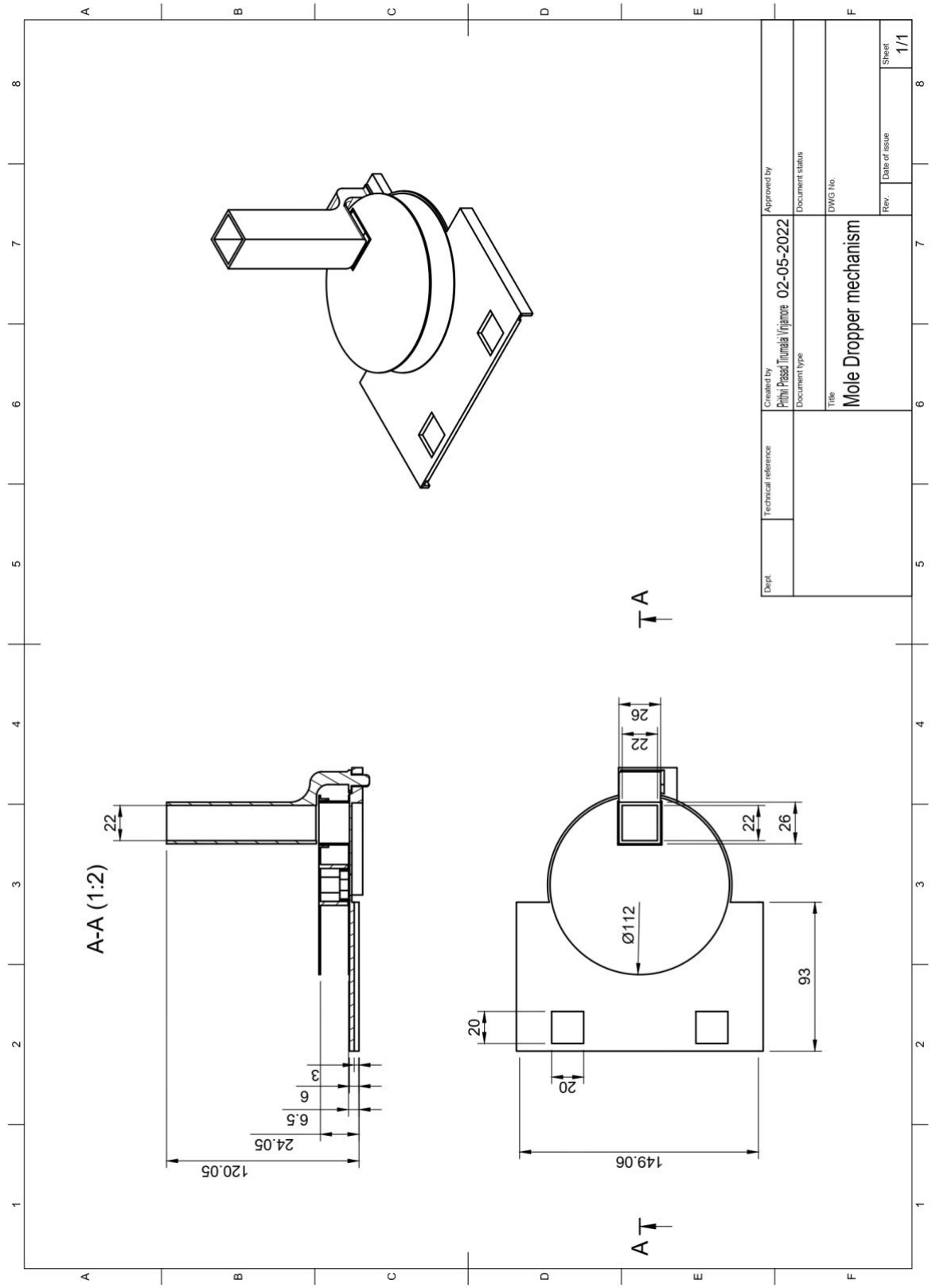
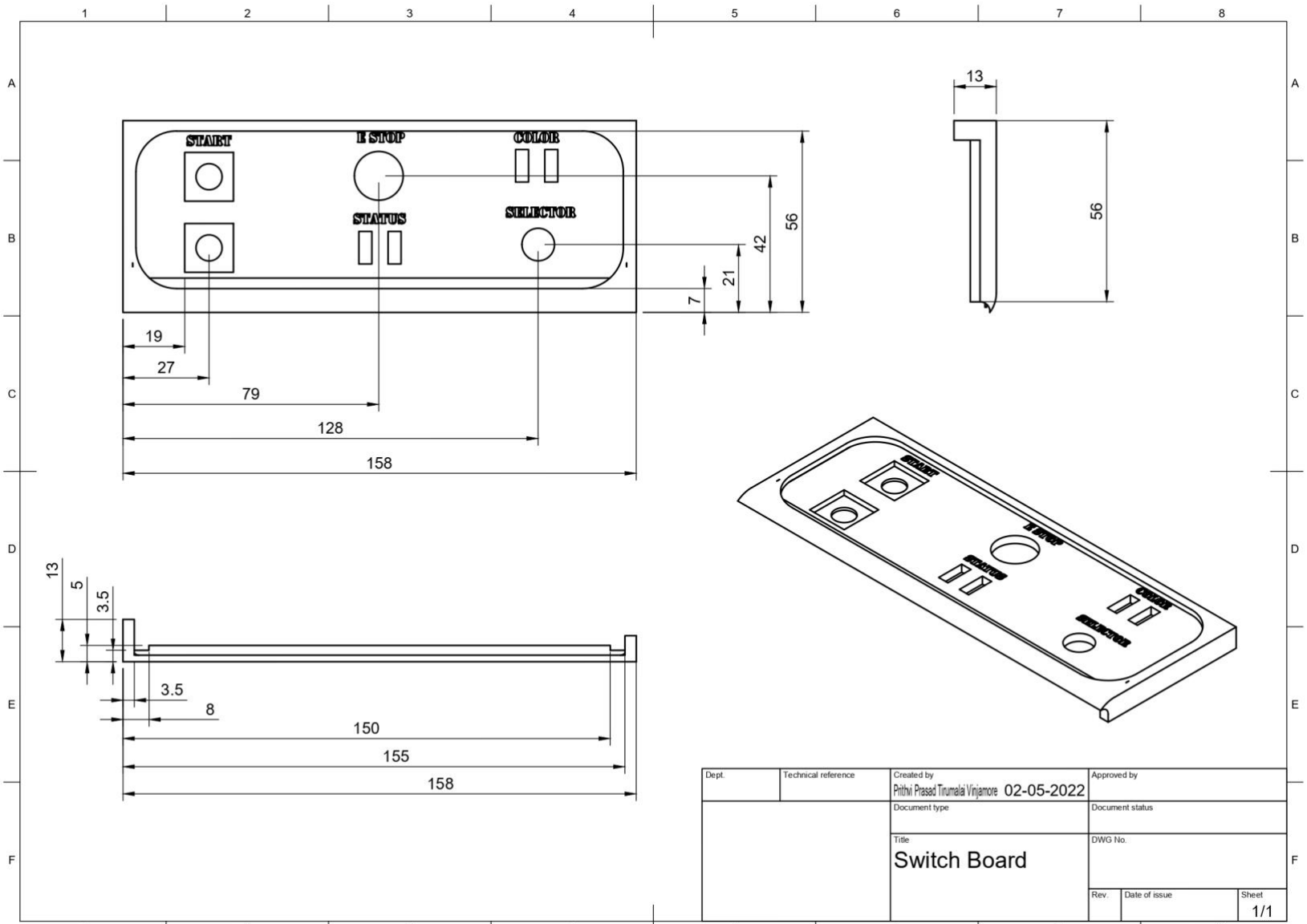
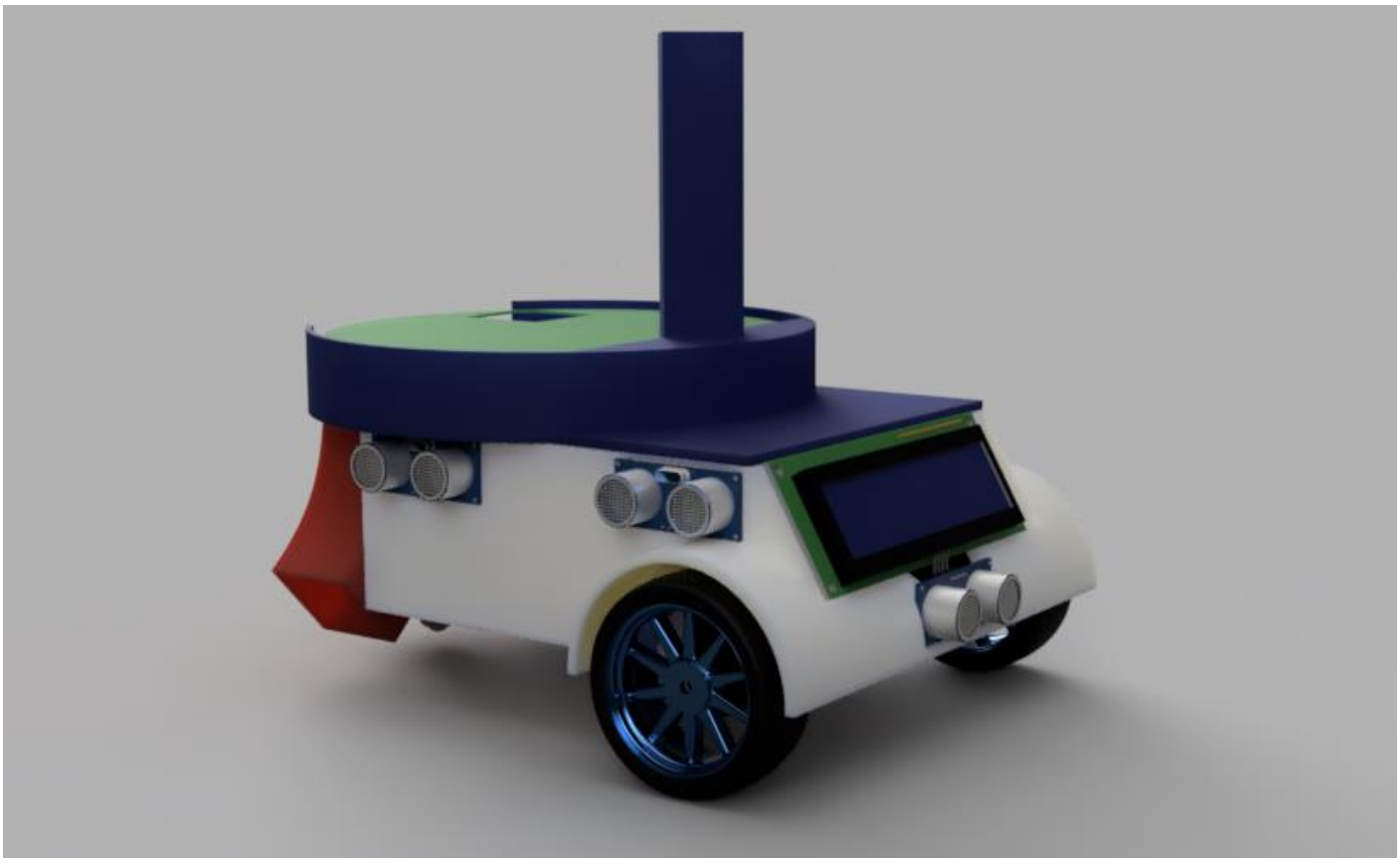


Fig. B5: Custom part – Mole dropping mechanism

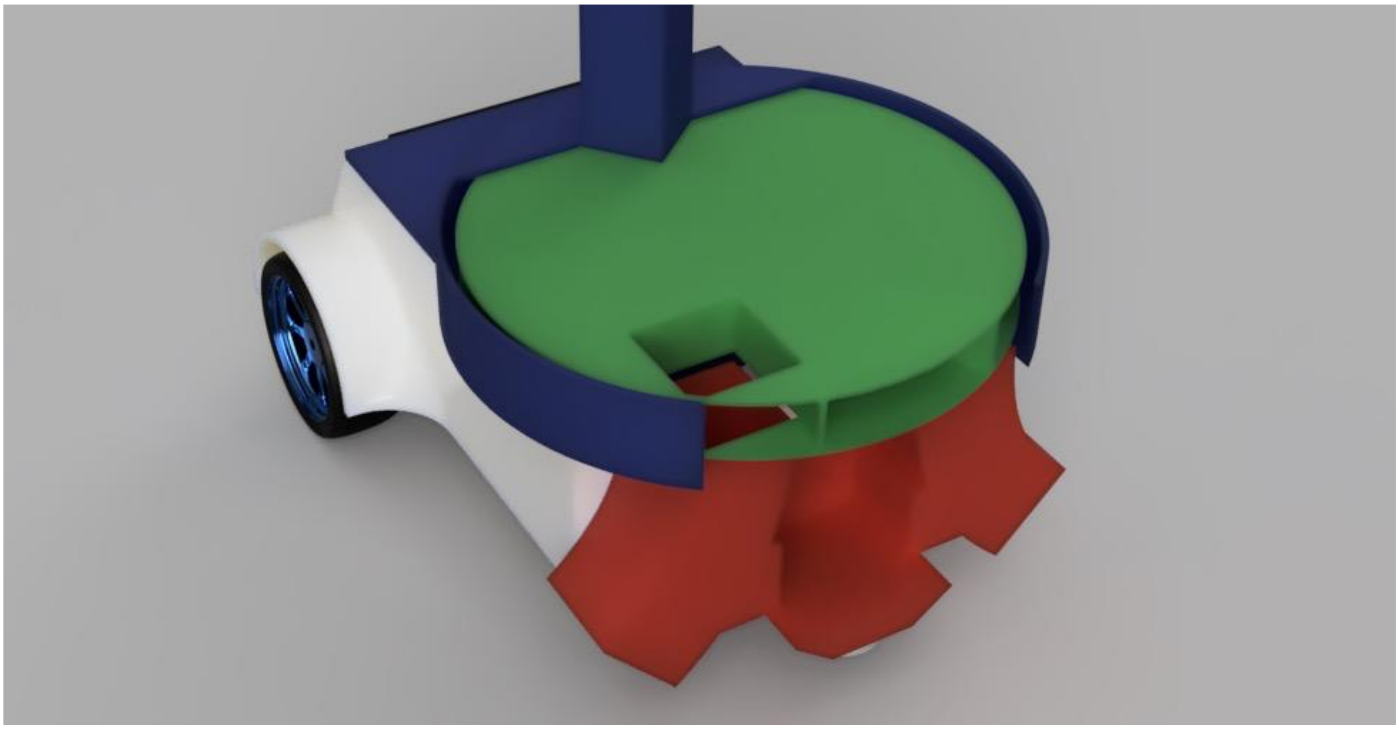


Dept.	Technical reference	Created by	Approved by
		Prithvi Prasad Tirumalai Vinjamore 02-05-2022	
		Document type	Document status
		Title: Switch Board	DWG No.
			Rev. / Date of issue / Sheet: 1/1

Fig. B6: Custom part – switch board



CAD render of Robot Layout



Mole – Whacker Dispenser Mechanism

Appendix – C (Prominent Code Logic)

```
40    // run FSM
41    if (BTN_STOP)
42      state = HALT;
43    switch (state) {
44      case WAIT:
45        if (BTN_GO)
46          state = PREP_REVERSE;
47        break;
48      case PREP_REVERSE:
49        if (DIST_PREP)
50          state = PREP_ROTATE;
51        break;
52      case PREP_ROTATE:
53        state = SQ1_MOVE;
54        break;
55      case SQ1_MOVE:
56        if (DIST_SQ1) {
57          if (COLOR_LEFT)
58            state = SQ1_DEPLOY_LEFT;
59          else if (COLOR_RIGHT)
60            state = SQ1_DEPLOY_RIGHT;
61          else
62            state = SQ2_MOVE;
63        }
64        break;
65      case SQ1_DEPLOY_LEFT:
66        if (COLOR_RIGHT)
67          state = SQ1_DEPLOY_RIGHT;
68        else
69          state = SQ2_MOVE;
70        break;
71      case SQ1_DEPLOY_RIGHT:
72        state = SQ2_MOVE;
73        break;
74      case SQ2_MOVE:
75        if (DIST_SQ2) {
76          if (COLOR_RIGHT)
77            state =  SQ2_DEPLOY;
78          else
```

```
            state = SQ3_MOVE;
        }
        break;
      case SQ2_DEPLOY:
        state = SQ3_MOVE;
        break;
      case SQ3_MOVE:
        if (DIST_SQ3) {
          if (COUNTER)
            state = FINISH_ROTATE;
          else if (COLOR_RIGHT)
            state = SQ3_DEPLOY;
          else {
            state = PREP_REVERSE;
            count++;
          }
        }
        break;
      case SQ3_DEPLOY:
        state = PREP_REVERSE;
        count++;
        break;
      case FINISH_ROTATE:
        state = FINISH_MOVE;
        break;
      case FINISH_MOVE:
        if (DIST_SQ3)
          state = HALT;
        break;
    }

    // get outputs
    const bool MOVE_REVERSE = state == PREP_REVERSE;
    const bool MOVE_FORWARD = state == SQ1_MOVE || state == SQ2_MOVE || state == SQ3_MOVE || state == FINISH_MOVE;
    const bool ROTATE_LEFT = state == PREP_ROTATE;
    const bool ROTATE_RIGHT = state == FINISH_ROTATE;
    const bool DEPLOY_LEFT = state == SQ1_DEPLOY_LEFT;
    const bool DEPLOY_RIGHT = state == SQ1_DEPLOY_RIGHT || state == SQ2_DEPLOY || state == SQ3_DEPLOY;

```

Fig. C1: FSM Logic

```
36  void loop() {
37    // read buttons
38    const bool estop_state = digitalRead(PIN_ESTOP) == HIGH;
39    const bool start_state = digitalRead(PIN_START) == LOW;
40
41    // read pot
42    const int pot_state = analogRead(PIN_POT);
43    const byte selected_color = pot_state > 682 ? GREEN : pot_state > 341 ? BLUE : RED;
44
45    // read color sensors
46    byte red, green, blue;
47    uint16_t lux = tcs.getRGB(&red, &green, &blue);
48
49    if (lux > 1200) {
50      if (red > green && red > blue) {
51        pixels.clear();
52        pixels.setPixelColor(0, pixels.Color(BRIGHTNESS, 0, 0));
53        pixels.setPixelColor(1, pixels.Color(BRIGHTNESS, 0, 0));
54        pixels.show();
55      }
56      if (green > red && green > blue) {
57        pixels.clear();
58        pixels.setPixelColor(0, pixels.Color(0, BRIGHTNESS, 0));
59        pixels.setPixelColor(1, pixels.Color(0, BRIGHTNESS, 0));
60        pixels.show();
61      }
62      if (blue > red && blue > green) {
63        pixels.clear();
64        pixels.setPixelColor(0, pixels.Color(0, 0, BRIGHTNESS));
65        pixels.setPixelColor(1, pixels.Color(0, 0, BRIGHTNESS));
66        pixels.show();
67      }
68    }
69    delay(250);
70
```

Fig. C2: Sensors sense