
# Comparing sensor configurations for line-following LEGO robots in maze navigation

Bachelor Thesis

**Name:** Marvin Knoll

**Email:** [redacted]

**Study Program:** Bachelor of Science (B.Sc.) - Software Engineering

**Semester:** Spring Semester 2023

**Enrollment Number:** 19.01.078

**Supervisor One:** Frank Trollmann

**Supervisor Two:** Ulrich von Zadow

**Date:** February 26, 2023

# Acknowledgments

I want to express my profound appreciation to my supervisor Frank Trollmann for his invaluable patience and guidance. I am also grateful to my second supervisor, Ulrich von Zadow, for his valuable work in helping me to refine my research and achieve a better final result. Finally, I thank Carla Mortensen for her precious help with formatting and vocabulary support.

# Declaration

I hereby confirm that I have written the thesis titled ***"Comparing sensor configurations for line-following LEGO robots in maze navigation"*** by myself, without contributions from any sources other than those cited in the text and bibliography. This also applies to all graphics, drawings, and images included in the thesis.

Furthermore, I confirm that neither this work nor parts of it have been previously or concurrently used as an assessment submission in other courses or in other examination proceedings.

Location, Date: _____ Signature: _____

# Abstract

This paper compares two line-following robots built from the LEGO Mindstorms kit and programmed with ROS2 to solve a simply connected maze. The robots were compared using three metrics: time, energy consumption, and reliability. Both robots used a single colour sensor and differential drive to scan and traverse the maze. The only difference between the two robots was that one could rotate the sensor 180 degrees to scan intersections while the other could not. Both robots drove eleven recorded test runs through the maze to analyse which robot performed better on the three metrics. The robots followed the lines accurately and correctly navigated all intersections. However, due to measurement inaccuracies, it was not possible to determine which robot was more energy-efficient. Nevertheless, the analysis showed that the robot with the rotatable sensor is generally faster at scanning and manoeuvring intersections. Based on these findings, this paper concludes that using a 180-degree rotatable colour sensor can improve scanning and navigation in a maze.

# Contents

# List of Figures



# List of Tables

# 1 Introduction

Line-following robots are low-cost and easy-to-build mobile units that have gained increasing attention in recent years. Applications for such robots range from in-house transportation to path guidance in museums or shopping malls. For these robots to function efficiently, many design decisions must be made correctly regarding sensors, propulsion, and software configurations. Researchers conducted several studies to understand how different configurations affect the robot's efficiency in line-following [1]–[3]. While most studies examine robot designs for following continuous lines, there needs to be more research on robot configurations for following lines with intersections that require scanning and navigation. This study addresses this research gap by comparing two robot configurations in maze solving. The compared robots have a single colour sensor mounted in front of the robot and use the same drive system. The difference between the robots is that one can rotate the sensor horizontally 180 degrees using a motor, while the other robot's sensor is fixed.

This paper first shows and discusses the relevant literature in Chapter 3. Then, it explains the robot configurations and the methodology used to test them in Chapter 4. Next, the thesis discusses the robot-specific test results in Chapter 5. Chapter 6 compares results and answers the research question. Chapter 7 briefly explains the relevance of the results to other robots and makes suggestions for future work.

# 2 Research question and sub-questions

The research question for this study is:

> Is a robot with a 180-degree horizontally rotatable colour sensor more time-, energy-, and cost-efficient than a robot with a fixed colour sensor solving simply connected line mazes?

The sub-questions are:

**$Q_1$:** Is it possible to control the motor that turns the sensor via ROS2 and nxt-python precisely enough to scan intersections and realign the robot?

**$Q_2$:** Which robot configuration is most reliable for following the lines, scanning the intersections and solving the maze?

# 3 Literature review

Line-following robots have been a topic of interest in robotics for many years. The goal is to enable mobile robots to navigate complex environments by following a line with high accuracy and reliability [2]. Various studies explore different approaches for designing robots and implementing the algorithms. This literature review will analyse the existing research on line-follower robots focusing on the different robot and algorithm configurations. The review will examine the motor and sensor configurations before moving on to the various line-tracking algorithms proposed in the literature.

## 3.1 Driving configuration

This section will discuss the existing line-following robot research focusing on the different approaches to making the robots drive. Only the most relevant approaches will be addressed: Ackerman (car-like) drive, two-wheel differential drive and two-wheel balancing drive. The Ackerman drive is similar to a car's steering mechanism and usually consists of two fixed wheels at the back of the vehicle and two steerable wheels at the front. The "two-wheel differential drive" and "two-wheel balancing drive" drive systems both consist of two wheels driven by two individual motors that can be controlled individually. The difference is that the differential drive usually uses a third, non-powered, freely rotating wheel, while the balancing configuration uses sensors and algorithms for balancing.

The Stanford cart described in the work of Moravec [4] is the second mobile robot after "Shakey the robot" [5] and the first line-following robot. The robot used four wheels and Ackerman (car-like) drive for steering. Compared to modern line-follower robots, Ackerman drive is a relatively rare form of locomotion for this type of robot as it is not highly manoeuvrable like other approaches mentioned below.

The most popular approach for line-followers is the two-wheel differential drive with a castor wheel. The authors Pakdaman and Sanaatiyan mention two types of drive: wheels and tank systems [2]. They state that wheel drive is better for these robots but do not explain why. The robot design used for their study

consists of two driven wheels at the back of the robot and a passive castor wheel at the front. This configuration has the advantage of being able to perform tight manoeuvres, turning on the spot and requiring only two motors.

Compared to the stanford cart [4], and the work of Pakdaman and Sanaatiyan [2], a more minimalistic robot locomotion design is the mobile inverted pendulum. The concepts got first implemented in the year 1996 [6] and consist of two wheels on which the robot balances. Fifteen years later, a group of researchers [7] further developed the technology to allow a self-balancing robot to follow lines. The benefits of this driving configuration are that it has a smaller footprint than three or four-wheeled robots while still being highly manoeuvrable and able to turn on the spot [8].

## 3.2 Sensor configurations

The sensor setup used by a line-following robot is the second part that can significantly influence its accuracy. Different configurations of sensors are used depending on the requirements of the robot. These configurations can differ in the sensors' number, type and positioning. This literature review focuses only on the approaches for sensing visual lines and not other types like magnetic lines.

The work of Pakdaman and Sanaatiyan describes the most popular sensor type used for line-following robots: infrared (IR) sensors [2]. IR sensors consist of two diodes, one of which sends infrared rays, and the other measures how much of that ray is reflected. To ensure high accuracy, the authors recommend shielding the sensors from ambient light [2]. The robot designed by Pakdaman and Sanaatiya uses 8 IR sensors in the front of the robot.

Baharuddin et al. specifically compared different IR sensor configurations, which vary in sensor alignment and the number of sensors [1]. The authors found that the number of sensors is vital in determining the resolution with which the robot senses the line. They found that at least two IR sensors are required to follow lines efficiently. To also detect intersections, they concluded that four sensors are needed.

Another, even more, important criterion for good line-following, according to

Baharuddin et al., is the positioning of the sensors [1]. In the configurations with two or four sensors, the sensors are aligned in a line perpendicular to the line the robot needs to follow. The same authors explain why having the sensors too near or too far apart can be problematic and give a formula for the optimal positioning of the sensors depending on the line width [1].

Finally, another study explored using a camera to detect the line [9]. The researchers were able to make this work in a simulated environment but failed to transfer it to the physical robot.

## 3.3 Line tracking algorithms

The third and last crucial part influencing the accuracy of the line-following robots is the algorithm used to control the motors based on the sensor input. Researchers have developed many different algorithms, but it is difficult to determine which performs best as it depends on various factors. This section will discuss the most relevant control algorithms for line-following robots: Simple logic, Proportional-Integral-Derivative (PID) controller, and Fuzzy logic, and mention some less common but valid approaches like Neural Networks and Q-learning.

Pakdaman and Sanaatiyan used simple logic to make the robot drive on the line [2]. They used an if statement to check the values from the IR sensors and turn the wheel motors on or off depending on the values. With this control strategy, the authors concluded that their robot could follow any curves and cycles [2].

According to multiple studies, PID controllers are a common approach for line-following robots [7], [10]–[12]. M. Engin and D. Engin implemented a PID controller and compared it to a simple "on/off" control [12]. Their experimental results show that the robot with the PID controller outperforms the simple control in terms of time to complete the track, driving speed, line tracking smoothness and tendency to stray from the line. However, for PID controllers to work well, they need to be tuned, as mentioned by M. Engin and D. Engin [12]. The tuning can be difficult as it usually has to be done manually.

Fuzzy logic controllers are another control strategy that researchers have found

to be efficient for line-following [13]–[16]. These controllers function based on the mathematical system of fuzzy sets developed by Zadeh [17]. Azlan et al. implemented a fuzzy logic controller and showed its supremacy over the simple logic controllers mentioned earlier [14]. They found that fuzzy logic controllers make the robot follow lines faster, more smoothly and more accurately.

Saadatman et al. used Q-learning to make the robot follow the line [18]. The authors trained the robot in a simulated environment and then conducted the test runs with a real robot. In their tests, the robot that used the simulation annealing-based Q-learning algorithm performed better than the typical proportional controller. In addition, this algorithm has the advantage that it does not need to be manually tuned, unlike the proportional controller.

Finally, Almeida et al. compared different control strategies for autonomous line-following robots [3]. The authors compared a variety of PID, Fuzzy and Neural Network (NN) controllers and found that NN controllers perform the best and simple proportional controllers the worst. Although neural networks perform best, the authors note that training is time-consuming, and the approach is generally unsuitable for microcontrollers with low processing power. Furthermore, they concluded that PD and PID controllers are hard to tune compared to Fuzzy controllers, where the programmer can define the rules more intuitively.

## 3.4 Maze-solving

Line-following robots can be programmed to perform simple tasks, such as following a continuous line, and more complex tasks, such as solving a maze of lines. In these more complex scenarios, the additional challenges include detecting intersections, choosing the correct path, and navigating through the intersections. This section will focus on comparing different algorithms used for maze solving in other studies.

Mishra and Bande presented three algorithms for maze solving using robots [19]. The first algorithm presented is wall following, where the robot follows the right or left wall to find the goal. The authors note that this algorithm only works for mazes with specific properties. Secondly, they analyse Dijkstra's algorithm,

which works for every type of maze. They state that it always finds the shortest path, but it has the drawback that the robot needs to scan the entire labyrinth before calculating the shortest path, which takes a lot of time and energy. Finally, they propose the Flood fill algorithm, which performs better than the other two in terms of the time the robot takes to complete the maze.

Another study compared non-graph-theory (NGT) algorithms (left-wall-follower and right-wall-follower) with graph-theory (GT) algorithms (DFS, BFS and Flood fill) [20]. The authors simulated solving mazes with the different algorithms and concluded that GT algorithms perform better than NGT algorithms. Regarding GT algorithms, they suggest using BFS over Flood fill for giant mazes as it requires less processing power while still performing adequately. If the maze to solve is small, the authors suggest using Flood fill as it finds the shortest path quicker than BFS.

## 3.5 Current state of research

Line-following robots are designed to navigate complex environments by following a line with high accuracy and reliability. Various robot designs were developed and studied by researchers, including different motor configurations, sensor configurations and navigation algorithms. While most research has examined robot configurations for line following without intersections, more research is needed on complex line following where intersections must be scanned and navigated. In particular, line-following robots with moving sensors have been studied very little. Therefore, this work addresses this research gap by comparing a robot with a fixed sensor with a robot with a 180-rotating colour sensor while solving a maze.

# 4 Methodology

This chapter describes the technical specifications and methods used to perform the experiments. This includes a description of the characteristics of the maze, the robot configurations, the software setup and the data collection.

## 4.1 Technical specifications

### 4.1.1 Maze properties

Figure 1 visualizes the maze used for the tests. Both the colours and the proportions correspond to the actual test maze. However, the numbers on the squares are only present in the figure and were added for better illustration.

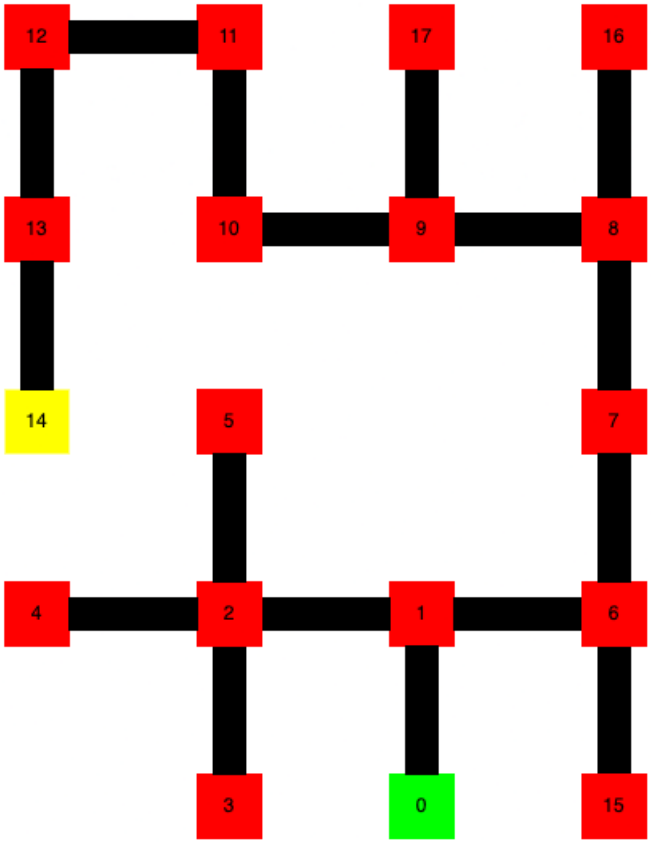



Figure 1: Visual representation of the test maze

The maze comprises 15 squares connected by straight lines that the robot must follow. The green square in Figure 1, represents the start of the maze and the yellow square marks the end. The maze was designed to contain at least one

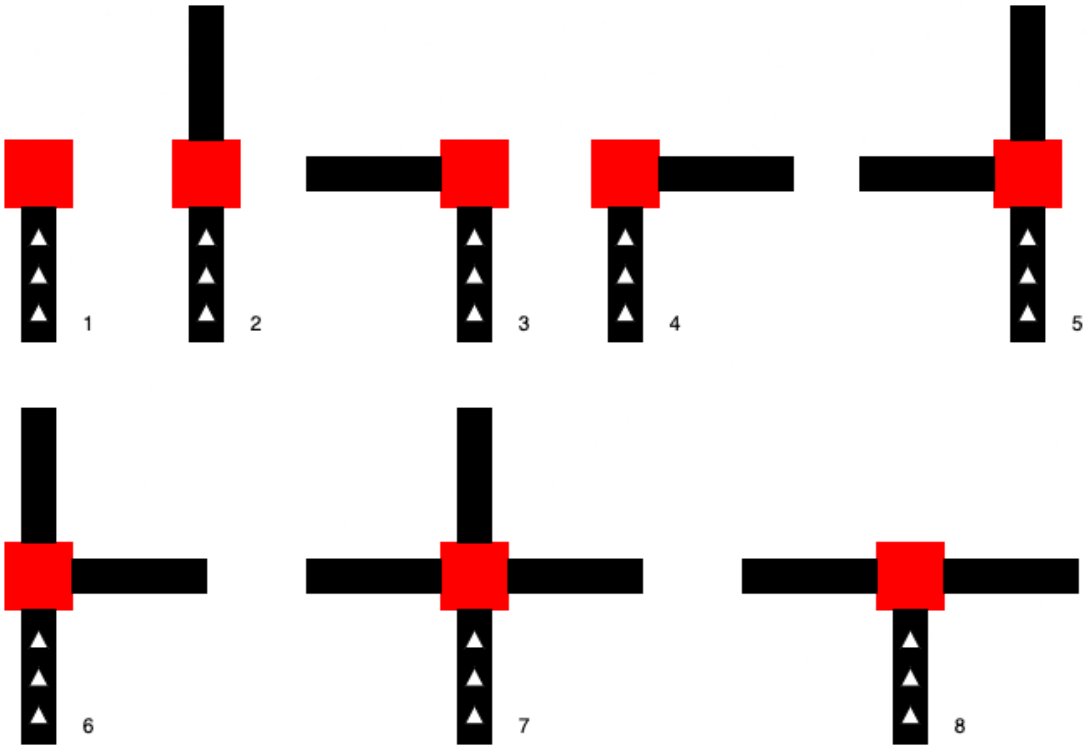



Figure 2: Visual representation of the intersection types

of all the intersection types described in Figure 2 and maintain an acceptable size.

To simplify the intersection scanning and manoeuvring process, only 90° turns are allowed; therefore, the maximum number of paths connected to one intersection is four. These criteria result in the eight intersection types represented in Figure 2. Here the arrows show the path from which the robot approaches the intersection. The test maze represented in Figure 1 contains all eight intersection types.

All red squares (intersections) have a width of 6cm, and the black lines (paths) connecting them have a length of 12cm and a width of 1,8cm. The intersections are wider than the paths, so the robot does not miss them even if it does not approach the intersection perfectly aligned. The length of the paths is optimised to keep the labyrinth as small as possible but, at the same time to give the robot enough space to align itself after an intersection. Different widths of the paths were tested, and it was found that the width of two parallel insulating tapes (1,8 cm) worked best. Too narrow paths cause the robot to overshoot the line during realignment, resulting in repeated attempts at realignment. Paths that are too

wide cause the robot to arrive at an intersection at an oblique angle, resulting in incorrect intersection type detection.

In general, different colours have been used to make it possible to detect intersections start and finish of the maze with a single colour sensor. This is necessary for navigation and for taking time and energy measurements. The robot detects intersections because of their red colour. At the intersections, the robot performs an intersection scan described in Sections 4.1.3.3 and 4.1.3.4, and the time it takes the robot to do this is measured.

To build the labyrinth, a white plastic sheet and coloured insulating tape were used. Both materials are slightly shiny, which is not ideal because of the occasional mismeasurements. For example, "blue" is sometimes measured when the sensor is at the edge of a path. Although the materials are not ideal, they are cheap and allow for quick customization of the maze.

### 4.1.2 Robot configurations

The following sections explain the different robot configurations used for the tests:

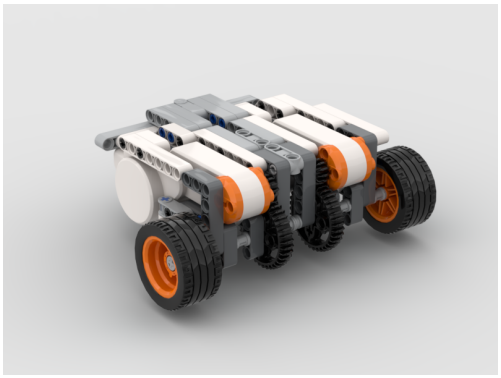
- Fixed-sensor robot (FixRob)
- Turning-sensor robot (TurnRob)

The robot configurations were chosen to test whether a rotating color sensor is advantageous for line tracking and intersection scanning compared to a fixed sensor.

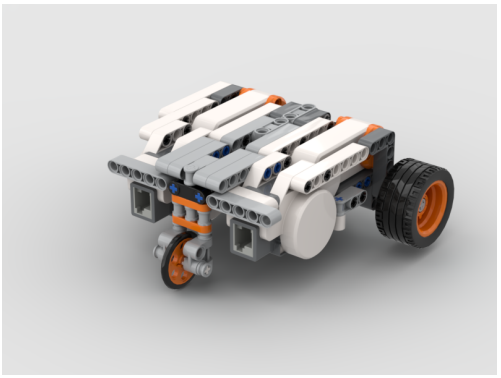
The part of the robot that provides propulsion, namely the two motorized front wheels and the rear steering wheel, remains the same for each configuration. This makes the robots more comparable and enables focused testing of the robots' sensor configuration. What changes between the robots is the sensor configuration for detecting the colour of the ground.

All robots used for these tests were built from the LEGO Mindstorms NXT 2.0[1] kit and custom-designed for this task. The build instructions for the driving

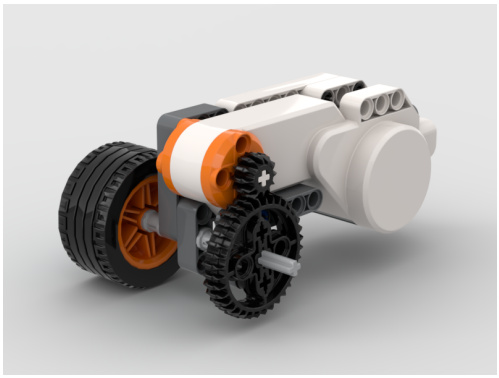
[1] `https://education.lego.com/en-us/downloads/retiredproducts/nxt/software`



(a) Front side



(b) Back side



(c) Drive wheel gears

Figure 3: Render of robot chassis (Created in Bricklink Studio 2.0)

part of the robot can be found in the readme of the nxt_ros2[2] repository.

The robots have two drive wheels at the front, 13.5 cm apart, as visualised in Figure 3a. Both are driven by an individual NXT motor, which enables differential drive. To increase the driving accuracy of the robot, a gear ratio of 3:1 between the motor and wheel was used (see Figure 3c). On the back of the robot is a non-driven, freely rotating castor wheel, which allows the robot to turn on the spot (see Figure 3b).

#### 4.1.2.1 Fixed-sensor robot (FixRob)

The FixRob configuration consists of a single non-moving colour sensor. This sensor is mounted to scan the colour of the ground 9.5cm in front of the robot and is aligned with the robot's centre (see Figure 4a). The sensor is positioned

[2] `https://github.com/marvinknoll/nxt_ros2`

approximately 2-3mm from the ground, which is the ideal distance according to the research of Pakdaman and Sanaatiyan [2]. This robot configuration does not use the motor to which the sensor is attached.

(a) Front side (b) Back side

Figure 4: Maze solving robot on the test maze

This robot configuration costs 110€ considering only the electronic parts. The parts required are:

- 1x NXT Brick - 60€
- 2x Motors - 15€
- 1x Colour sensor - 20€

#### 4.1.2.2 Turning-sensor robot (TurnRob)

The TurnRob configuration comprises one 180° turnable colour sensor. The sensor is attached to an NXT motor mounted on the robot (see Figure 4a). Turning the motor allows the sensor to rotate 90° in both directions. Since the NXT motors do not have an absolute encoder but only a relative encoder, the robot can't know the position of the motor/sensor but only the position relative to its starting point. Thus the sensor must be manually aligned with the centre of the robot before the execution. A 3:1 gearbox between the motor and the sensor improves rotation accuracy.

The idea behind this robot configuration is that only the sensor rotates at the intersection to check which paths are available. Once the scan is complete and the correct path is selected, the robot realigns itself and moves on. The detailed intersection scanning process is explained in Section 4.1.3.4. The sensor does not turn for alignment on the track, but the robot rotates.

The cost for this robot configuration lies at 125€ considering only the electronic parts. The parts required are:

- 1x NXT Brick - 60€
- 3x Motors - 15€
- 1x Colour sensor - 20€

### 4.1.3 Software setup

#### 4.1.3.1 General

The robots were programmed in Python using the Robot Operating System 2 (ROS2). The entire code runs on a host machine that needs to be constantly connected to the robot via a USB cable. The ROS2 package *nxt_ros2*[3], created by the author, was used to integrate the NXT brick into ROS2. The setup is visualised in Figure 5.

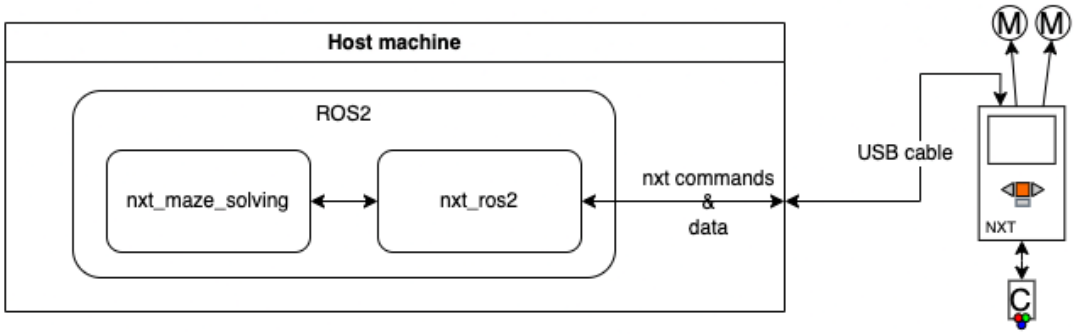



Figure 5: General software architecture

Although this setup has a slower feedback loop between the sensor input and motor output compared to running the code directly on the robot, it allows the use of the various tools provided by ROS2.

The code for making the robots follow lines and drive through the maze was put in a separate ROS2 package called *nxt_maze_solving*[4]. This package uses the ROS2 interfaces provided by *nxt_ros2*[3] to receive sensor data and control the motors.

[3] https://github.com/marvinknoll/nxt_ros2
[4] https://github.com/marvinknoll/nxt_maze_solving

Detailed documentation of how *nxt_ros2* works internally can be found in its Readme file on GitHub[3].

#### 4.1.3.2 Generic maze-solving algorithm

To make the different robots more comparable, they use the same underlying generic algorithm described in Figure 6; for navigating and solving the maze. The strategy to solve the maze is called "left-hand-rule", where the robot always chooses the leftmost path at each intersection.

The robot initially sends a "Start" benchmarking message with the battery voltage and timestamp. It then enters a loop to find the end of the maze. In the loop, it uses sensor data to determine its state. If at an intersection, it scans the leftmost path and takes it. If it loses the line, it tries to align itself with it again. If on the line, it goes straight. If it reaches the end, it stops all motors and sends an "End" benchmarking message. Finally, statistics are calculated and stored before the program terminates.

While the robots all follow the same maze-solving strategy, their methods for scanning intersections, realigning to the leftmost path and determining their states vary.

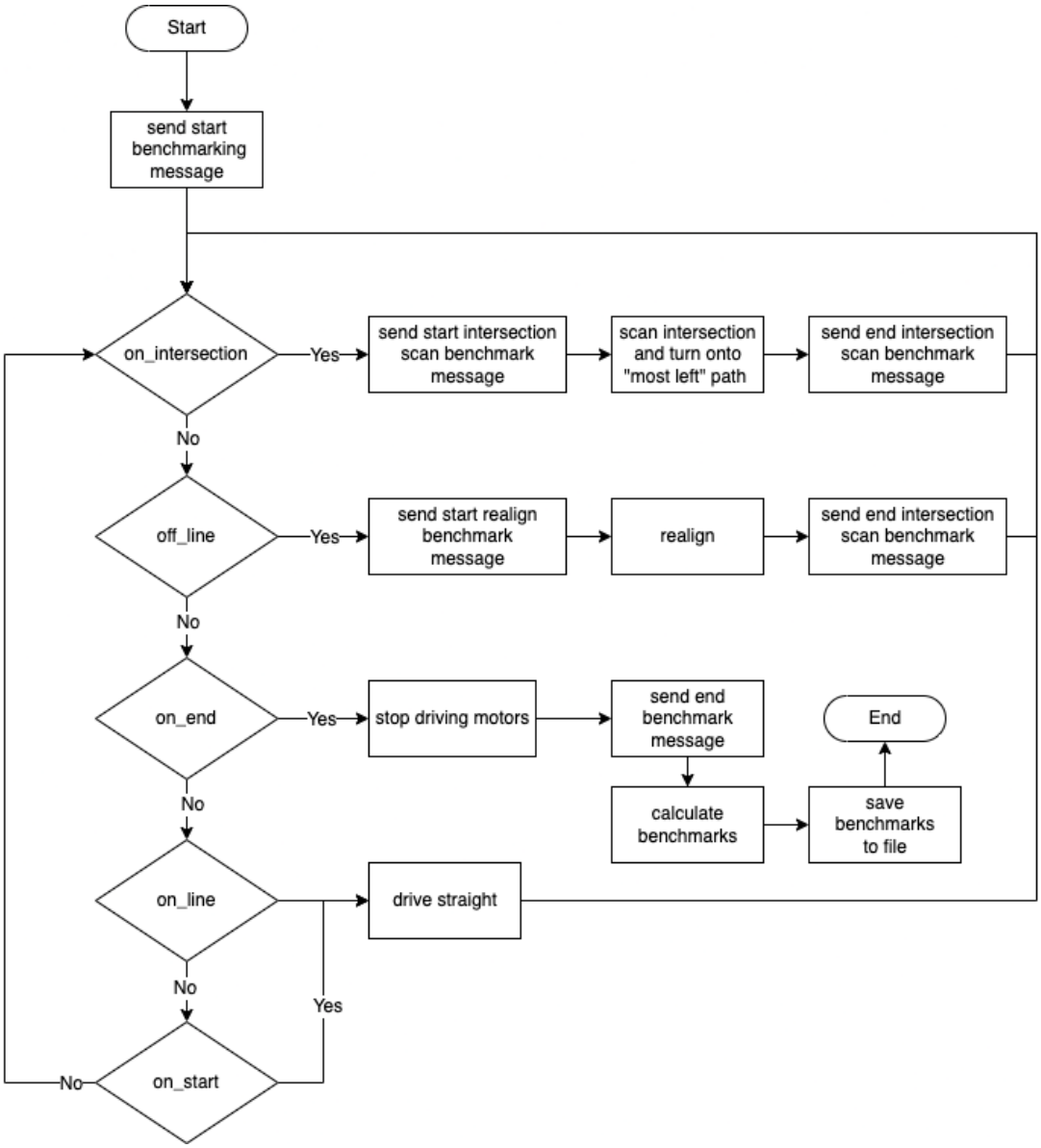



Figure 6: Flowchart of the generic maze-solving algorithm

#### 4.1.3.3 Fixed-sensor robot specific implementations

Table 1 explains the specific implementations of the functions for determining the robot state.

Table 1: Fixed-sensor robot's custom state conditions

| **State** | **Custom Implementation (Pseudo code)** |
|---|---|
| on_line | colour_sensor_value == black |
| off_line | (colour_sensor_value == white) and (scanning_intersection == False) |
| on_intersection | (colour_sensor_value == red) |
| on_start | colour_sensor_value == green |
| on_end | colour_sensor_value == yellow |

The intersection scanning and realignment strategies are implemented as state machines to keep the code more readable. For example, the intersection scan sequence was divided into several individual operations, each corresponding to a state machine's state.

**State machine for intersection scanning and navigation** First, the robot positions itself so its centre of rotation is in the middle of the intersection. In the next step, the robot rotates 135 degrees counterclockwise around its axis to ensure it is on the left of a possible left path. Finally, the robot rotates clockwise until it finds a path, which must be the one furthest to the left and, therefore, the path to take. The number of degrees by which the robot has turned can be used to determine which path has been taken. Now the robot is aligned with the path taken and ready to drive forward to the next intersection. Figure 12 in the appendix shows a detailed diagram of this state machine.

**State machine for realignment on the line** When the robot gets "off_line", it begins a realignment process using a state machine. This makes the robot turn on the spot in one direction and then the other to find the line. The robot uses two strategies to optimise this process. First, it turns only 20 degrees in both directions to quickly find the line and avoid turning in the wrong direction for too long. If it still does not find the line, it expands the search to 60 degrees in both directions. The second strategy is that the robot remembers which way

it turned at the last intersection, allowing it to make an educated guess about which side of the line it is on when it misses it. This allows the robot to turn in the direction where it expects to find the line and find it more quickly.

#### 4.1.3.4 Turning-sensor robot specific implementations

This robot configuration uses the custom implementations described in Table 2 to distinguish between the individual states.

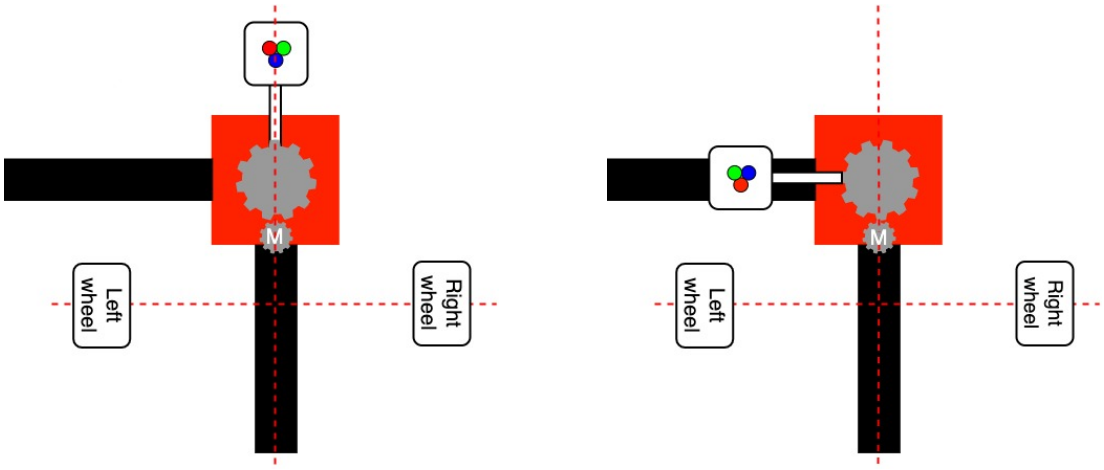
Table 2: Turning-sensor robot's custom state conditions

| **State** | **Custom Implementation (Pseudo code)** |
|---|---|
| on_line | colour_sensor_value == black |
| off_line | (colour_sensor_value == white) and (scanning_intersection == False) |
| on_intersection | colour_sensor_value == red |
| on_start | colour_sensor_value == green |
| on_end | colour_sensor_value == yellow |

The intersection scanning and realignment strategies are again implemented using state machines. The realignment process for the turning-sensor robot is the same as for the fixed-sensor robot and is described in Section 4.1.3.3.

**Strategy for intersection scanning and navigation** Once the robot is in the "on_intersection" state, it repositions itself so that the colour sensor's centre of rotation is at the centre of the intersection (see Fig. 7a). It then rotates the sensor 90 degrees counterclockwise (see Fig. 7b). Next, it reads the sensor values while turning the sensor clockwise 135 degrees. If the robot detects the line during this process, it knows it has found the line. Depending on the angle of the line, the robot can determine whether the line represents the left or straight path. If the robot does not detect the line, it repositions the sensor at a 0-degree angle and then turns clockwise on the spot until it finds the right or back path.

This strategy avoids turning the robot in the wrong direction by scanning the left and straight path by turning only the sensor. A detailed diagram of this state machine can be found in the appendix in Figure 13.




(a) Initial position (b) Scan to the left

Figure 7: Visual representation of TurnRob's intersection scanning strategy

## 4.2 Data collection

The data collection process included 11 test runs for each robot configuration, performed sequentially. The tests were conducted on November 18, 2022, for the fixed-sensor robot and on November 22, 2022, for the turning-sensor robot. The test maze was set up in a closed room, with the only light source being a conventional ceiling lamp.

The robot was powered by rechargeable nickel metal hydride (NiMH) batteries, which were fully charged and allowed to cool to room temperature before each test sequence. The charging and plugging order was the same for each test. The room temperature was maintained at 19-20°C for each robot configuration and test run.

Several measurements were taken during each test run. These included the total duration of the run, the start and end battery voltage in millivolts, and the duration of each intersection scan and realignment. Also counted were the number of realignments and whether the run was completed successfully or the robot failed to reach the end of the maze.

The decrease in battery voltage was calculated using the average voltage measurement taken at the start and end of each test. Both the start and end voltages were determined by taking 20 consecutive measurements and calculating the average of those readings. The data was collected, processed and then saved to a

file using the measurement feature of the *nxt_maze_solving* package and analysed in Google Sheets.

Since LEGO stopped selling some of the parts used in the robots, an average of the prices on eBay was calculated to estimate the cost of the robots.

The data collection method used allowed for accurate and reliable results. The controlled conditions ensured the data remained comparable and reproducible across all test runs. This allowed effective evaluation of the robot's performance.

# 5 Results and robot-specific discussions

This chapter presents and discusses the results of the tests with FixRob and TurnRob. The robots were tested under controlled conditions described in Section 4.2. The main focus of this chapter is analysing and discussing the robot-specific benchmarks in terms of speed, intersection navigation, power consumption and cost. The raw data of the measurements and the analysis tables can be found on GitHub[5].

## 5.1 Fixed-sensor robot

### 5.1.1 Results

Table 3 summarises the key statistics for the eleven runs of the robot configuration with a single fixed sensor. On average, the complete runs took 334.77 seconds and had a relatively low standard deviation of 9.4 seconds. The most time was spent at intersections, with an average of 243.17 seconds, while the least time was spent on realignments, at 34.39 seconds on average. The standard deviations of the measured times show that the time spent on the line was the most constant, while the time spent realigning was the least constant. The average voltage difference of the battery before and after the test runs was 57.55 millivolts.

Table 3: Fixed-sensor robot - Summary of key statistics

| | Time [s] | | | | | |
|---|---|---|---|---|---|---|
| | Total | Intersections | Black lines | Realignments | #Realignments | Voltage difference [mV] |
| Average | 334.77 | 243.17 | 57.21 | 34.39 | 26.18 | 57.55 |
| Sum | 3682.45 | 2674.83 | 629.31 | 378.31 | 288 | 633 |
| Minimum | 323.083 | 239.66 | 56.07 | 23.09 | 17 | 5 |
| Maximum | 350.168 | 245.18 | 58.27 | 49.43 | 32 | 136 |
| Standard Deviation | 9.49838 | 1.85923 | 0.65651 | 8.27200 | 3.97034 | 36.0398 |

Figure 8 shows the battery voltage before and after each of the eleven runs, indicating the robot's power consumption.

Figure 9 shows the average time that the robot required to complete the individual intersection types described in Figure 2. Dead ends, with an average of 22.7 seconds, took the robot the longest to scan and navigate. In contrast, "left

[5] https://github.com/marvinknoll/nxt_maze_solving/tree/main/benchmarks

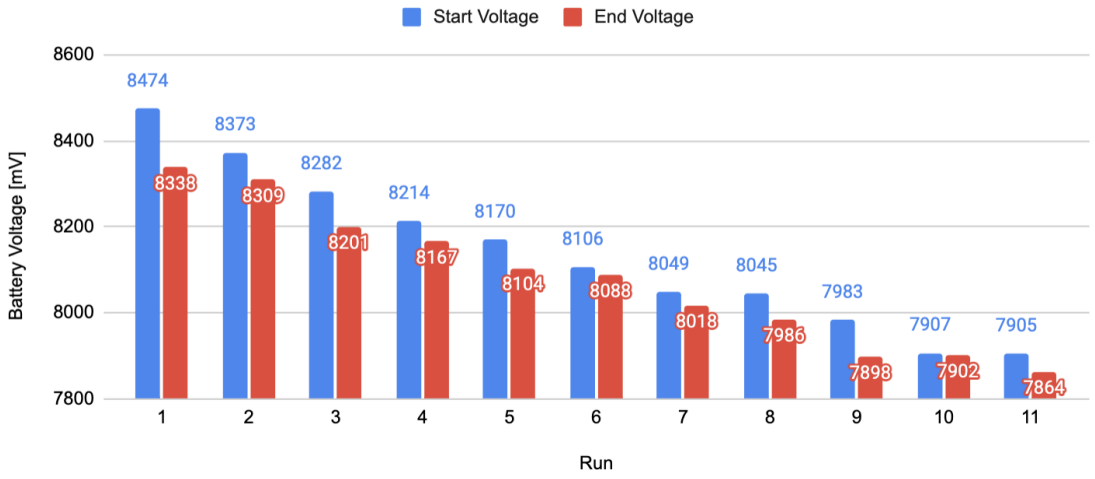



Figure 8: FixRob - Comparison of start and end battery voltage per run

& straight", "left & straight & right", "left", and "left & right" intersections took the least time with averages between 10.87 and 11.03 seconds.

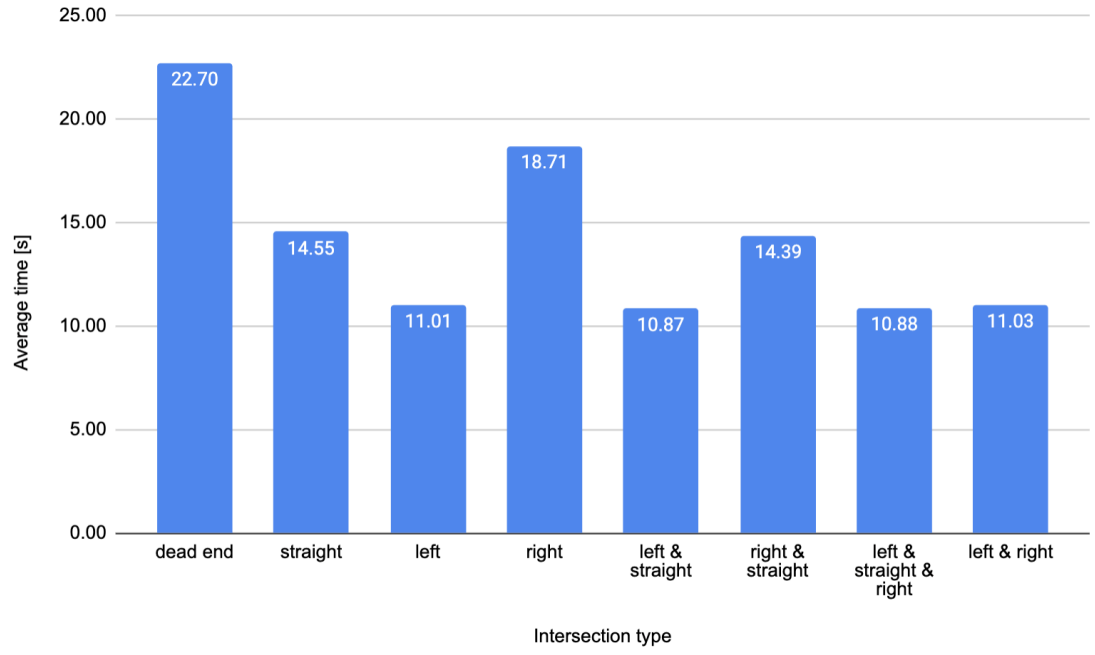



Figure 9: FixRob - Average time per intersection type

### 5.1.2 Discussion

The standard deviation of 9.498 for the total run duration indicates the consistent performance of the robot. From the standard deviation of 8.272, it becomes clear that the time for realignment is the least consistent one. These deviations are understandable because the robot has only one sensor and therefore does not

know in which direction to reorient itself to find the line. Although the robot tries to make an educated guess about which direction to scan first, it sometimes scans in the wrong direction, resulting in a higher standard deviation.

The collected data allowed for calculating the time the robot spent exclusively on the line. This was done by adding the time spans between intersection scans and realignments. The granular time measurements also allowed for calculating the time spent between two specific intersections. It was found that the time between the last intersection (#13) and the end of the maze (#14) averaged 13 seconds, while all other lines between two intersections averaged only 2.57 seconds. This difference is present because the detection of the end of the maze is relatively slow, and the timing stops only after the end has been detected and the driving motors are still.

The high standard deviation of 36.04 in the battery voltage data and the fact that successive runs' end and start voltages differ from each other suggest significant measurement inaccuracies. It is not fully clear why the measurements are so inconsistent. However, it is known that factors such as temperature or start voltage can strongly influence the measured voltage. Another factor which makes it harder to measure the battery voltage is that the voltage of NiMH batteries can vary significantly during discharge. Finally, it is also possible that the NXT brick is inaccurate in measuring the battery voltage.

The data in Figure 9 show the average time spent at the different intersection types. In the chart, four groups of intersection types with similar times can be identified. The group elements have in common which direction the robot finally takes at the intersection.

An interesting observation is that *right* turns take longer than *left* turns. This is because the robot turns 135 degrees counterclockwise to find a possible *left* path, and when it finds it, it takes it. Also, for a *right* turn, it first rotates 135 degrees counterclockwise to ensure no *left* path is available and then must rotate 225 degrees clockwise to align with the *right* path, which takes more time.

The most time was required for *dead ends*, which makes sense since the robot needs to turn the farthest to scan and navigate such intersections. As Section 4.1.3.3 describes, the robot turns 135 degrees counterclockwise to ensure there

is no *left* or *straight* path available and then 315 degrees clockwise to check the availability of the *right* path and, finally, head back as it is a *dead end*.

In summary, the results have shown that the robot requires the least time for the highest priority turn of *left*. For the remaining turns, *straight*, *right*, and *back*, more and more time is required in descending order of priority.

The cost of the robot is the bare minimum for a line-following robot built with LEGO Mindstorms, as all the electronic parts are necessary. However, what could undoubtedly be reduced are the remaining parts that make up the robot.

## 5.2 Turning-sensor robot

### 5.2.1 Results

Table 4 presents the critical statistics for the robot configuration with a single turning sensor. On average, this robot took 333.3 seconds to complete a single run with a relatively low standard deviation of 9.28775. The average time spent scanning and navigating intersections was the longest at 240.20 seconds, while the average time spent realigning when the line was lost was the shortest at 36.79 seconds. With a standard deviation of 0.829523, the average time of 56.41 seconds spent on the line was the most constant, and the average time for realignments at 36.79 was the least constant with a standard deviation of 8.094613. The average voltage difference of the battery before and after the test runs was 48.64 millivolts per run, with a standard deviation of 19.63299.

Table 4: Turning-sensor robot - Summary of key statistics

| | Time [s] | | | | | |
|---|---|---|---|---|---|---|
| | Total | Intersections | Black lines | Realignments | #Realignments | Voltage difference [mV] |
| Average | 333.40 | 240.20 | 56.41 | 36.79 | 25 | 48.64 |
| Sum | 3667.45 | 2642.22 | 620.51 | 404.72 | 275 | 535 |
| Minimum | 317.782 | 235.01 | 54.78 | 26.40 | 20 | 21 |
| Maximum | 347.078 | 244.50 | 57.56 | 48.10 | 33 | 82 |
| Standard Deviation | 9.28775 | 3.221237 | 0.8295 | 8.094613 | 4.449719 | 19.63299 |

Figure 10 shows the battery voltage before and after each of the eleven runs, indicating the robot's power consumption.

The chart in Figure 11 shows the average times required to scan and navigate each intersection type described in Figure 2. The robot requires the most time

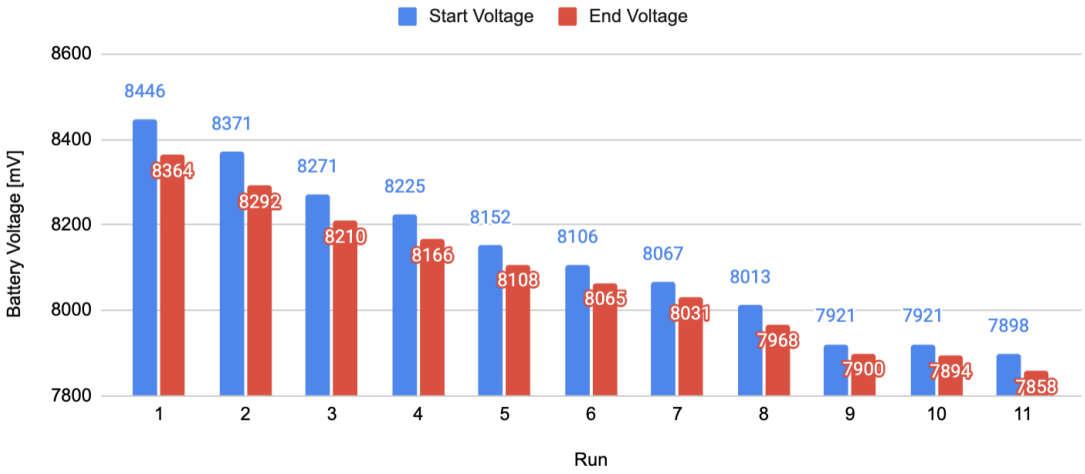



Figure 10: TurnRob - Comparison of start and end battery voltage per run

to complete "dead end" intersections and the least for "right & straight" and "straight" intersections.

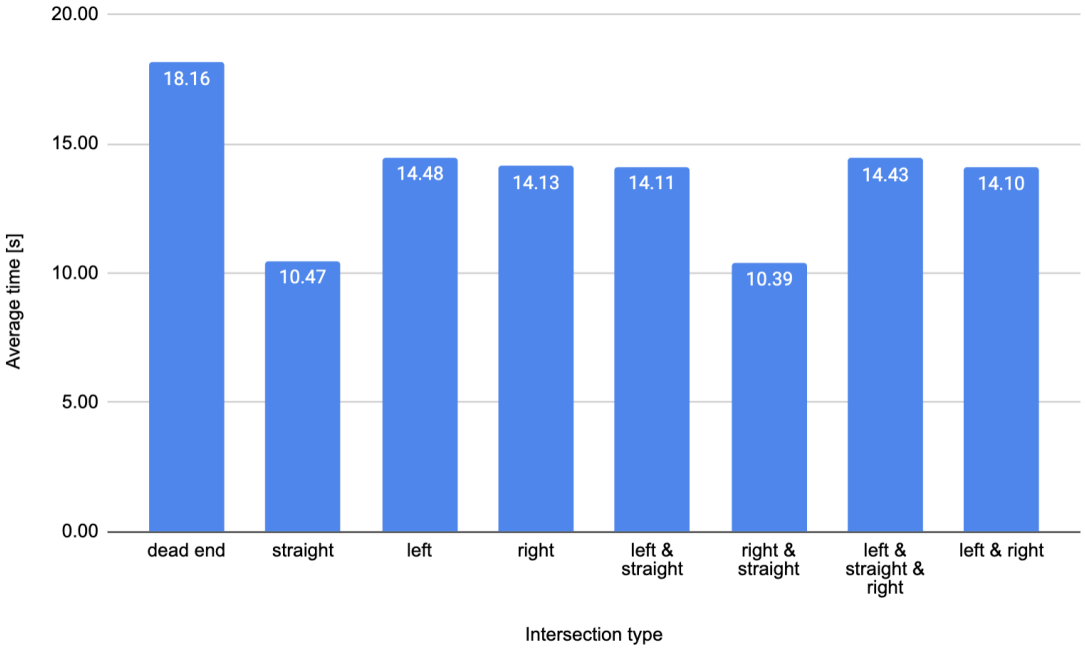



Figure 11: TurnRob - Average time per intersection type

### 5.2.2 Discussion

From the results, it is clear that also this robot configuration has consistent time in navigating through the maze, indicated by the relatively low standard deviation of 9.287. The deviations in time required for realignments are denoted by the standard deviation of 8.094 and have the same origin as the robot with

the fixed sensor. Further explanation of why this deviation occurs can be found in Section 5.1.2.

The time this robot spends on the line was calculated the same way as for FixRob. Again, it was found that the average time spent on the line between intersections #13 and #14 was 10.45 seconds, and the average time spent on all other individual lines between intersections was only 2.55 seconds. This difference is present because the detection of the end of the maze is relatively slow, as already described for FixRob in Section 5.1.2.

For this robot, it is possible to detect a trend in the difference in battery voltage before and after each run by looking at the chart in Figure 10. The voltage drops steeply during the first runs because NiMH batteries' voltage also drops sharply at the beginning and then flats out. However, it can be seen in the graph that the last four measurements seem to be inaccurate, as it is expected that the consumed voltage becomes more and more constant as the voltage of the battery drops.

The results of the measurements of the average time required per intersection type in Figure 11 show three groups characterised by very similar averages. The first group includes intersections where the robot continues driving *straight*, averaging between 10.39 and 10.47 seconds. The second group contains intersections where the robot turns *right* or *left* and is recognised by an average between 14.11 and 14.48 seconds. The last group contains the *dead end*, which takes 18.16 seconds on average.

The data show that *dead ends* take the longest to scan and navigate. This is because the robot first turns the colour sensor to scan if there is a *left* or *straight* path available and then needs to turn 180 degrees to drive *back* and exit the *dead end*. Since the robot has to turn the farthest here to realign with the path, this intersection type takes the most time to navigate.

Intersections where the robot finally drives *straight* are the fastest ones to complete as the robot directly continues driving forward after the scan is complete and does not need to turn.

As expected, the results show that the robot required the same amount of time for both *left* and *right* turns. This outcome is because the robot first scans

for potential *left* and *straight* paths and then, depending on the result, turns 90 degrees to the *left* or *right* to align with the upcoming path. Since the scanning process is always the same and turning 90 degrees to the *left* or *right* takes up the same amount of time, the average *left* and *right* turns at intersections are the same.

## 5.3 FixRob vs. TurnRob: Intersection time results

This section presents data that compares FixRob and TurnRob in terms of the time it takes them to scan and navigate through the eight intersection types represented in Figure 2. These results are discussed in Section 6.1.1.1.

Time differences at intersections were the greatest compared to time spent realigning or on the paths(see Figure 3 and 4). Table 5 unifies the data from Figure 9 and Figure 11 and shows the difference in the time the two robots require to complete the different intersection types in seconds and percentage. The data show that FixRob is faster only at intersections where it finally takes the *left* path. In contrast, TurnRob is faster than FixRob at all other intersection types, where the robot ultimately drives *straight*, *right* or *back*.

Table 5: Comparison of both robot's intersection types statistics. (*Path taken* is **L**eft, **S**traight, **R**ight, **B**ack from the perspective where the robot arrived at the intersection.)

| Intersection | | | FixRob compared to TurnRob (Avg.) | |
|---|---|---|---|---|
| Id | Description | Path taken | Single intersection[s] | Percentage[%] |
| 1 | dead end | B | +4.54 | +25.00 |
| 2 | straight | S | +4.08 | +38.97 |
| 3 | left | L | −3.46 | −23.92 |
| 4 | right | R | +4.58 | +32.42 |
| 5 | left & straight | L | −3.24 | −22.94 |
| 6 | right & straight | S | +4.00 | +38.52 |
| 7 | left & straight & right | L | −3.55 | −24.61 |
| 8 | left & right | L | −3.07 | −21.80 |
| Sum | | | +3.88 | +3.52 |

The sum of the time required for TurnRob to complete each of the eight intersection types is 3.88 seconds (3.52%) less than that of FixRob. Therefore, for a maze with exactly one intersection per intersection type, TurnRob is expected to take 3.88 seconds (3.52%) less to navigate through the intersections than FixRob.

According to these calculations, the time difference between the two robots for intersection scanning and navigation for the entire test maze is estimated to be 3.85 seconds. The average total intersection time for FixRob is 243.17 seconds and 240.20 seconds for TurnRob.

# 6 Discussion and Research Limitations

This chapter compares and discusses the results obtained with the two robot configurations, answers the research question and sub-questions and finally notes the limitations of this study.

## 6.1 Comparison of the robot configurations

The following sections compare the robots regarding time, number of realignments, power consumption and cost. The time comparison is split into three categories: time at intersections, time for realignments and time driving on the paths.

### 6.1.1 Time Measurements

#### 6.1.1.1 Scanning and navigating intersections

The results presented in Section 5.3 show that FixRob is faster than TurnRob only at intersections, where the robot eventually turns *left*. This is due to how the robots are programmed to scan and navigate at intersections. While FixRob immediately turns left, sees the *left* path and takes it, TurnRob first turns the sensor to scan for a left path and then turns left to take it.

Although FixRob initially turns 135 degrees to the left to avoid missing the *left* path in case it does not enter the intersection perfectly aligned, this 45-degree turn and subsequent turn back to the *left* path is less time-consuming than the scan that TurnRob performs.

On the other hand, TurnRob is faster than FixRob at intersections where it does not turn left because ensuring that there is no *left* path by rotating the sensor takes less time than rotating the entire robot 135 degrees counterclockwise and then turning back to take the *straight*, *right* or *back* path.

Finally, Table 5 shows that TurnRob is only 3.88 seconds or 3.52% faster than FixRob when scanning and driving through all eight intersection types once. Although this time improvement is not substantial, it is worth mentioning because it was constrained by LEGO Mindstorms hardware limitations. Since the motor

for turning the sensor had to be turned relatively slowly, the time saving was not that great. If motors with high accuracy at higher speeds were used, the time improvement over FixRob could be much more significant.

#### 6.1.1.2 Driving on the paths

As expected, both robots performed similarly regarding time spent on the black lines. Both robots used the same drive system and drove at the same speed. This resulted in FixRob driving 57.21 seconds and the TurnRob driving 56.41 seconds purely on paths. It is important to note that according to the measurements, both robots took significantly longer for the path between intersection #13 and #14 than for the rest of the paths. This is due to a measurement error and is discussed in more detail in Sections 5.1.2 and 5.2.2.

#### 6.1.1.3 Realignments to the path

The time spent for realignments when the line was lost was similar for both robot configurations. This was expected as the robots use the same realignment strategy described in Section 4.1.3.3.

### 6.1.2 Number of realignments

The average number of realignments per run indicates how precisely the robot was aligned with the path after passing through an intersection and how accurately it followed paths. This number was similar for both robots. FixRob performed 26.18 realignments on average per run, while TurnRob realigned 25 times on average. Since the robots must traverse eighteen black paths to complete the maze, FixRob realigns 1.45 times and TurnRob 1.39 times per path. These numbers are quite large, considering that the paths are straight.

Although no definitive conclusion can be drawn, the similar number of realignments between the robots suggests that both were equally well aligned after scanning and navigating intersections. This is because if one of the robots had not been, it would have needed more realignments on average than the other.

### 6.1.3 Power consumption

Energy efficiency is another crucial factor in the design of a line-follower robot. During the tests, the power consumption of each robot was measured, but the measurements were inaccurate. The inaccuracy is reflected in the high standard deviation, making it difficult to conclude which robot is better for power consumption. Possible reasons for this are described in Section 5.1.2.

TurnRob was expected to consume less power than the FixRob. The hypothesis was that TurnRob only needed to turn the motor that rotates the sensor to scan for left and straight paths. On the other hand, FixRob required turning both drive motors to scan intersections, which should have consumed more power. This would have saved TurnRob energy, especially at intersections where the robot finally does not turn left. This is because TurnRob does not have to turn 135° to the left and then 135° back to drive straight, right or backwards out of the intersection.

### 6.1.4 Cost

The cost analysis of the two robots showed that TurnRob is less cost-effective than FixRob due to its additional motor. The difference between the two robots is about 15 Euros considering only the electrical parts. It should also be noted that FixRob's costs are minimal since none of the electrical components can be removed without losing the ability to follow lines and solve mazes.

## 6.2 Answering the research question and sub-questions

### 6.2.1 Research question

The research question of this study was: **"Is a robot with a 180-degree horizontally rotatable colour sensor more time-, energy-, and cost-efficient than a robot with a fixed colour sensor solving simply connected line mazes?"** To answer this question, a series of measurements and analyses were performed. In terms of speed, both robots performed similarly in the test maze. While the more detailed comparison of the individual time categories in Section

6.1.1 showed that both robots performed similarly in realignment and navigation on the line, it was found that TurnRob generally performed slightly better than FixRob in scanning and navigating intersections.

Which robot is better in energy consumption could not be determined due to inaccurate measurement results. Nevertheless, TurnRob is the recommended robot in terms of energy consumption since it should consume less power according to the theory described in Section 6.1.3. More details about the measurement inaccuracies can be found in Section 5.1.2.

Finally, when comparing the robots in terms of cost, FixRob is slightly cheaper than TurnRob. Since TurnRob uses an additional motor, it is 15 Euros more expensive than FixRob.

### 6.2.2 Sub-questions

**$Q_1$: Is it possible to control the motor that turns the sensor via ROS2 and nxt-python precisely enough to scan intersections and realign the robot?**

TurnRob's eleven successful runs showed that the robot could rotate the sensor precisely enough to scan intersections. As described in Section 4.1.3.4, the robot first rotates the sensor to the left to check if the left path is available and then checks the availability of the straight path by turning the sensor back. If precise control had not been possible, the robot would not have been able to traverse the maze because it would have missed paths at intersections. Although the accuracy of the motor was not measured directly, it was also observed that it returned precisely to its original position after the scan was completed.

However, the NXT motor's precision and speed are insufficient for realigning the robot to the line. To implement this process efficiently, it would be necessary to turn the sensor only a few degrees to determine as quickly as possible which side the line is in relation to the robot and then correct the trajectory. However, as described in the documentation of the underlying library for controlling the motors, it is not possible or too inaccurate to rotate the motor less than 50 degrees [21].

**$Q_2$: Which robot configuration is most reliable for following the lines, scanning the intersections and solving the maze?**

The eleven successful runs of both robots show that they have very similar results in terms of reliability when solving the maze. Each robot could reliably follow the line, with FixRob requiring an average of 26.18 realignments and TurnRob needing 25 per run. Both robots could scan and navigate all intersections error-free during the test runs. In summary, both robots performed similarly regarding reliability when solving the maze.

## 6.3 Limitations

Although many things were considered, this study has certain limitations. The following points are intended to show under what conditions the results are valid and to help future researchers avoid the same shortcomings.

An explicit limitation of the study is the inaccuracy of the measurements of the robot's power consumption, due to which the robots could not be compared in terms of energy consumption. The battery voltage measurement function built into the robot was expected to provide sufficiently accurate data. However, the high standard deviation of the voltage data indicates substantial measurement inaccuracies. It is not clear why the data deviated so much. As described in the data collection Section 4.2, efforts were made to keep the same conditions for both robots to make the results more comparable. However, the battery voltage measurements are affected by many factors, as explained in Section 5.1.2, and the NXT Brick likely gave inaccurate readings.

Another limitation is the small variety of robot configurations. Only two configurations were designed and tested for this study. This limits the results to robots with a single sensor and differential drive control. Analysing configurations with multiple sensors would be interesting, as they would allow for more efficient realignment or intersection scanning strategies. These improvements could impact the speed of the robot and potentially save energy.

A final limitation is that the robots were only tested in a single maze. The maze is shown in Figure 1, and the detailed properties are described in Section 4.1.1. Although the test maze was designed to include all possible intersection types shown in Figure 2, testing the robots on several different mazes would have made the results more robust.

# 7 Conclusion

Line-following robots are used in various industries as they are reliable, cheap and easy to build. As the designs of these robots vary in driving, sensing and controlling setups, it is crucial to understand how the different configurations affect efficiency. By testing two different robots built out of the popular LEGO Mindstorms NXT 2.0 kit inside a maze, this study found that the robot using a motor for turning the colour sensor for scanning intersections is slightly faster than the robot with a single fixed colour sensor. Hereby the time savings could be increased if faster and more accurate motors were used. Thus, the results suggest using a rotating sensor for efficient intersection scanning when only a single colour sensor is used.

## 7.1 Relevance of the results for other robots

To determine if the results are relevant for conventional robots, it is necessary to analyze how similar the LEGO Mindstorm NXT 2.0 components are compared to conventional components. The electronic LEGO MINDSTORMS parts used in this study are the colour sensor, the motors, and the NXT brick. Here, the colour sensor is comparable to the widely used TCS230/TCS3200 sensor, and the motor can be compared to a conventional 9V DC motor combined with an optical encoder and a 1:48 gearbox. On the other hand, the NXT device is difficult to compare, but an Arduino Mega or a Raspberry Pi could be used to read and control the sensors and motors. Therefore, the LEGO parts are generally well-comparable with standard components.

What has not yet been discussed, but could lead to different results, is how the robots in this study and conventional robots are programmed. As described in Section 4.1.3.1, FixRob and TurnRob were programmed with nxt_ros2 running on a computer connected to the robot. While this has several advantages, it has the disadvantage of having a relatively slow feedback loop between the sensor input and motor output. In contrast, conventional line-following robots are usually designed with all the code running on the robot's microcontroller. This allows these robots to react much faster and more accurately to sensor input, which

means they can follow lines and scan intersections more precisely and quickly than the LEGO robots. However, conventional robots could also be programmed like the robots in this study. The ROS2 package "nxt_maze_solving" created for this study was implemented in a very generic way so that non-LEGO robots could be easily integrated.

In summary, conventional robots of similar size built from the above components and programmed in the same way as the robots in this study should achieve similar results. However, it must be said that non-LEGO robots could also use faster, more precise, and more energy-efficient components, which would lead to different results.

Finally, it should be mentioned that the time ratios should be similar for conventional robots. This means these robots will also likely spend most of their time at intersections compared to time on the track or for realignments. In addition, these robots will also spend more time scanning and navigating specific intersections than others. However, since no tests have been performed, making a definitive statement about this is impossible.

## 7.2 Future work

This final section revisits the limitations identified in Section 6.3 and suggests how future researchers could avoid the same shortcomings.

The inaccuracies in the power consumption data made it impossible to compare the two robots in this dimension. As mentioned earlier, this was due to several factors, one of which was that the power consumption was measured using the NXT device. Since this likely provided inaccurate measurement results, future studies could use external devices to measure power consumption.

A further limitation mentioned in Section 6.3 was the small variety of robot configurations. Future studies could implement different sensor configurations and experiment with external, non-LEGO sensors by integrating them through ROS2. Researchers could also test different driving systems, such as Ackerman or holonomic systems, to better understand their advantages and disadvantages in different scenarios.

In addition to the small number of different robot configurations, testing on a single maze was also a limitation of the study. In the future, scientists could build more test mazes or simulate them. Since the entire project is based on ROS2 and parts of the robot models already exist, the simulation of the robots would require little additional work.

Finally, to see if the results of this study are relevant and transferable to conventional non-LEGO robots, scientists could build robots and test them in the same maze.

# References


[1] M. Z. Baharuddin, I. Z. Abidin, S. S. K. Mohideen, Y. K. Siah, and J. T. T. Chuan, "Analysis of line sensor configuration for the advanced line follower robot", *University Tenaga Nasional*, 2005.

[2] M. Pakdaman and M. M. Sanaatiyan, "Design and implementation of line follower robot", in *2009 Second International Conference on Computer and Electrical Engineering*, vol. 2, 2009, pp. 585–590. DOI: 10.1109/ICCEE.2009.43.

[3] L. Almeida, A. Mota, and P. Fonseca, "Comparing control strategies for autonomous line-tracking robots", in *AMC'98 - Coimbra. 1998 5th International Workshop on Advanced Motion Control. Proceedings (Cat. No.98TH8354)*, 1998, pp. 542–547. DOI: 10.1109/AMC.1998.743594.

[4] H. Moravec, "The stanford cart and the cmu rover", *Proceedings of the IEEE*, vol. 71, no. 7, pp. 872–884, 1983. DOI: 10.1109/PROC.1983.12684.

[5] N. J. Nilsson *et al.*, "Shakey the robot", 1984.

[6] Y.-S. Ha and S. Yuta, "Trajectory tracking control for navigation of the inverse pendulum type self-contained mobile robot", *Robotics and Autonomous Systems*, vol. 17, no. 1, pp. 65–80, 1996, ISSN: 0921-8890. DOI: https://doi.org/10.1016/0921-8890(95)00062-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0921889095000623.

[7] N. M. A. Ghani, F. Naim, and T. P. Yon, "Two wheels balancing robot with line following capability", *World Academy of Science, Engineering and Technology*, vol. 55, no. 7, pp. 634–638, 2011.

[8] V. Coelho, S. Liew, K. Stol, and G. Liu, "Development of a mobile two-wheel balancing platform for autonomous applications", in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 575–580. DOI: 10.1109/MMVIP.2008.4749594.

[9] J. Dupuis and M. Parizeau, "Evolving a vision-based line-following robot controller", in *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, 2006, pp. 75–75. DOI: 10.1109/CRV.2006.32.

[10] M. Gomes, L. Bássora, O. Morandin, and K. Vivaldini, "Pid control applied on a line-follower agv using a rgb camera", in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 194–198. DOI: 10.1109/ITSC.2016.7795553.

[11] E. H. Binugroho, D. Pratama, A. Z. R. Syahputra, and D. Pramadihanto, "Control for balancing line follower robot using discrete cascaded pid algorithm on adroit v1 education robot", in *2015 International Electronics Symposium (IES)*, 2015, pp. 245–250. DOI: 10.1109/ELECSYM.2015.7380849.

[12] M. Engin and D. Engin, "Path planning of line follower robot", in *2012 5th European DSP Education and Research Conference (EDERC)*, 2012, pp. 1–5. DOI: 10.1109/EDERC.2012.6532213.

[13] G. Antonelli, S. Chiaverini, and G. Fusco, "A fuzzy-logic-based approach for mobile robot path tracking", *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 2, pp. 211–221, 2007. DOI: 10.1109/TFUZZ.2006.879998.

[14] N. Z. Azlan, F. Zainudin, H. M. Yusuf, S. F. Toha, S. Z. S. Yusoff, and N. H. Osman, "Fuzzy logic controlled miniature lego robot for undergraduate training system", in *2007 2nd IEEE Conference on Industrial Electronics and Applications*, 2007, pp. 2184–2188. DOI: 10.1109/ICIEA.2007.4318797.

[15] D. Ibrahim and T. Alshanableh, "An undergraduate fuzzy logic control lab using a line following robot", *Computer Applications in Engineering Education*, vol. 19, no. 4, pp. 639–646, 2011. DOI: https://doi.org/10.1002/cae.20347. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.20347. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.20347.

[16] H.-J. Chen and Y.-H. Kao, "Design of fuzzy control applied to the path following of lego-nxt system", in *2012 International conference on Fuzzy Theory and Its Applications (iFUZZY2012)*, 2012, pp. 263–267. DOI: 10.1109/iFUZZY.2012.6409713.

[17] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility", *Fuzzy sets and systems*, vol. 1, no. 1, pp. 3–28, 1978.

[18] S. Saadatmand, S. Azizi, M. Kavousi, and D. Wunsch, "Autonomous control of a line follower robot using a q-learning controller", in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, pp. 0556–0561. DOI: 10.1109/CCWC47524.2020.9031160.

[19] S. Mishra and P. Bande, "Maze solving algorithms for micro mouse", in *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, 2008, pp. 86–93. DOI: 10.1109/SITIS.2008.104.

[20] A. M. Sadik, M. A. Dhali, H. M. Farid, T. U. Rashid, and A. Syeed, "A comprehensive and comparative study of maze-solving techniques by implementing graph theory", in *2010 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 1, 2010, pp. 52–56. DOI: 10.1109/AICI.2010.18.

[21] nxt-python library, *Nxt.motor.basemotor*, (accessed February 13, 2023), 2021. [Online]. Available: https://ni.srht.site/nxt-python/latest/api/motor.html#nxt.motor.BaseMotor.

# A. State machine diagrams

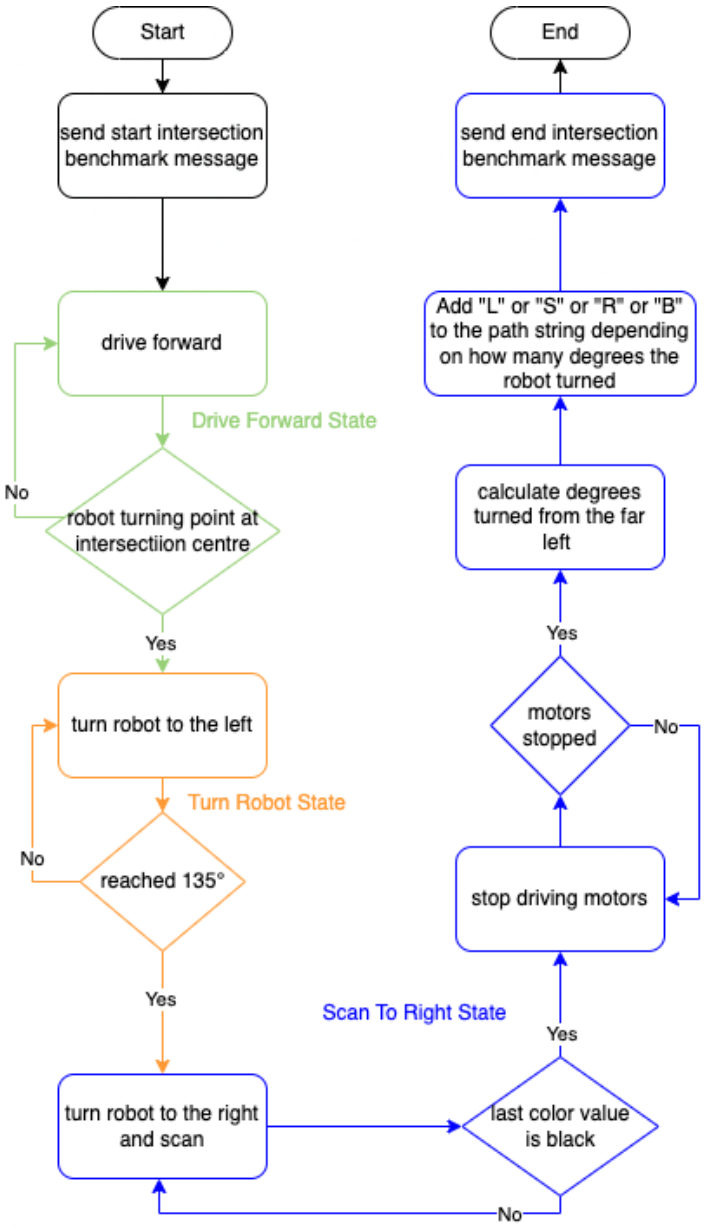



Figure 12: Fixed-sensor robot intersection scan state machine


Figure 13: Turning-sensor robot intersection scan state machine