

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LATKRABANG
SCHOOL OF ENGINEERING
ENGINEERING GROUP OF **(RAI : 06)**



01416316 – COMPUTER PROGRAMMING FOR RAI

FINAL PROJECT REPORT

INSTRUCTED BY: DR. POOM KONGHUAYROB

NAME: LUCKY, SITAPAT , JATURAPAT

GROUP NO: 6 - CIRCUIT BREAKERS

DATE OF SUB.: 11/11/2024

1. Autonomous Robot

1.1. Code

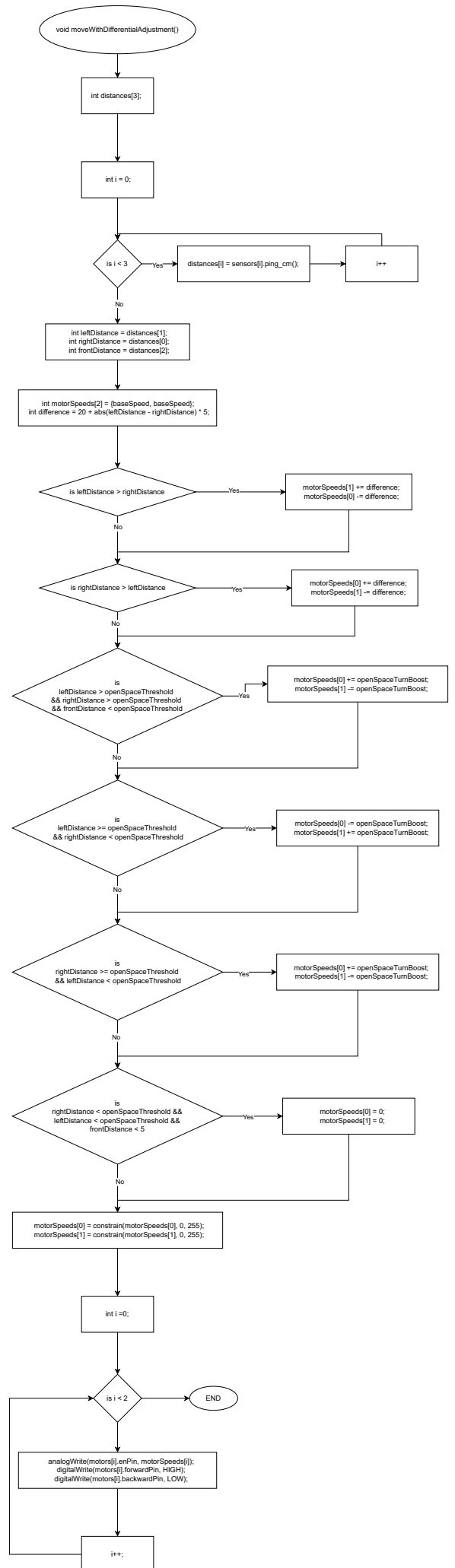
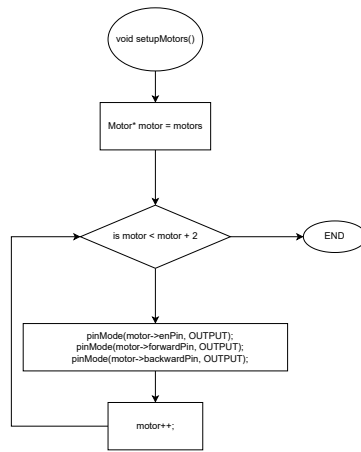
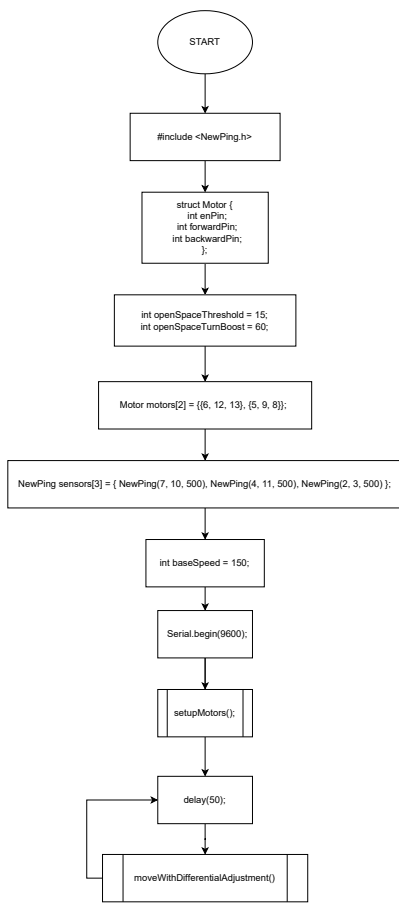
```
2. #include <NewPing.h>
3.
4. struct Motor {
5.     int enPin;
6.     int forwardPin;
7.     int backwardPin;
8. };
9.
10. int openSpaceThreshold = 15;
11. int openSpaceTurnBoost = 60;
12.
13. Motor motors[2] = {{6, 12, 13}, {5, 9, 8}};
14.
15. NewPing sensors[3] = { NewPing(7, 10, 500), NewPing(4, 11, 500), NewPing(2, 3, 500) };
16.
17. int baseSpeed = 150;
18. void setupMotors() {
19.     for (Motor* motor = motors; motor < motors + 2; motor++) {
20.         pinMode(motor->enPin, OUTPUT);
21.         pinMode(motor->forwardPin, OUTPUT);
22.         pinMode(motor->backwardPin, OUTPUT);
23.     }
24. }
25.
26. void setup() {
27.     Serial.begin(9600);
28.     setupMotors();
29. }
30.
31. void moveWithDifferentialAdjustment() {
32.     int distances[3];
33.
34.     for (int i = 0; i < 3; i++) {
35.         distances[i] = sensors[i].ping_cm();
36.     }
37.
38.     int leftDistance = distances[1];
39.     int rightDistance = distances[0];
40.     int frontDistance = distances[2];
41.
42.     int motorSpeeds[2] = {baseSpeed, baseSpeed};
43.     int difference = 20 + abs(leftDistance - rightDistance) * 5;
44.
45.     if (leftDistance > rightDistance) {
```

```

46.   motorSpeeds[1] += difference;
47.   motorSpeeds[0] -= difference;
48. } else if (rightDistance > leftDistance) {
49.   motorSpeeds[0] += difference;
50.   motorSpeeds[1] -= difference;
51. }
52.
53. if (leftDistance > openSpaceThreshold && rightDistance > openSpaceThreshold &&
    frontDistance < openSpaceThreshold) {
54.   motorSpeeds[0] += openSpaceTurnBoost;
55.   motorSpeeds[1] -= openSpaceTurnBoost;
56. } else if (leftDistance >= openSpaceThreshold && rightDistance < openSpaceThreshold)
    {
57.   motorSpeeds[0] -= openSpaceTurnBoost;
58.   motorSpeeds[1] += openSpaceTurnBoost;
59. } else if (rightDistance >= openSpaceThreshold && leftDistance < openSpaceThreshold)
    {
60.   motorSpeeds[0] += openSpaceTurnBoost;
61.   motorSpeeds[1] -= openSpaceTurnBoost;
62. } else if (rightDistance < openSpaceThreshold && leftDistance < openSpaceThreshold
    && frontDistance < 5) {
63.   motorSpeeds[0] = 0;
64.   motorSpeeds[1] = 0;
65. }
66.
67. motorSpeeds[0] = constrain(motorSpeeds[0], 0, 255);
68. motorSpeeds[1] = constrain(motorSpeeds[1], 0, 255);
69.
70. for (int i = 0; i < 2; i++) {
71.   analogWrite(motors[i].enPin, motorSpeeds[i]);
72.   digitalWrite(motors[i].forwardPin, HIGH);
73.   digitalWrite(motors[i].backwardPin, LOW);
74. }
75.}
76.
77.void loop() {
78.  delay(50);
79.  moveWithDifferentialAdjustment();
80.}
81.

```

1.2. Flowchart



1.3. Code explanation

The autonomous robot uses the `<NewPing.h>` library for ultrasonic sensor handling and a differential control strategy for navigation. It reads distances from the front, left, and right using three ultrasonic sensors and uses this data to center itself between obstacles. The `<NewPing.h>` library simplifies ultrasonic sensor integration by handling the pulse timing and distance calculation and defining the pin modes for the ultrasonic sensors as well.

The robot's code is structured around functions, structures, arrays, pointers and loops. The base speed of 150 is set to ensure that the robot does not go too fast or too slow. A `Motor` struct represents each motor with `enPin`, `forwardPin`, and `backwardPin`. Two `Motor` structs are stored in an array, `motors[2]`, allowing the robot to manage both motors using a pointer-based setup in the `setupMotors()` function. The PWM pins 5 and 6 are specifically chosen for the motors as these have a higher frequency allowing for the motors to spin smoothly. The `sensors[3]` array uses `NewPing` objects to handle each ultrasonic sensor, with each object configured to measure distances up to 500 cm for the left, right, and front positions.

In the `moveWithDifferentialAdjustment()` function, a loop iterates over the sensors array to record each distance in the `distances` array. Using these values, the robot calculates a difference between left and right distances, helping it determine if it is centered. If the robot is closer to one side, it calculates an adjustment based on this difference, incrementally increasing speed on one motor while decreasing it on the other. This differential adjustment is applied symmetrically, ensuring the robot's movement remains smooth and controlled.

The robot also checks for open spaces on the sides. If it detects open space on one side, it adjusts speeds to turn toward that side. If both sides have open space, it turns right, assuming this to be the shorter path. This is achieved by adding a pre-set boost (`openSpaceTurnBoost`) to one motor while subtracting it from the other. If the robot finds both sides and the front blocked within close proximity, it stops both motors to avoid a collision.

All motor speed adjustments are constrained within a PWM range of 0–255 to prevent overdriving. A small delay of 50 milliseconds in the main loop function stabilizes the robot's actions, creating a smooth feedback loop.

1.4. 3D drawings

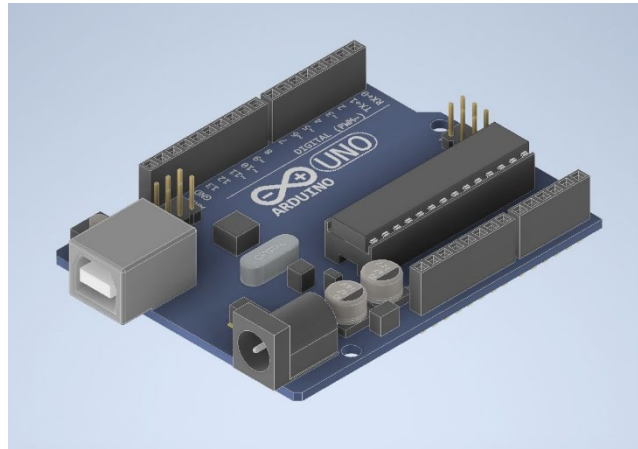


Figure 1.4.1 – Arduino Uno

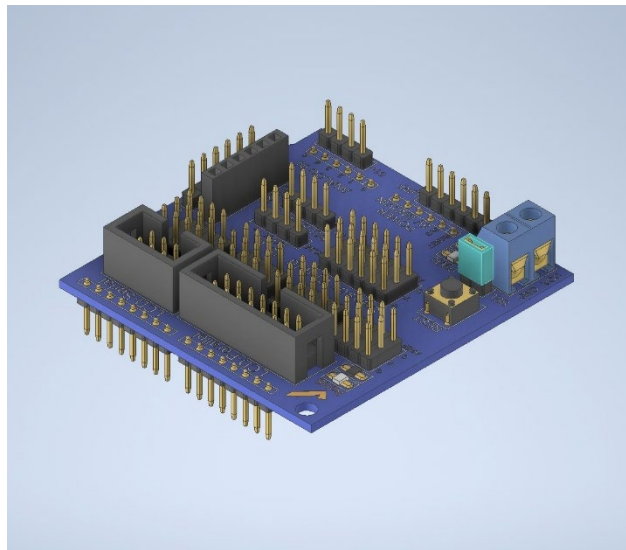


Figure 1.4.2 – Arduino Sensor Shield

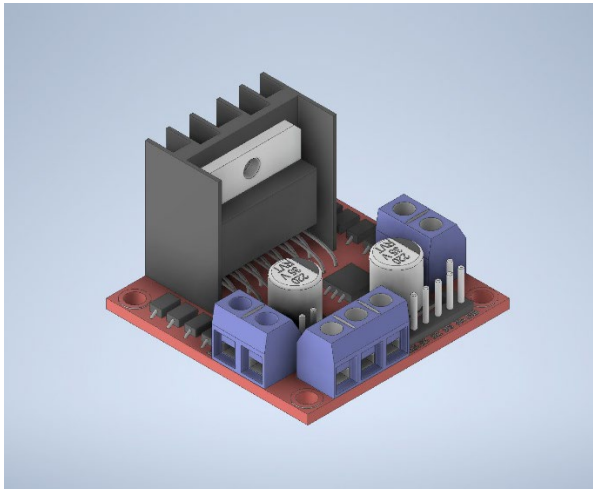


Figure 1.4.3 – L298N Motor Driver

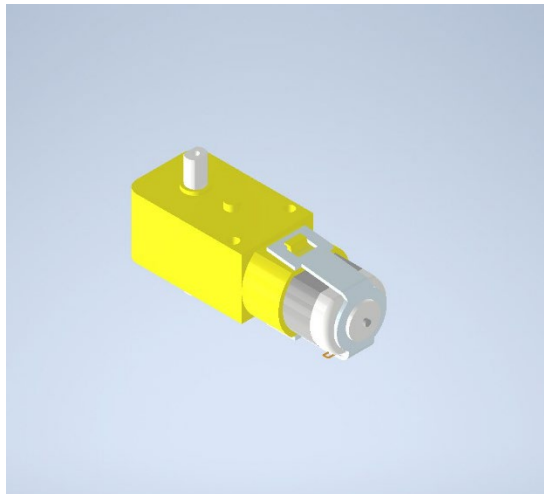


Figure 1.4.4 – DC Motor

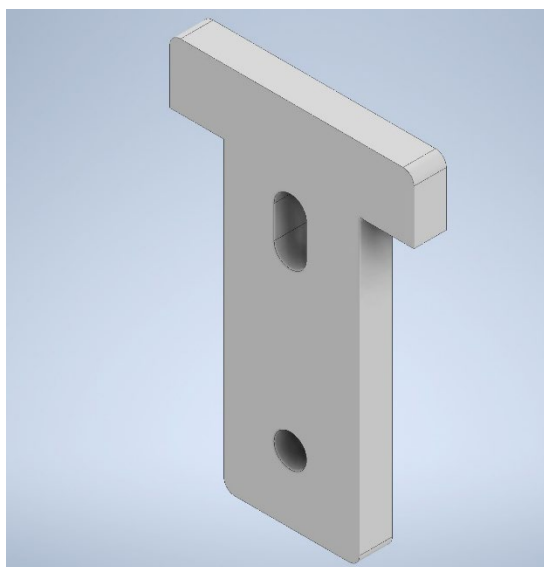


Figure 1.4.5 – DC Motor Bracket



Figure 1.4.6 – Wheels

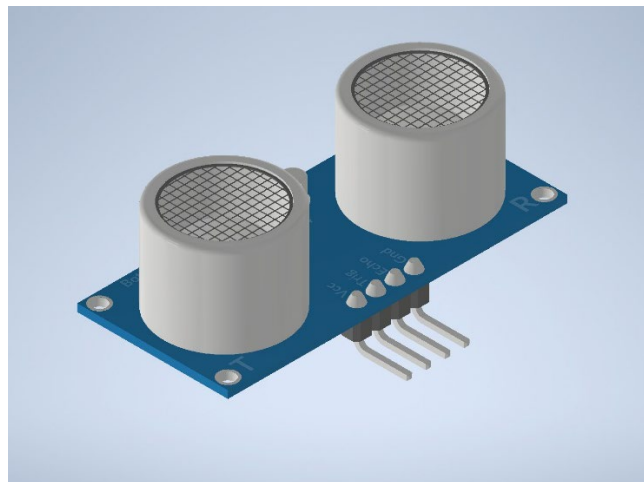


Figure 1.4.7 – Ultrasonic Sensor

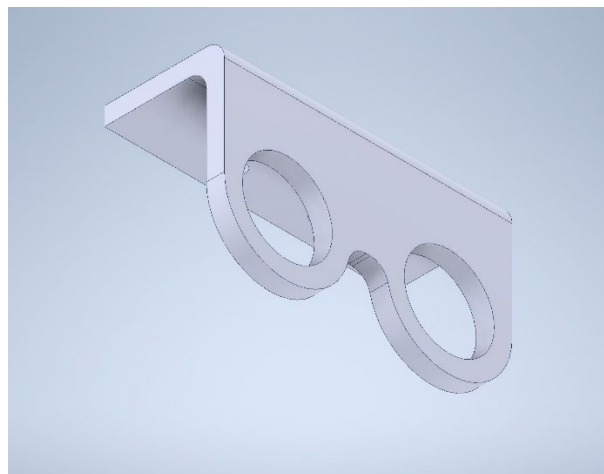


Figure 1.4.8 – Ultrasonic Sensor Mount

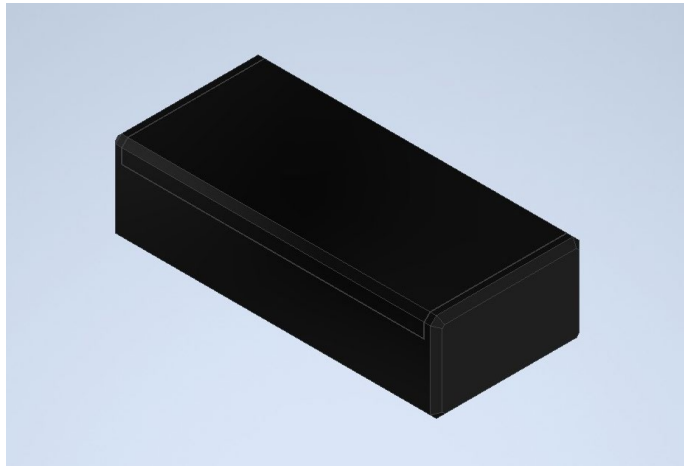


Figure 1.4.9 – Battery Holder

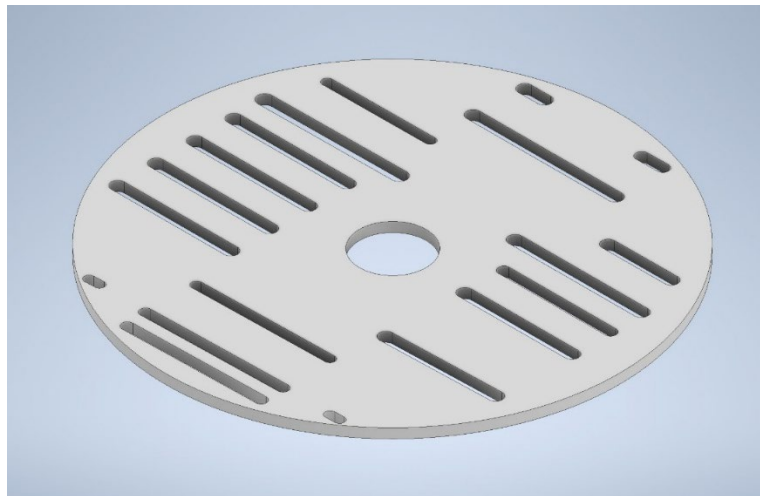


Figure 1.4.10 – Car Body 1

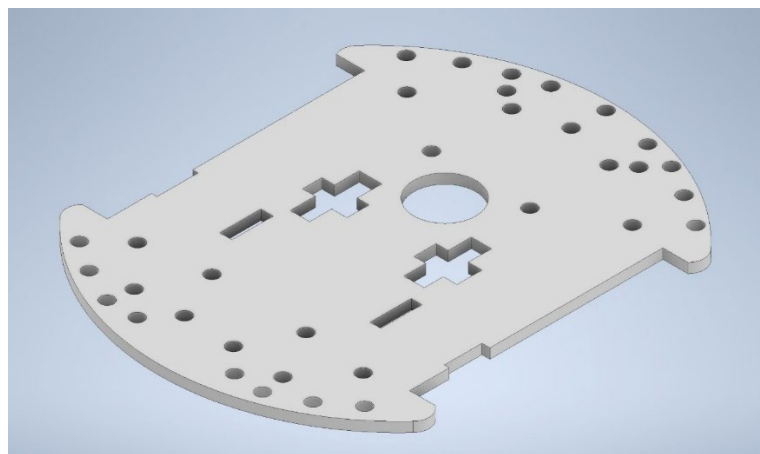


Figure 1.4.11 – Car Body 2

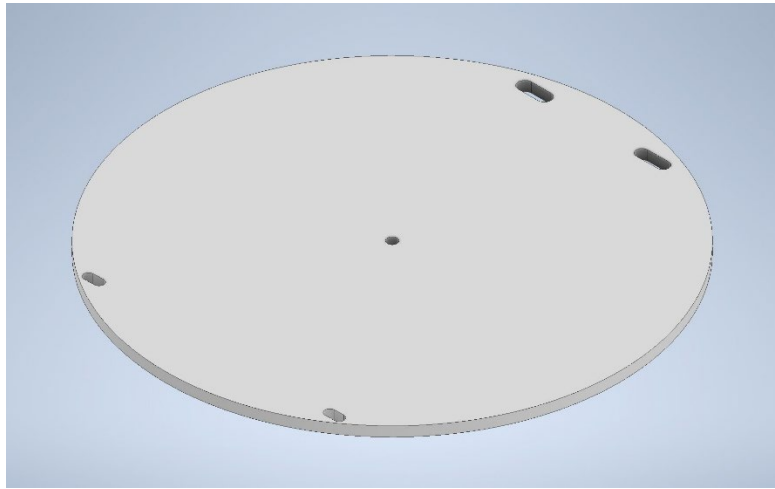


Figure 1.4.12 – Ball Holder Base

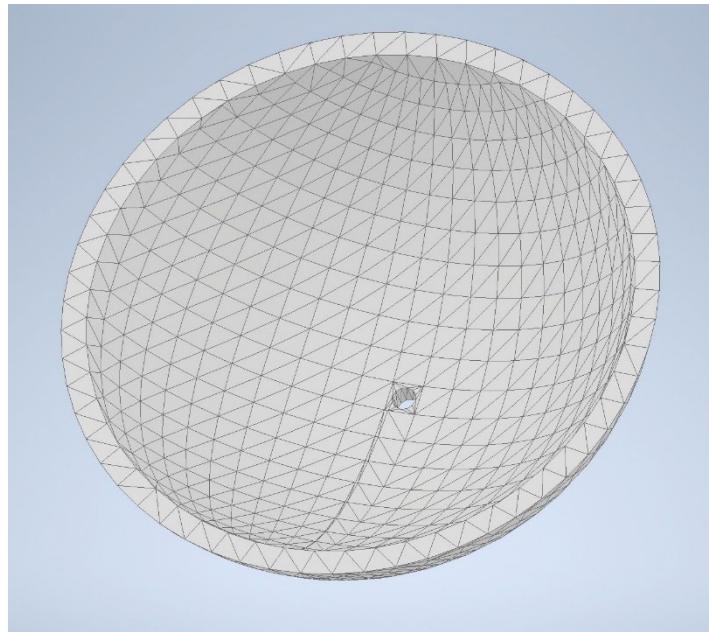


Figure 1.4.13 – Ball Holder

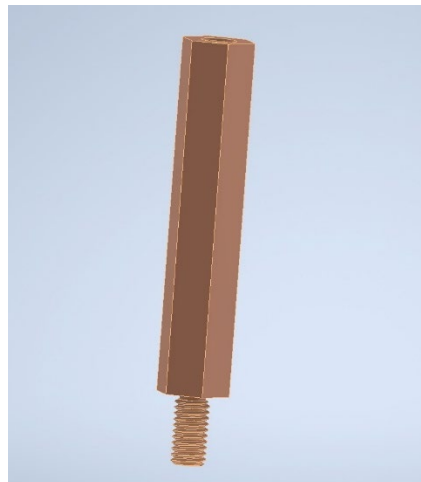


Figure 1.4.14 – Spacer 30 mm

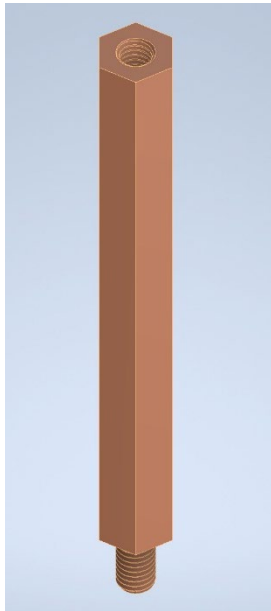
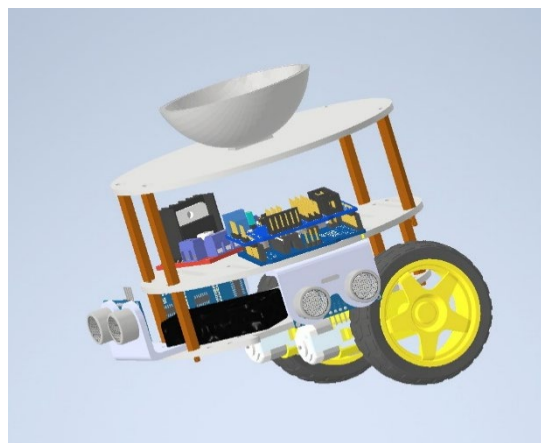
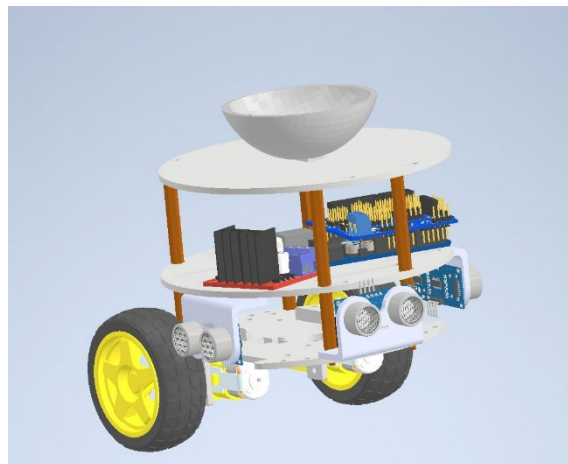


Figure 1.4.15 – Spacer 50 mm

Full Body



2. Manual Robot

2.1. Code

```
import gamepad
import time

from cyberpi import mbot2

MAX_SPEED = 200
MAX_SPEED_SERVO = 180
MAX_SPEED_GRIPPER = 180

speed = 90
speed2 = 0

def calculate_motor_speeds(y_axis, x_axis):
    left_speed = y_axis + x_axis
    right_speed = y_axis - x_axis

    left_speed = max(min(left_speed, 1), -1) * MAX_SPEED
    right_speed = max(min(right_speed, 1), -1) * MAX_SPEED

    return left_speed, right_speed

while True:
    y_axis = gamepad.get_joystick('Lx') / 100
    x_axis = gamepad.get_joystick('Ly') / 100
    left_speed, right_speed = calculate_motor_speeds(y_axis, x_axis)
    mbot2.drive_speed(left_speed, right_speed)

    if gamepad.is_key_pressed('L1'):
        mbot2.motor_set(100, 'M1')

    if gamepad.is_key_pressed('R1'):
        mbot2.motor_set(0, 'M1')

    if gamepad.is_key_pressed('Up'):
        speed = min(speed + 1, MAX_SPEED_SERVO)
        mbot2.servo_set(speed, 's2')
        time.sleep(0.01)

    if gamepad.is_key_pressed('Down'):
        speed = max(speed - 1, 0)
        mbot2.servo_set(speed, 's2')
        time.sleep(0.01)

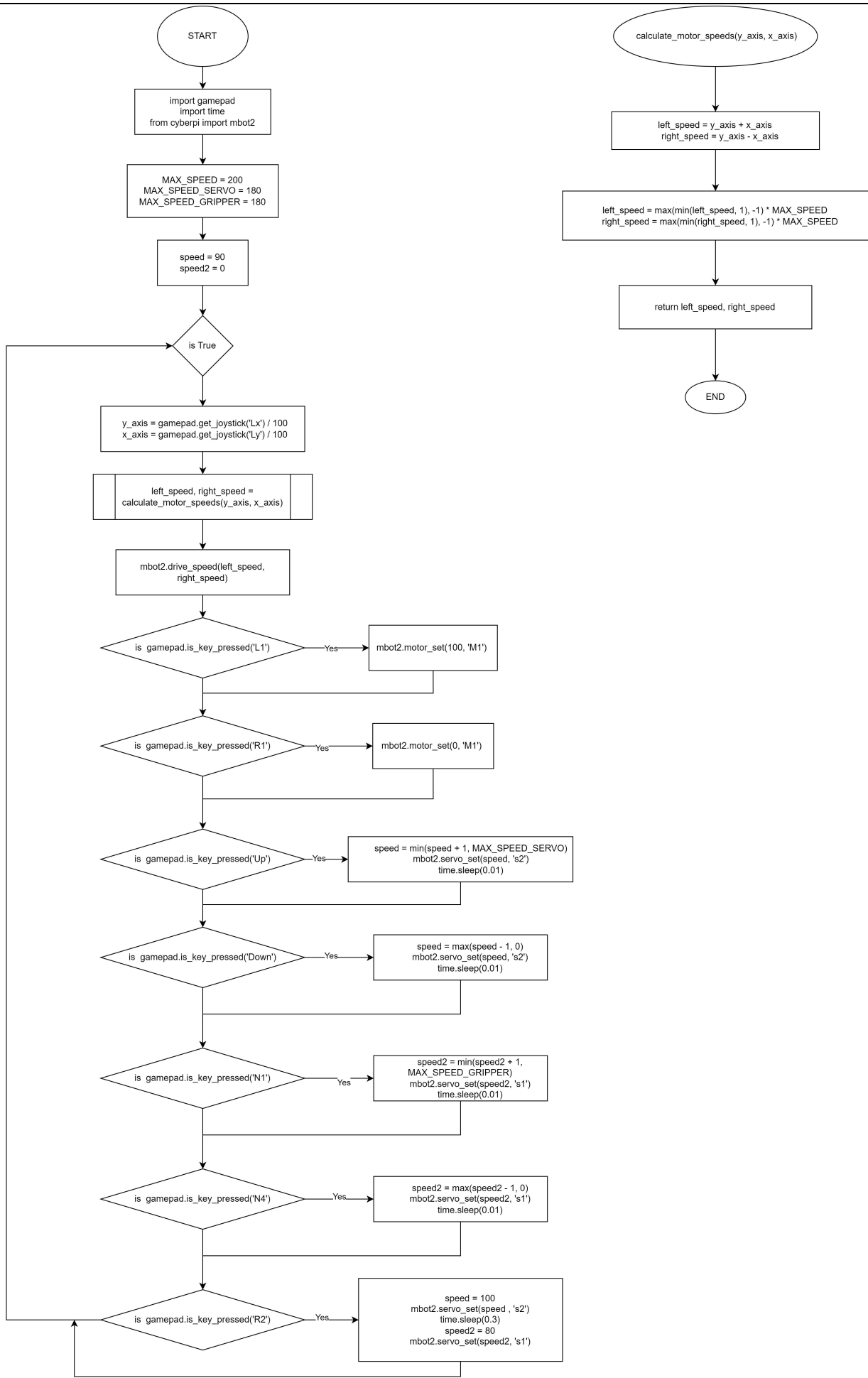
    if gamepad.is_key_pressed('N1'):
        speed2 = min(speed2 + 1, MAX_SPEED_GRIPPER)
```

```
    mbot2.servo_set(speed2, 's1')
    time.sleep(0.01)

if gamepad.is_key_pressed('N4'):
    speed2 = max(speed2 - 1, 0)
    mbot2.servo_set(speed2, 's1')
    time.sleep(0.01)

if gamepad.is_key_pressed('R2'):
    speed = 100
    mbot2.servo_set(speed, 's2')
    time.sleep(0.3)
    speed2 = 80
    mbot2.servo_set(speed2, 's1')
```

2.2. Flowchart



2.3. Code explanation

The left joystick on the gamepad of the manual robot controls the robot's wheels, which use encoder motors. A function, `calculate_motor_speeds(y_axis, x_axis)`, is responsible for converting joystick input values into motor speeds that can be applied directly to the wheels. It does this by summing the joystick's y-axis and x-axis values to determine `left_speed` and `right_speed` for the motors. These speeds are then scaled to stay within `MAX_SPEED`, ensuring the speed does not exceed the motor limits.

The shooter mechanism is operated by pressing `L1` to spin up a DC motor attached to the shooter to full speed, and `R1` to stop it. The grabber arm, which can move up and down, is controlled by the `Up` and `Down` buttons. When `Up` is pressed, the servo motor connected to the grabber (`s2`) increases its angle gradually, allowing the grabber arm to rise smoothly until it reaches `MAX_SPEED_SERVO`. Pressing `Down` decreases the angle in small increments, smoothly lowering the arm.

The robot's claws, part of the grabber mechanism, are operated using the `N1` and `N4` buttons. `N1` gradually increases the angle of a servo motor (`s1`), allowing the claws to open incrementally, while `N4` decreases the angle to close the claws. For a quick-release shooting mechanism similar to a catapult, the `R2` button sets the grabber to a 100-degree angle, positioning it for release. The claws open as soon as the grabber reaches this angle, letting the robot shoot the ball. A small delay is added after each incremental adjustment for the servos to ensure stable, precise movements.

2.4. 3D Model

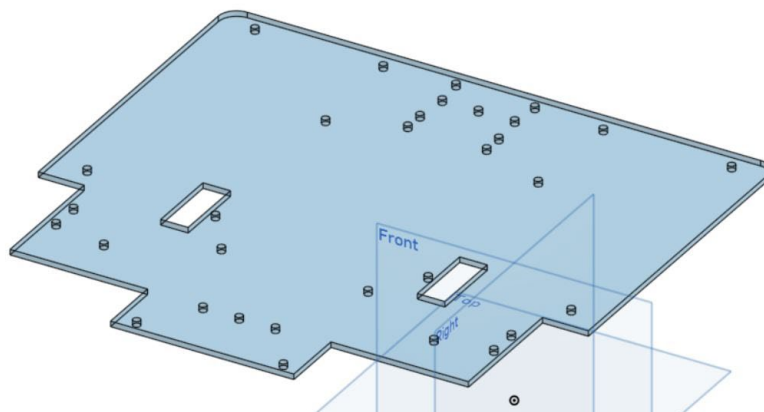


Figure 2.4.1. Main-Robot Base

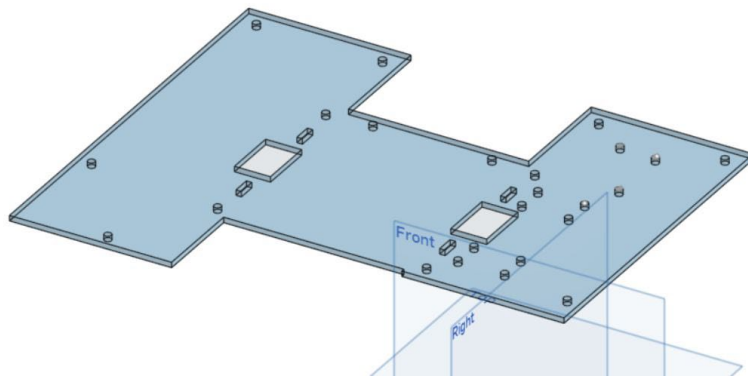


Figure 2.4.2. Second-Robot Base

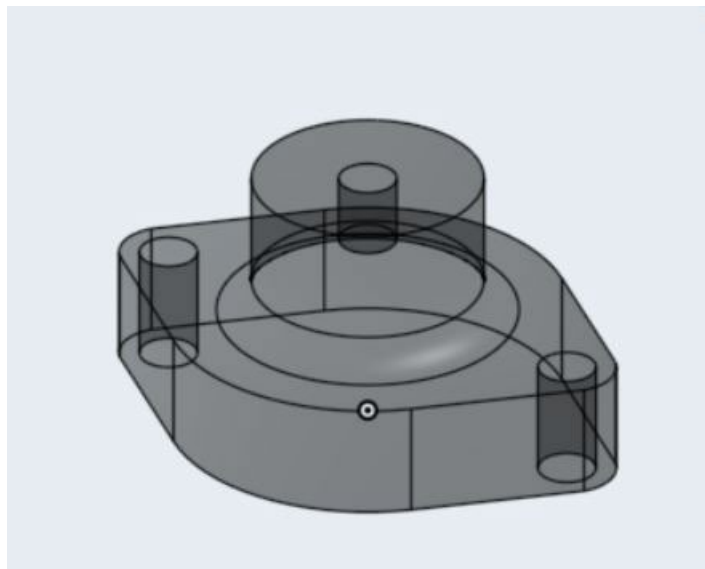


Figure 2.4.3. Ball-Wheel Support Base

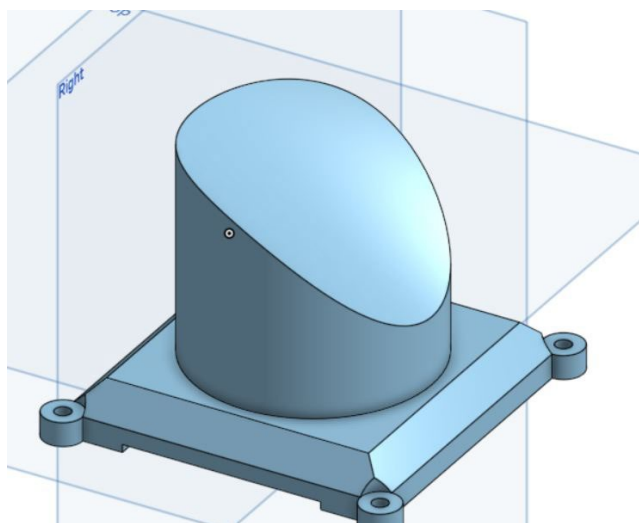


Figure 2.4.4. Shooter Base

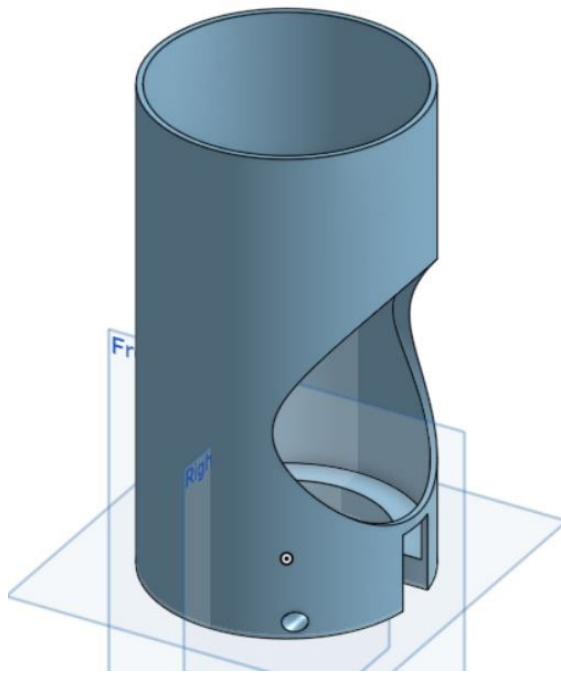


Figure 2.4.5. Gun Barrel

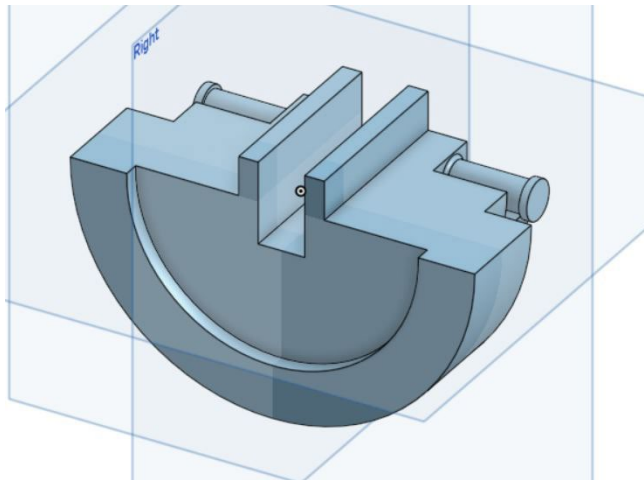
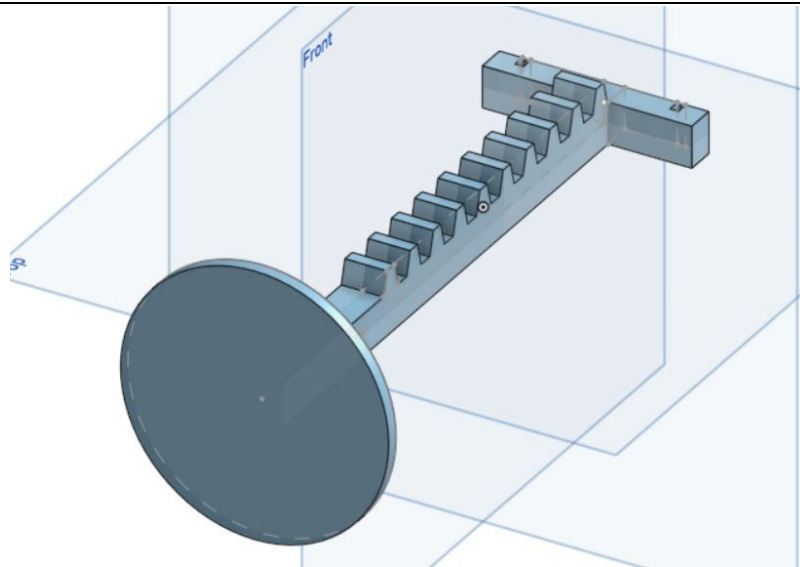


Figure 2.4.6. Shooter Support for Gear Track



2.4.7. Gear Track with Shooting Base

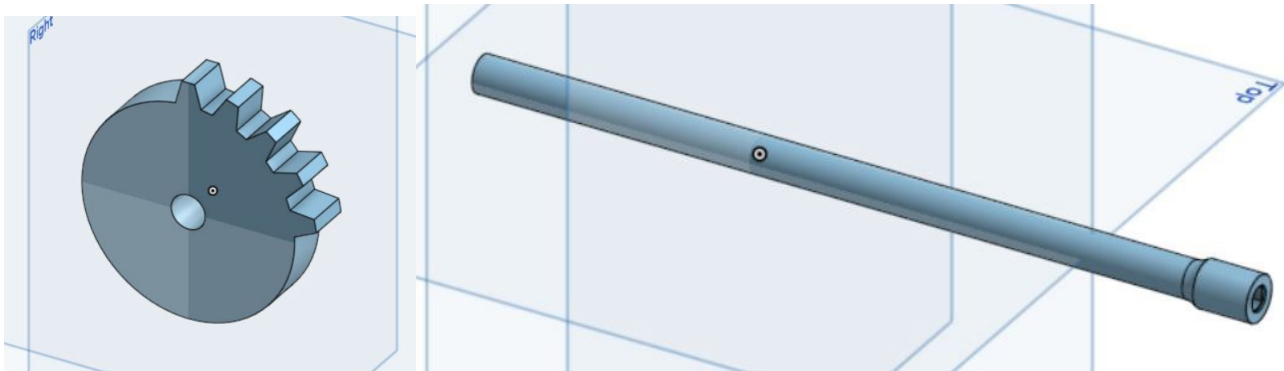


Figure 2.4.8. Gear and Gear Axle for Shooter

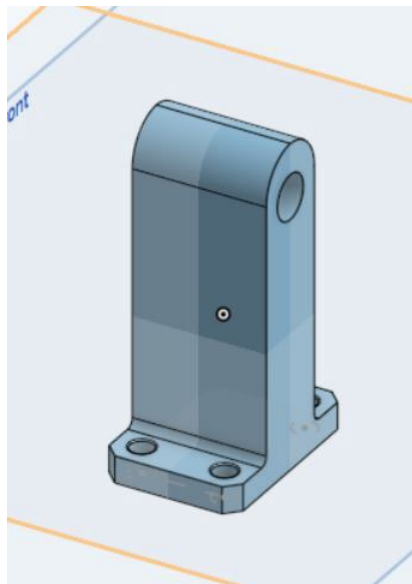


Figure 2.4.9. Axle Support

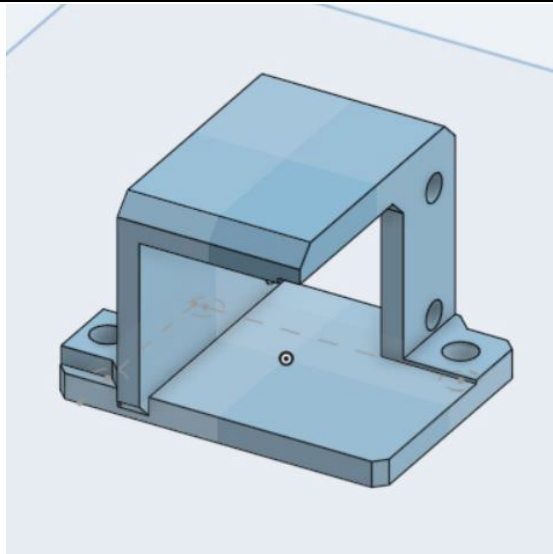


Figure 2.4.10. DC-Motor Base

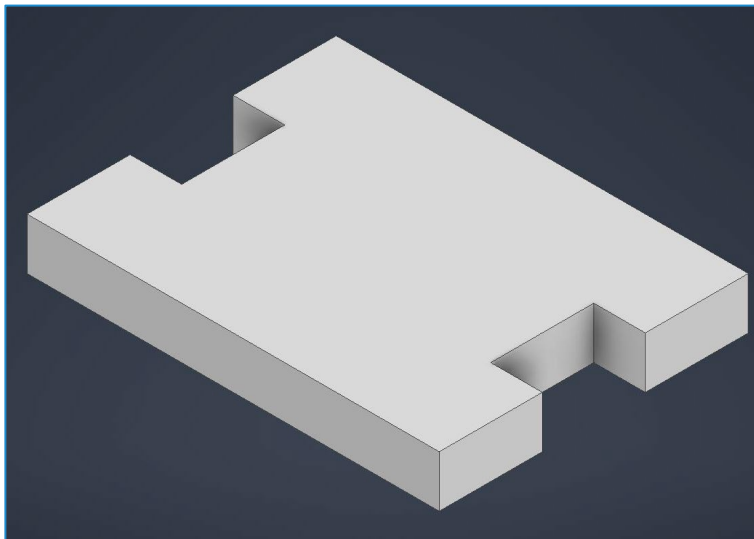


Figure 2.4.11. Small claw spacer

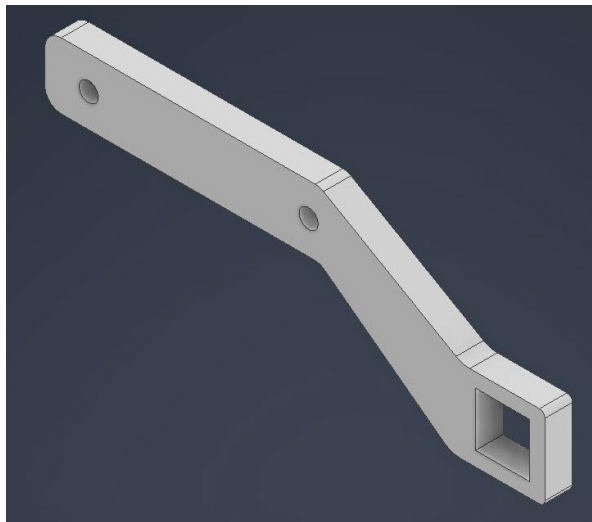


Figure 2.5.12. Left connecting arm



Figure 2.4.13. Right connecting arm

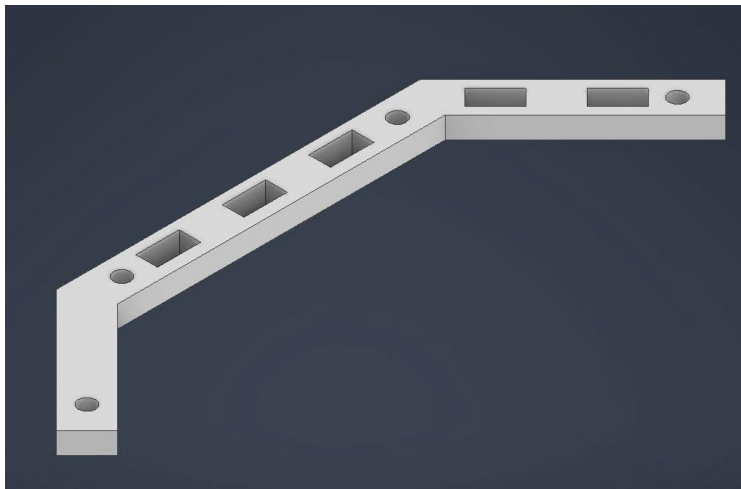


Figure 2.4.14. Claw

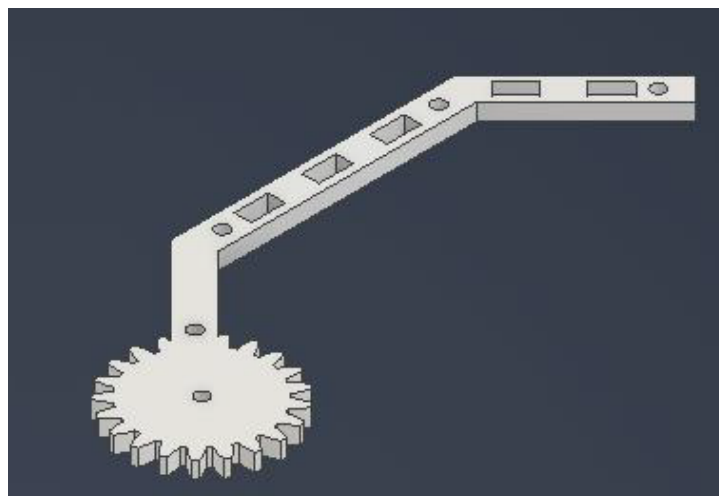


Figure 2.4.15. Left claw

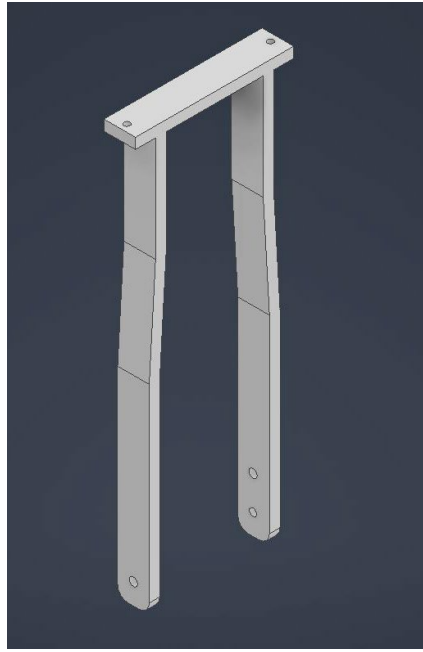


Figure 2.4.16. Support beam

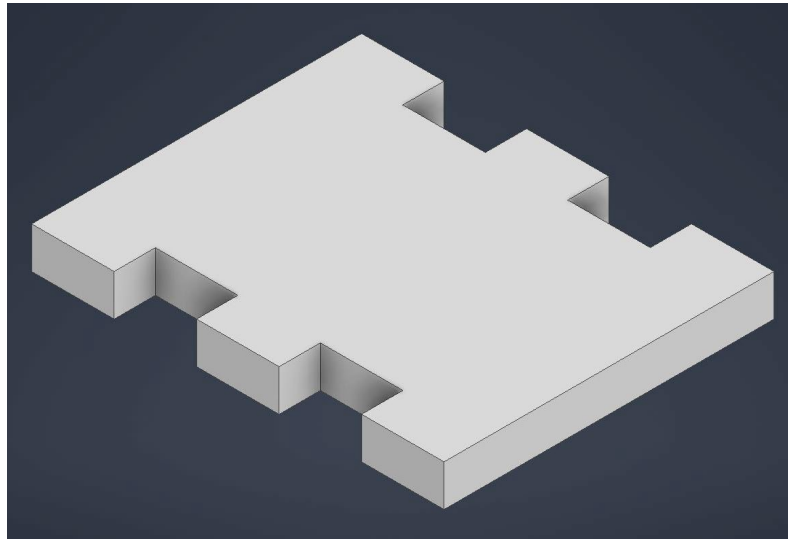


Figure 2.4.17. Large claw spacer

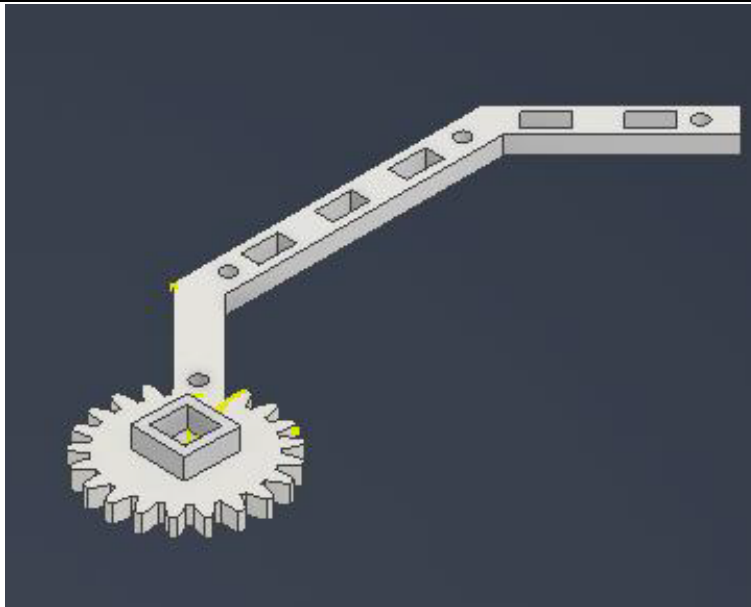


Figure 2.4.18. Right claw

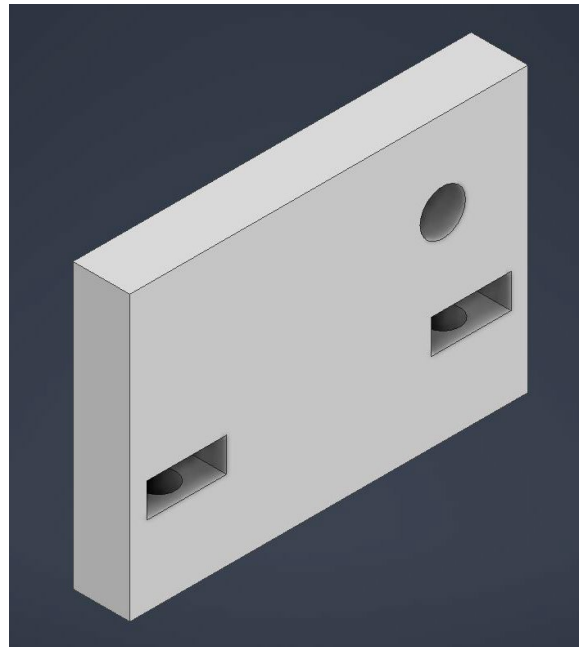


Figure 2.4.19. Right Wall

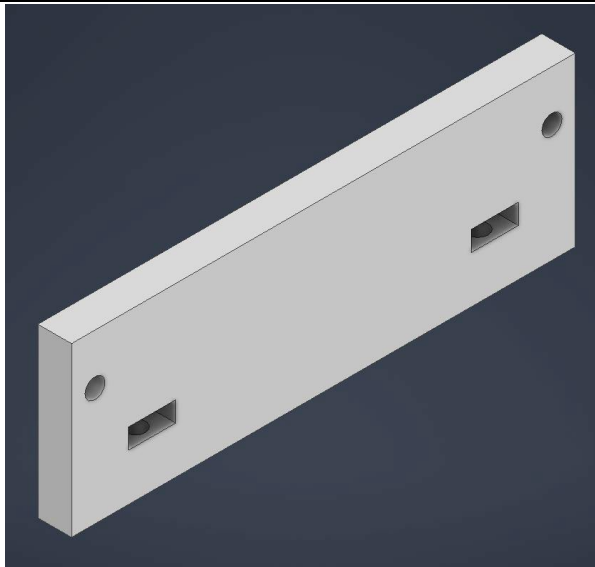


Figure 2.4.20. Back

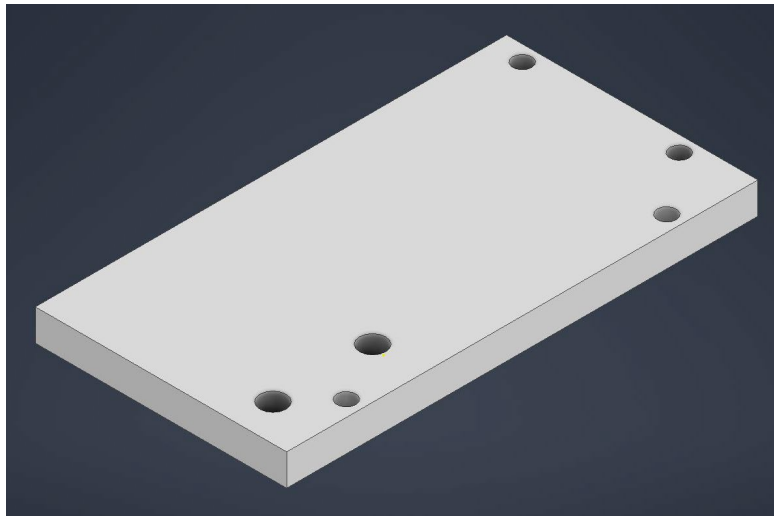


Figure 2.4.21. Claw base

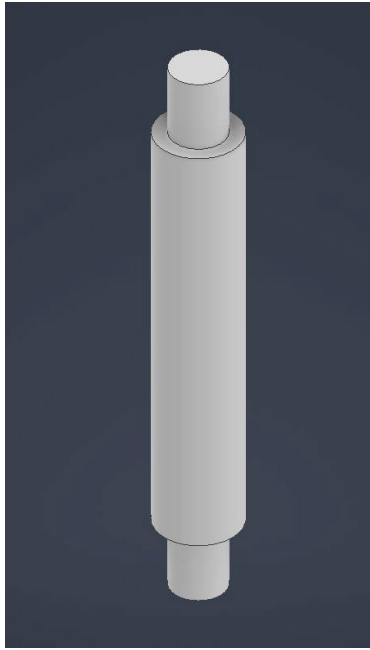


Figure 2.4.21. Pin spacer

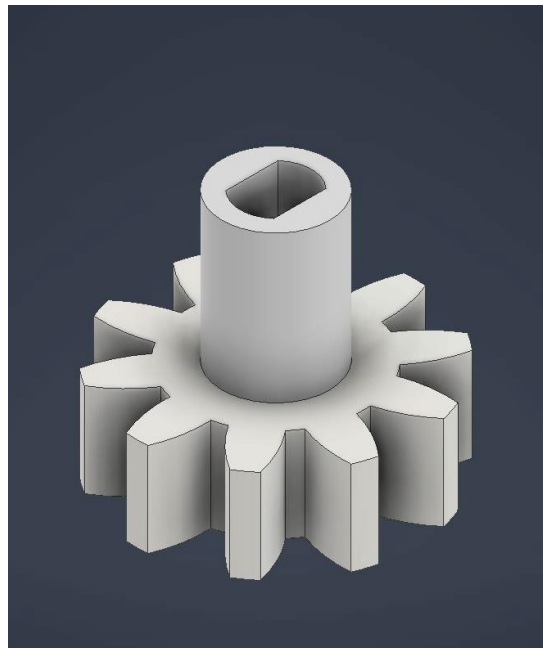


Figure 2.4.22. Dc gear box



Figure 2.4.23. Larger gear

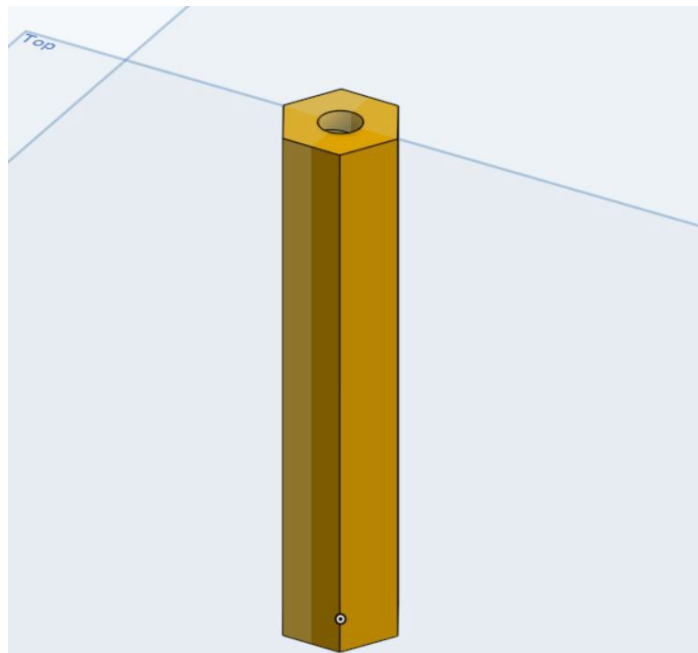


Figure 2.4.24. Spacer

Full Body

