RMIT UNIVERSITY

School of Science, Engineering & Technology

Final Project

COSC3011 Programming Autonomous Robots

Lecturer:

Dr. Ginel DORLEON

Author:

Kim Nhat Anh (s3978831)

Dang Cuu Dang Khoa (s3979159)

RMIT University

May 25, 2025

I declare that in submitting all work for this assessment I have read, understood and agree to the content and expectations of the Assessment declaration.

Abstract

This report presents the development and evaluation of an autonomous robot for obstacle avoidance and navigation tasks, as part of Final Project for COSC3070 – Programming Autonomous Robots at RMIT University. The project involved designing a control algorithm using sensor feedback, implementing the logic on embedded hardware, and testing the robot in the environment. The methodology combined reactive behavior with a simple rule-based system for path planning. Results demonstrate successful autonomous operation in controlled scenarios, with room for improvement in real-time adaptability. The findings highlight the importance of sensor calibration and software-hardware integration in robotics.

Contents

1 Introduction

Autonomous robots play a critical role in the advancement of smart systems across industries such as logistics, transportation, and manufacturing. These systems rely on embedded intelligence to perceive their environment, make decisions, and act without human input. A fundamental capability in such systems is the ability to follow predetermined paths while adapting to dynamic environments through obstacle avoidance and signal interpretation.

This project aims to replicate these core functionalities using the mBot2 Neo robot, a programmable platform equipped with sensors. The robot was tasked with navigating a complex maze environment by accurately following either a white or yellow line, detecting and responding to visual stop/go signs (red/green color), and avoiding static and dynamic obstacles. The environment mimics real-world traffic scenarios where autonomous systems must balance goal-directed behavior with real-time responsiveness.

The report documents the approach taken to achieve these tasks, the challenges encountered, and the evaluation of the robot's performance within the given constraints.

2 Methodology

The autonomous navigation behavior of the mBot2 Neo robot in this project is driven by a combination of sensor inputs and control algorithms designed for line following, obstacle avoidance, and color-based decision making.

2.1 mBot2 Neo Robot Overview

The mBot2 Neo is a programmable mobile robot equipped with various sensors and actuators suitable for autonomous navigation tasks. It includes a quad RGB sensor for color detection, an ultrasonic distance sensor for obstacle avoidance, and motor encoders for precise movement control. The robot also features a built-in camera, an LED matrix, and is powered by a CyberPi microcontroller, which supports Python programming and real-time sensor integration.

This platform offers flexibility and simplicity for implementing sensor-based decision-making, path following algorithms, and reactive behaviors required for this project.

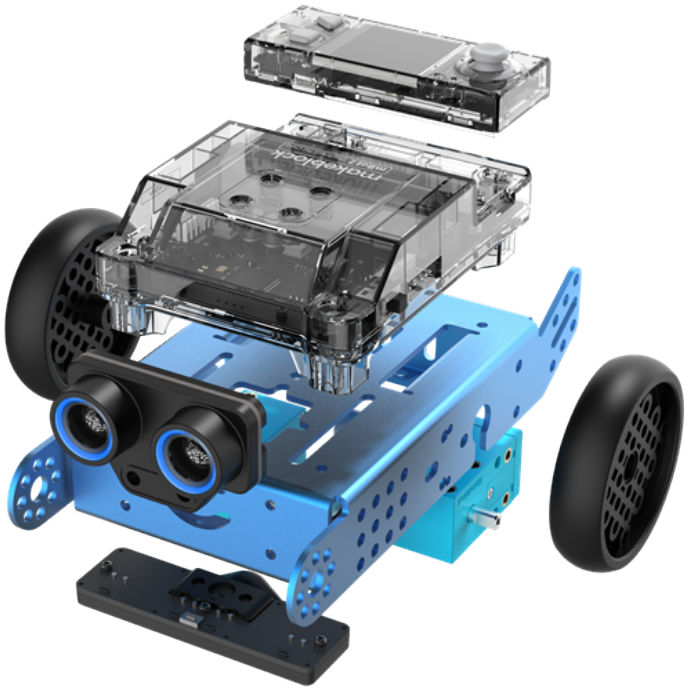

Figure 1: The mBot2 Neo robot with CyberPi [1]

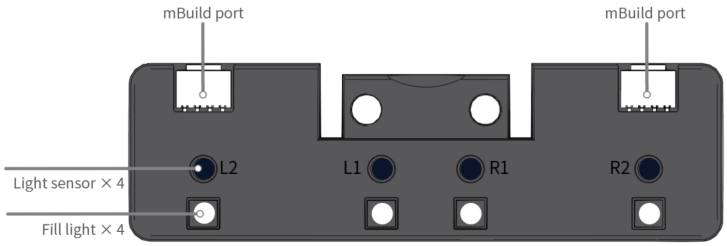

Figure 2: The quad RGB Sensor [2]

Figure 3: The Ultrasonic Sensor [3]

2.2 Line Following using Proportional Control Algorithm (PCA)

To ensure smooth and accurate path following, a Proportional Control Algorithm (PCA) was implemented. This algorithm adjusts the robot's steering proportionally to the error detected

between the center of the robot and the center of the path. The error is calculated based on the intensity readings from the quad RGB sensor mounted on the front of the robot.

When the robot deviates from the path, the PCA applies a corrective steering adjustment proportional to the deviation. This allows for responsive and stable line following, even through curves and slight shifts in path alignment.

2.3 Color Detection using Quad RGB Sensor

A quad RGB sensor is used to detect both the path color and visual indicators (traffic signs) embedded in the maze. The sensor differentiates between multiple colors (white, yellow, red, green), and corresponding behaviors are triggered based on the detected color:

- **White:** Normal speed — this is the default behavior when no special signs are detected.
- **Yellow:** The robot slows down to ensure cautious navigation.
- **Red:** The robot stops immediately and waits for green, if not turn around.
- **Green:** The robot resumes movement and increases speed on second detection.

This color-coded system simulates real-world traffic control mechanisms and ensures the robot can adapt dynamically to the environment.

2.4 Obstacle Detection using Ultrasonic Sensor

An ultrasonic sensor is mounted on the front of the robot to detect obstacles. It continuously measures the distance to objects ahead in real-time. When an object is detected within a predefined safety threshold, the robot halts and initiates an obstacle handling routine.

The avoidance behavior follows a multi-stage decision process:

- The robot first comes to a complete stop and waits for the obstacle to be removed.
- If the obstacle remains after 10 seconds, the robot assumes the path is blocked and performs a U-turn to find an alternative route.

- If the obstacle only partially covers the path and the remaining space is sufficient, the robot attempts to **drift or steer around** the obstacle and continue following the line.

This logic ensures the robot behaves intelligently in dynamic environments, minimizing manual intervention while maintaining a consistent navigation flow. The timing and spatial thresholds used in this process were calibrated through iterative testing to balance responsiveness with stability.

2.5 T-Section

At T-junctions, the robot evaluates the forward path using both color recognition and ultrasonic sensing. If a red signal is detected or an obstacle is present directly ahead, the robot identifies the route as blocked. Instead of stopping indefinitely, it makes a decision to turn either left or right—depending on which side is unobstructed—to continue navigating the course. This ensures continuous movement and demonstrates autonomous path decision-making based on real-time environmental feedback.

3 Implementation

The control system for the mBot2 Neo robot was implemented in Python, utilizing the CyberPi microcontroller's event-driven framework. The software integrates sensor data processing, motor control, and decision-making to achieve autonomous navigation, line following, and obstacle avoidance behaviors.

3.1 Control Architecture

The robot's behavior is triggered by specific user inputs on the CyberPi controller, which initiate event handlers defined using the `event` library. These events include starting line following, stopping the robot, and entering a color debug mode for sensor verification.

3.2 Line Following Algorithm

A Proportional Control Algorithm (PCA) governs the robot's steering during line following. The quad RGB sensor provides real-time feedback on the robot's **offset** [4] from the path, which represents the lateral displacement of the robot relative to the center of the line.

The offset is a signed value:

- A value of `0.0` indicates perfect alignment with the center of the line.
- A positive value indicates deviation to the right.
- A negative value indicates deviation to the left.

This offset is multiplied by a proportional gain constant (`kp`), determined experimentally via trial and error, to calculate the necessary correction for motor speeds. The correction helps realign the robot back to the center of the path. This ensures smooth and continuous steering adjustments to keep the robot aligned.

The function `drive_adjust(offset_val, base_speed)` encapsulates this control logic, computing motor powers dynamically using the formula:

```
correction = kp * offset_val
motor_left = base_speed + correction
motor_right = base_speed - correction
```

This allows the robot to react proportionally to its deviation and maintain accurate path following under varying conditions.

3.3 Color-Based Behavior

The robot uses the quad RGB sensor to detect color signals along the path, triggering specific behaviors:

- **White:** Default line following at standard speed.
- **Yellow:** Temporary speed reduction with a 5-second slowdown period then return to normal speed.

- **Red:** Immediate stop and backward a litte bit, wait for ten seconds to do an U turn resume on green.
- **Green:** Resumption of forward movement after a red stop.

These behaviors simulate traffic light control, enhancing the robot's responsiveness to environmental cues.

3.4 Obstacle Avoidance

An ultrasonic distance sensor continuously measures the proximity of obstacles ahead. If an object is detected within 15 cm, the robot stops and waits up to 10 seconds for clearance. If the obstacle persists beyond this timeout, the robot performs a 180-degree turn to seek an alternative path.

This multi-stage obstacle handling routine balances safety and progress, preventing collisions while minimizing unnecessary stops.

3.5 Event-Driven Control Flow

The program architecture leverages asynchronous event handling to manage user inputs and robot states. The `@event.is_press` decorators define callbacks for buttons A (stop), B (start navigation), and the middle joystick (color debug mode). This design allows smooth switching between operational modes without blocking sensor processing or motor control.

3.6 Additional Features (Creative Features)

The color debug mode continuously displays sensor color readings on the console for real-time verification, aiding in calibration and troubleshooting. The LED matrix and audio tones provide multimodal feedback to improve user awareness of the robot's state and environment.

- Red: stops and waits for green (blinking red LEDs and alert tone)
- Green: resumes movement (blinking green LEDs and confirmation tone)
- Yellow: slows down temporarily (yellow LEDs and warning tone)

- Obstacle detected: halts with periodic tones for 10 seconds

3.7 Parameter Tuning

The proportional gain constant `kp` was fine-tuned by trial and error to achieve stable and responsive line tracking. Initial tests used `kp = 0`, progressively increased to `0.22` for optimal performance, highlighting the importance of empirical tuning in embedded robotics control systems.

Overall, the implementation integrates sensor fusion, control algorithms, and user interaction in a robust, modular design suitable for autonomous navigation challenges faced by the mBot2 Neo robot.

3.8 Flowchart

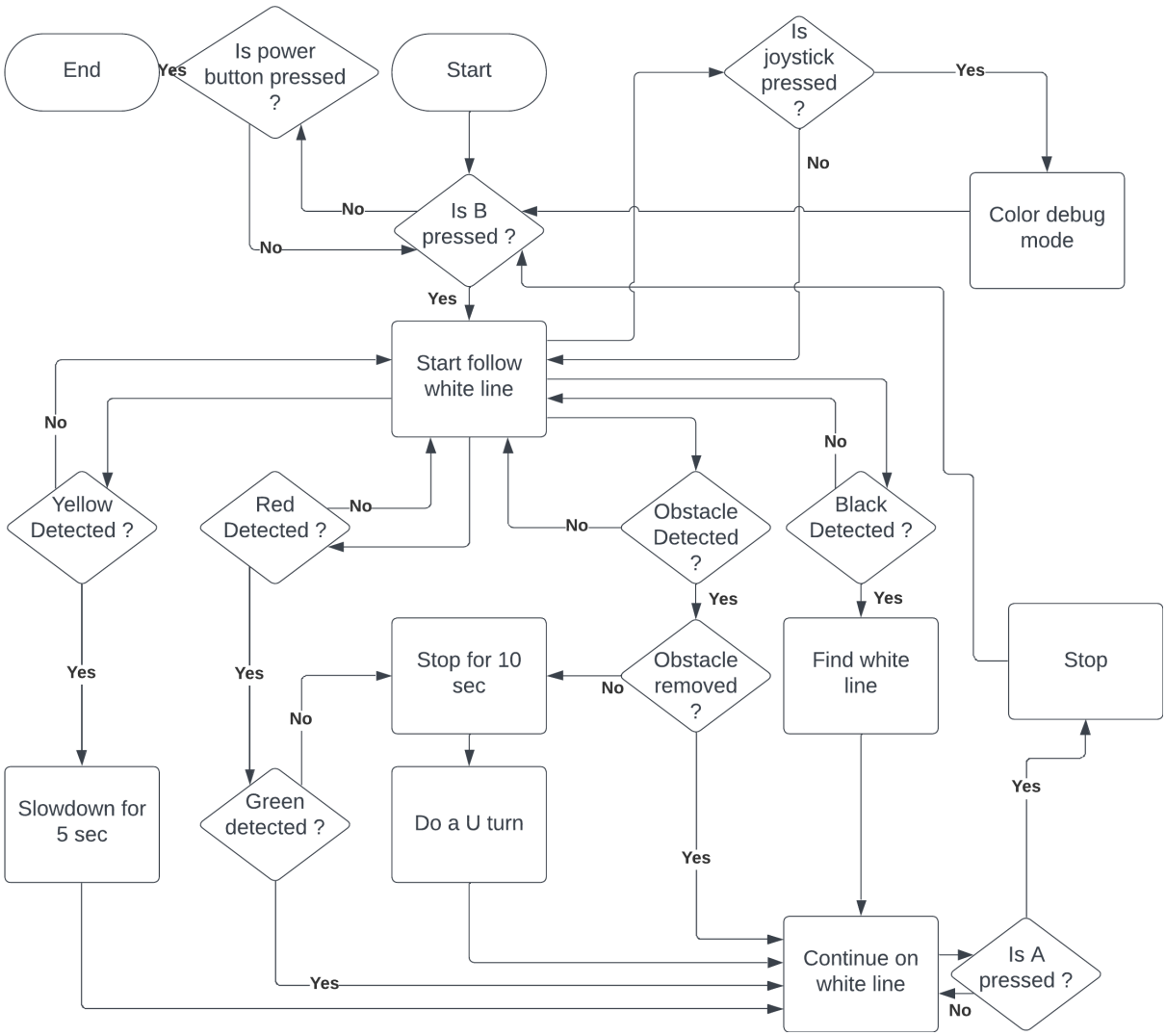

Figure 4: The program work flow

4 Results, Evaluation & Discussion

Overall, the robot demonstrated effective performance in its line-following behavior and responsiveness to environmental cues. The implementation of the Proportional Control Algorithm allowed the robot to track the white line smoothly, with minor drift being corrected in real time using offset-based adjustments.

The robot responded reliably to all color-based signals:

- It slowed down appropriately upon detecting yellow, accompanied by visual and auditory

indicators.

- It paused correctly upon encountering a red signal and waited for a green cue to resume movement.
- The transitions between states were handled with accurate timing and clear feedback via LEDs and tones.

Obstacle detection was also effective when the obstacle completely blocked the path. The robot stopped safely, issued warnings, and resumed motion upon clearance. However, one limitation was observed in scenarios where the obstacle only partially obstructed the path. In such cases, the robot behave the same (wait for the obstacle to be remove) which is unexpected.
When the robot encounters a T-junction and detects a red signal or an obstacle blocking the forward path, its response varies based on internal timing and conditions. In some instances, the robot randomly selects an alternate path—turning left or right to continue navigation. In other cases, it pauses for 10 seconds to check whether the obstacle clears. If the blockage remains after the wait, the robot performs a U-turn to explore a different route. This random yet reactive behavior simulates adaptive decision-making in uncertain environments.
Despite this shortcoming, the robot fulfilled its core objectives and exhibited robust performance under most test conditions. The combination of sensor feedback, control algorithms, and state-based behaviors ensured an overall successful demonstration of autonomous navigation and traffic-signal interpretation.

Future work may involve tuning control parameters further and improving the obstacle avoidance logic to handle partial obstructions more intelligently.

5 Conclusion

This project successfully implemented a line-following robot using proportional control and color detection. The mBot2 demonstrated reliable basic navigation along the track and interaction with environmental cues. While some limitations in obstacle avoidance remain, the system provides a solid foundation for further enhancements in autonomous behavior and control strategies.

References

[1] "What is mBot Neo/mBot2?" Makeblock Support, accessed May 2025. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/1500006183021-What-is-mBot-Neo-mBot2

[2] "Quad RGB Sensor," Makeblock Support, accessed May 2025. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/24279693845527-Quad-RGB-Sensor

[3] "Ultrasonic Sensor 2," Makeblock Support, accessed May 2025. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/24278821629335-Ultrasonic-Sensor-2

[4] "APIs for MBuild Modules," Makeblock Help Center, accessed May 2025. [Online]. Available: https://www.yuque.com/makeblock-help-center-en/mcode/cyberpi-api-mbuild#Xe14m