TABLE OF CONTENTS

1. Problem Statement "Autonomous Robotics Systems"

Design an autonomous robot system using the PRIZM library for Arduino that incorporates line-following capabilities and obstacle avoidance. The robot is equipped with line sensors and an ultrasonic sensor to navigate its environment.

Requirements:

1. Line Following:

- **Implement a line-following behavior using the line sensor data.**
- **The robot should adjust its motor powers based on the readings from line sensor.**

2. Obstacle Avoidance:

- **Integrate obstacle avoidance functionality using the ultrasonic sensor.**
- **If the distance measured by the ultrasonic sensor at port 4 is greater than 25 cm, the robot should move forward with reduced speed to avoid collisions. Otherwise, it should move forward at a higher speed.**

3. Motor Control:

- **Configure the motor directions appropriately for effective movement.**
- **Motor 1 direction should be inverted to achieve desired robot behavior.**

4. LED Indication:

- **Use the red LED to provide a visual indication based on the line sensor reading. The LED should be turned off when the line is detected and turned on otherwise.**

5. Consistent Operation:

- **Ensure that the robot operates consistently and smoothly in both line-following and obstacle avoidance modes.**
- **Fine-tune motor powers and sensor thresholds for optimal performance.**

Constraints:

- **The code should be executed in a loop with a delay of 50 milliseconds at the end of each iteration.**
- **Utilize the PRIZM library functions for sensor readings, motor control, and LED manipulation.**

Evaluation:

- **The robot's ability to follow a line and navigate around obstacles effectively.**
- **Smooth transitions between line-following and obstacle avoidance modes.**
- **Proper implementation of motor control and LED indications.**

Note: Ensure the correct wiring and hardware connections are made for the robot to interact with the sensors and actuators as specified in the PRIZM library documentation.

2. Methodology / Procedure/ Algorithm

Algorithm for the Code:

1. Start the program.

2. Initialize the PRIZM controller and set the motor inversion for DC Motor 1.

3. Enter the main loop:

3.1. Read the line sensor value (sensor 3):

- **If the line is detected (value == 0), do the following:**
- **Set DC Motor powers to 125 (Motor 1) and 30 (Motor 2).**
- **Turn off the Red LED (set it to LOW).**

3.2. If no line is detected (value == 1), do the following:

- **Set DC Motor powers to 30 (Motor 1) and 125 (Motor 2).**
- **Turn on the Red LED (set it to HIGH).**

3.3. Read the distance from the ultrasonic sensor (sensor 4) in centimeters.

3.4. Check the distance reading:

- **If the distance is greater than 25 cm, do the following:**
- **Set DC Motor powers to 50 (both motors).**

- **If the distance is less than or equal to 25 cm, do the following:**
  Set DC Motor powers to 125 (both motors).

3.5. Check the line sensor again:

- **If the line is detected (value == 1), do the following:**
- **Turn off the Red LED (set it to LOW).**

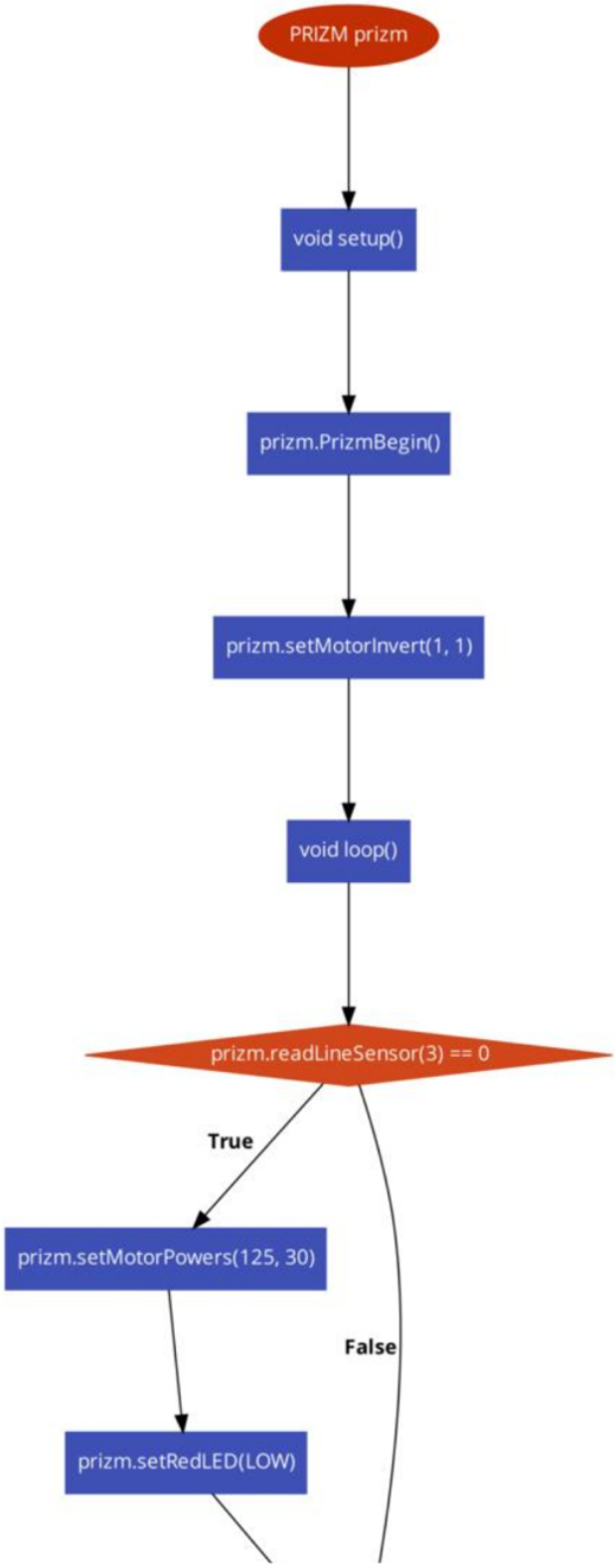
- **If no line is detected (value == 0), do the following:**
- **Turn on the Red LED (set it to HIGH).**

3.6. Add a delay of 50 milliseconds to prevent rapid toggling of the LED.

4. Repeat the main loop (step 3) until the program is stopped or interrupted.

5. End the program.

3. Flowchart:


PRIZM prizm
void setup()
prizm.PrizmBegin()
prizm.setMotorInvert(1, 1)
void loop()
prizm.readLineSensor(3) == 0
True
False
prizm.setMotorPowers(125, 30)
prizm.setRedLED(LOW)

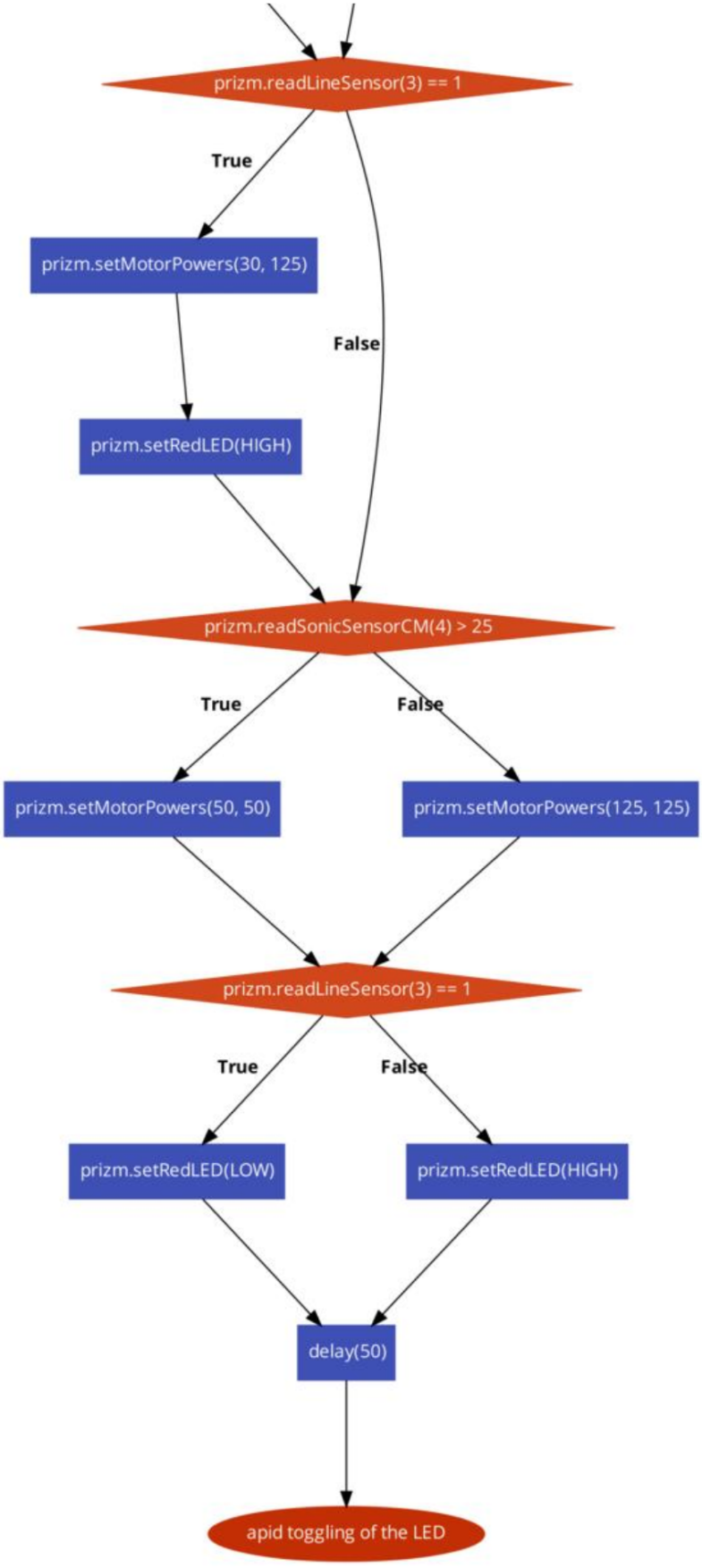


Fig:1 flowchart

4. Coding (C/Python)

```
#include <PRIZM.h>
PRIZM prizm;

void setup() {
  prizm.PrizmBegin();
  prizm.setMotorInvert(1, 1);

}

void loop () {
{

  if (prizm.readLineSensor(3) == 0) {
    prizm.setMotorPowers(125, 30);

  }

  if (prizm.readLineSensor(3) == 1) {
    prizm.setMotorPowers(30, 125);

  }
}

  if (prizm.readSonicSensorCM(4) > 25) {
    prizm.setMotorPowers(50, 50);
  } else {
    prizm.setMotorPowers(125, 125);
  }

  if (prizm.readLineSensor(3) == 1) {
    prizm.setRedLED(LOW);
  } else {
    prizm.setRedLED(HIGH);
  }

  delay(50);
}
```

5. Modules of the proposed work

It appears that you are using the PRIZM library for programming a robot or similar device. The provided code snippet includes the setup and loop functions, where you initialize and control the PRIZM module. Here's a breakdown of the modules in your code:

Modules:

1. PRIZM Initialization

- Library Inclusion: `#include <PRIZM.h>`
- PRIZM Object: `PRIZM prizm; `
- Initialization: `prizm. PrizmBegin (); `
- Motor Inversion: `prizm.setMotorInvert(1, 1); `

2. Setup Function

- Function: 'void setup () {...}`
- Initialization: Initializes the PRIZM module and sets motor inversion.

3. Loop Function

- Function: 'void loop () {...}`
- Line Sensor Control:
  - Reads line sensor 3 and adjusts motor powers accordingly.
- Sonic Sensor Control:
  - Reads sonic sensor 4 and adjusts motor powers based on the distance.
- LED Control:
  - Controls the red LED based on the state of line sensor 3.
- Delay: `delay (50); `

Proposed Work:

The existing code appears to control a robot or similar device based on input from line sensors and a sonic sensor. The proposed work might involve:

1. Enhancing Sensor Logic:
   - Fine-tune the sensor readings and motor power adjustments for smoother control.

2. Adding Comments:
   - Include comments in the code to explain the purpose of each section or line.

3. Modifying Motor Powers:
   - Adjust motor power values based on the specific requirements and characteristics of your robot.

4. Adding More Functionality:
   - Extend the code to include additional sensors, behaviors, or features based on your project needs.

5. Error Handling:

- Implement error handling or contingency plans for unexpected sensor readings or issues.

6. Testing:
   - Test the robot in different environments to ensure reliable performance.

7. Documentation:
   - Document the code and any modifications made for future reference.

Remember to adapt the code according to your specific hardware, robot configuration, and project requirements. Additionally, testing and iterating on the code are crucial for achieving the desired behavior.

6. Results/Screenshots

pps.ino

```
#include <PRIZM.h> // include PRIZM library
PRIZM prizm; // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
  prizm.PrizmBegin(); // initialize PRIZM
  prizm.setMotorInvert(1, 1); // invert the direction of DC Motor 1
  // to harmonize the direction of
  // opposite-facing drive motors
}

void loop() {
  // Beam reflected, no line detected
  if (prizm.readLineSensor(3) == 0) {
    prizm.setMotorPowers(125, 30);
    prizm.setRedLED(LOW);
  }
  // No reflected beam, line detected
  if (prizm.readLineSensor(3) == 1) {
    prizm.setMotorPowers(30, 125);
    prizm.setRedLED(HIGH);
  }

  if (prizm.readSonicSensorCM(4) > 25) {
    prizm.setMotorPowers(50, 50); // if distance greater than 25 cm, do this
  } else {
    prizm.setMotorPowers(125, 125); // if distance less than 25 cm, do this
  }

  // Check and invert the Red LED based on the sensor reading
  if (prizm.readLineSensor(3) == 1) {
    prizm.setRedLED(LOW);
  } else {
    prizm.setRedLED(HIGH);
```

Output

```
Sketch uses 4060 bytes (12%) of program storage space. Maximum is 32256 bytes.
Global variables use 225 bytes (10%) of dynamic memory, leaving 1823 bytes for local variables. Maximum is 2048 bytes.
```

Fig:2 input and output

output:

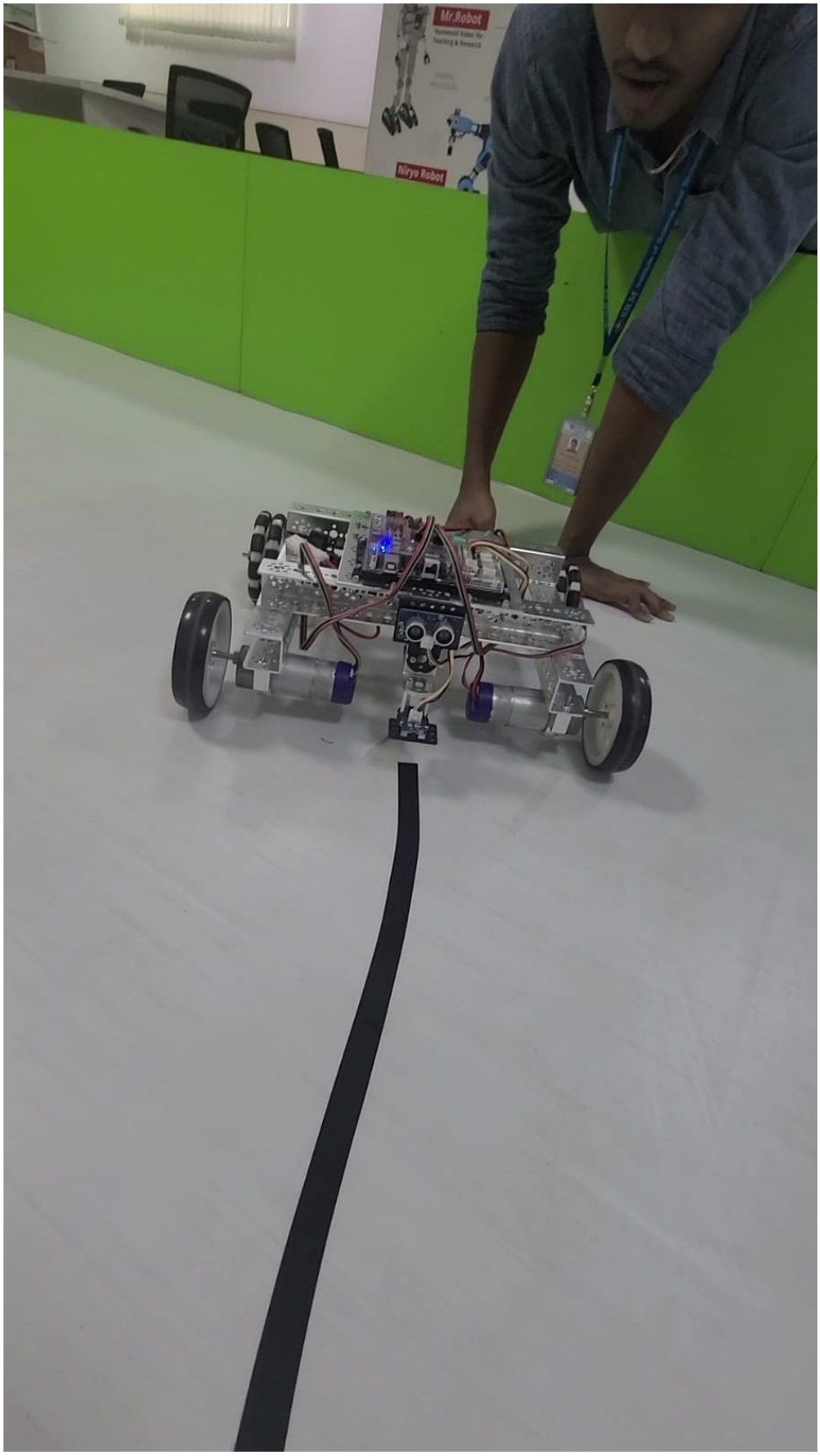


Fig:3 Autonomous Robot

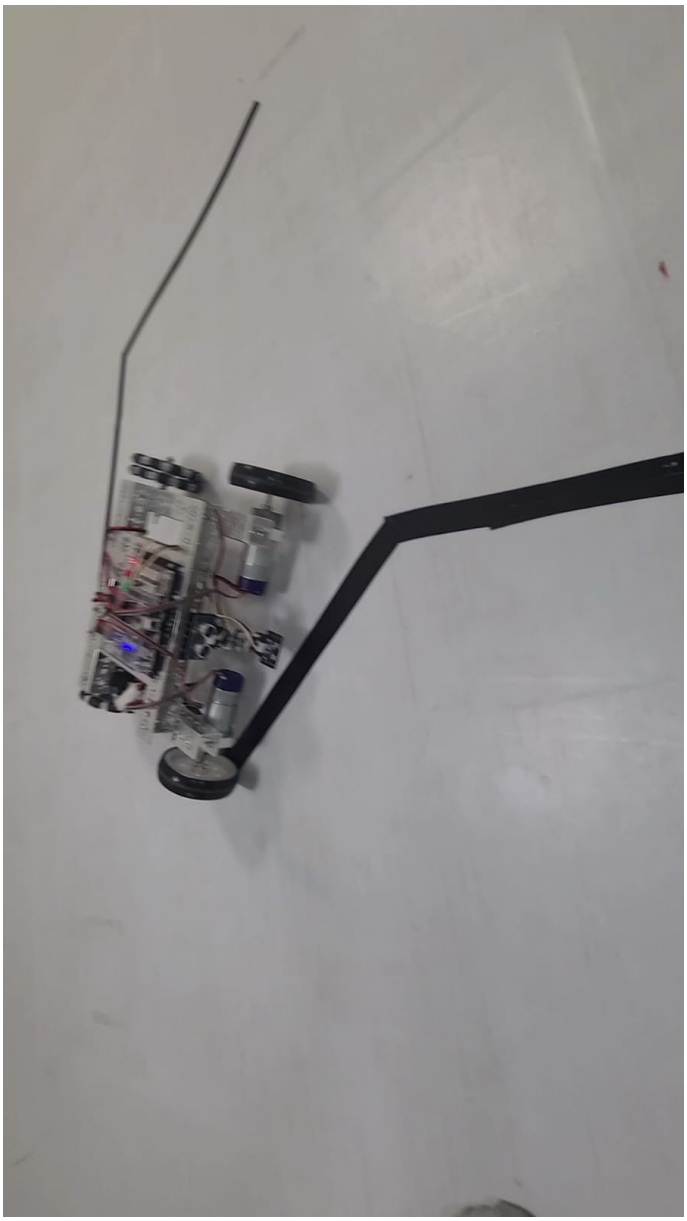


Fig:4 obstacle Avoidance

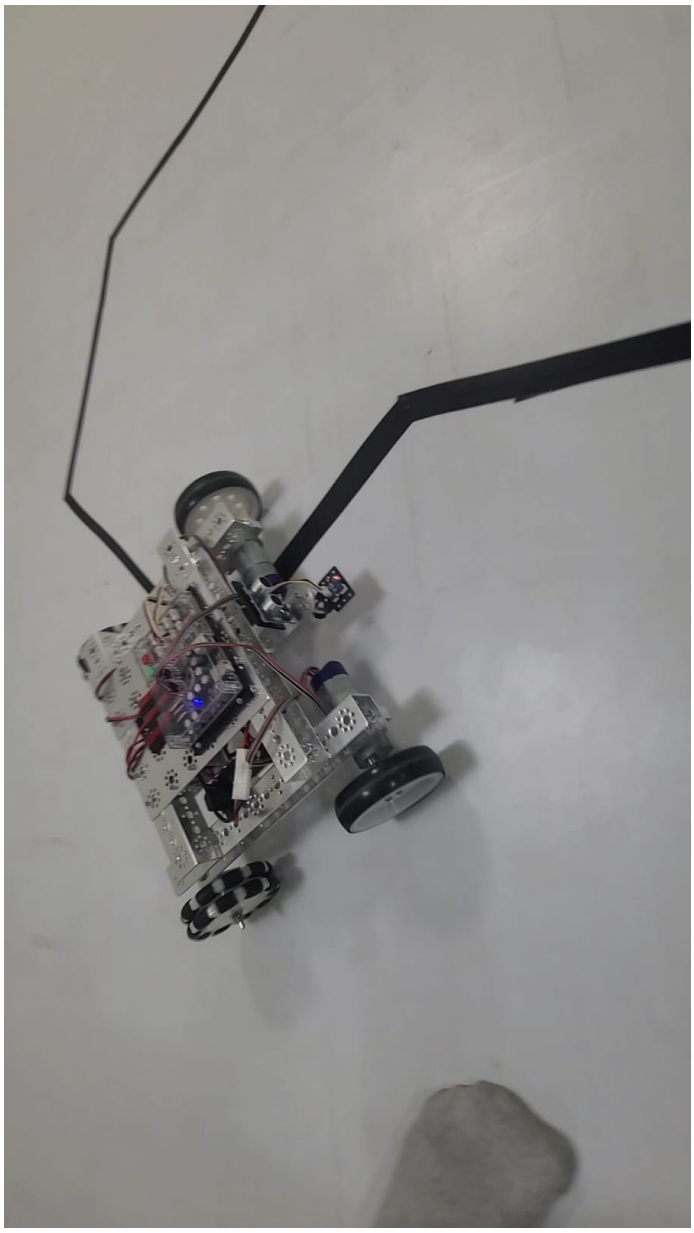


Fig:5 Line Following

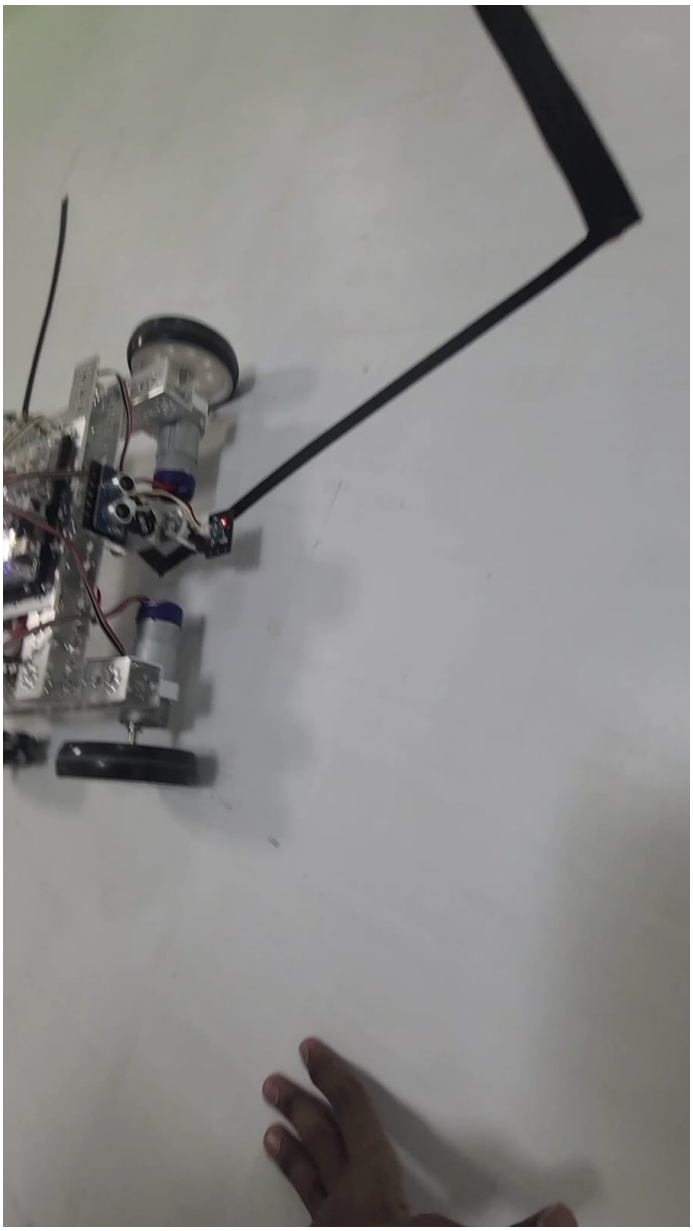


Fig:6 LED Indication

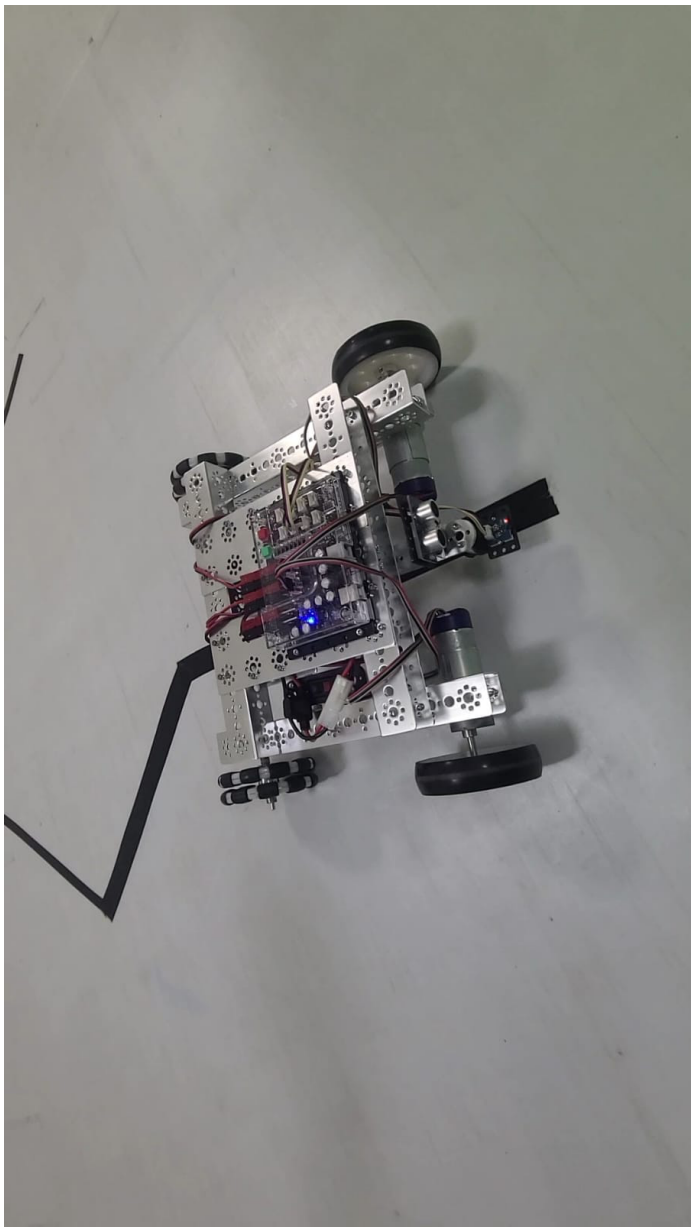


Fig:7 Consistent Operation

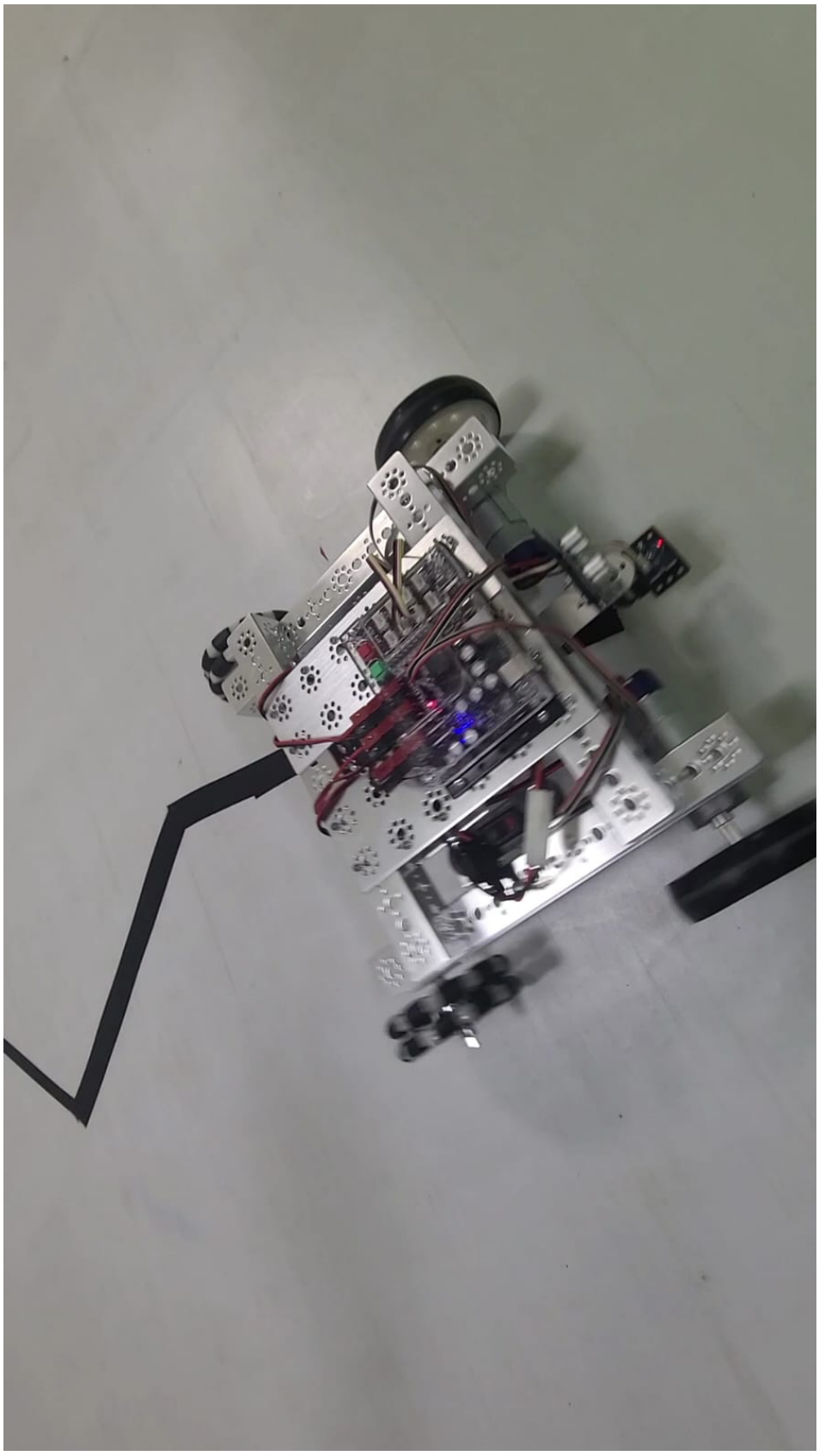


Fig:8 robot's ability to follow a line and navigate.

7. Conclusion

- **The code implements a simple behavior for a robot to follow a line using line sensors, avoid obstacles using a sonic sensor, and control an LED based on sensor feedback.**
- **The logic suggests a basic navigation algorithm where the robot adjusts its motor powers and LED status in response to the detected environment.**
- **Further enhancements and functionalities could be added based on the specific requirements of the robot's task, such as additional sensor inputs or more complex decision-making algorithms.**

8. References

1. **TETRIX® PRIZM® Robotics(https://www.pitsco.com/)**
2. **https://app.code2flow.com/ (for flowchart)**
3. **https://docs.arduino.cc/ (**Arduino **library)**
4. **https://chat.openai.com/**
5. **https://www.grammarly.com/**
6. **https://github.com/**
7. **https://www.ibm.com/topics/rpa**
8. **https://code.visualstudio.com/**