2023

EEE3099S Milestone 4

THE REPORT

LLOYD ROSS, CAIDE LANDER, IMRAAN HARTLEY

Introduction

In recent years, self-driving vehicles have become a symbol of mechatronics engineering, with notable advancements from companies like Tesla and Lucid. Our design project is centered around the development of an autonomous robot equipped with advanced algorithms designed for a 'treasure hunt' mission. This robot utilizes line tracking to navigate a relatively simple maze and incorporates ultrasonic sensors to detect object distances.
For our milestone 2, the team simulated the system in MATLAB, drawing insights from sources like MATLAB's official YouTube channel. This report discusses the transition from simulation to practical implementation, detailing the electrical design and algorithms used to make this robot successfully complete its mission. The primary components include the Arduino Nano 33 IoT microcontroller, level shifters, encoder sensors, ultrasonic sensors, motors, and an H-bridge, all interconnected on a Vero Board.
The robot's primary objectives are to navigate the maze, detect objects, follow lines with a lit LED, and identify and return to the closest object upon completion.

System level Design

The central concept behind this project's design is the creation of a maze-solving robot. The initial blueprint encompasses the development of a light-colour sensor and communication circuitry that empowers the robot to discern a red light as a signal to halt and a green light to initiate forward movement. Essential component values were meticulously determined, along with the identification of additional components. A compact Vero board layout was designed to maximize space efficiency by creating strategic breaks in the board.
The subsequent pivotal phase of the design process involved formulating a comprehensive line-following algorithm, encompassing all requisite paths for line detection and tracking. Rotational capabilities were integrated by modifying the input rotation angle during line sensing. In the event of a required rotation, the robot would sense the direction necessary for turning, execute the rotation, and then reengage line sensing. Furthermore, a crucial objective was to ensure the robot's movement over 1 meter. This was accomplished by implementing a loop mechanism to monitor the robot's position. If it was not detected that the robot had traversed a meter, it would continue moving straight until the prescribed distance was achieved. Encoder sensors were put in front of the robot which calculated the distance travelled using the ticks per rotation of each wheel.

To master control over the robot's motion and line tracking, a set of four sensors was strategically positioned on the robot's front. Two sensors were placed in the middle for precise line sensing, while one was located at each end for detecting the other sides. For effective line following, the strategy involved detecting if any of the sensors deviated from the line, prompting a turn until they regained alignment with the line. When a 90-degree turn was required, the middle two sensors, in combination with one end sensor, were employed to detect the line, and the robot adjusted its orientation accordingly.

At T-junctions, all sensors were engaged to scan the surroundings, and the robot continued moving forward until a new line was detected. object detection using an ultrasonic sensor was put in place to detect objects as well as convert the waves into distance the object is from the robot and at what distance this is compared to the desired distance it needs to be before stopping in front of the object.

Localisation was also implemented using all the above sensors and showing the ability of the robot to recall and track back its previous steps by following its path. All of this was done on MATLAB and the algorithms were implemented within it.

Electrical Design

	Voltage Used (V)	Max Current (A)
Line Sensor (4x)	3.3V	<10mA
Voltage Regulator – Arduino	7.8V	1.2A
H-bridge	7.8V	2A
Ultrasonic Sensors	5V	15mA (peak); 20mA (max)
Encoders (2x)	5V	<20mA

Table 1 - Voltage and current of components

The Arduino Nano 33 IoT uses a MPM3610 DC-DC converter as its voltage regulator. This converter has a maximum input voltage of 21V and a maximum output current of 1.2A. However, it is important to note that the maximum output current is dependent on the input voltage and the ambient temperature. For example, if the input voltage is 21V and the ambient temperature is 25°C, the maximum output current is 1.2A. However, if the input voltage is 21V and the ambient temperature is 70°C, the maximum output current is reduced to 0.6A.
It is also important to note that the Arduino Nano 33 IoT does not have any heatsinking for the voltage regulator. This means that if you are drawing a lot of current from the voltage regulator, it may overheat. To avoid this, it is recommended to operate the voltage regulator below its maximum output current.

In terms of current calculations for the 3.3V line and 5V:

The maximum current draw on the 3.3V line is 40mA (4*Line Sensor = 4*10mA = 40mA).
The rest of the max current can be calculated by combining the following components:

- Arduino: 1.2A
- H-bridge: 2A
- 5V line (Ultrasonic Sensors - 20mA max and 2x Encoders - 20mA max each, totalling 40mA)

Therefore, the total current load is 1.2A + 2A + 40mA = 3.2A, with a maximum potential of 40mA on the 3.3V line, resulting in a total current of approximately 3.26A.

In actuality this is the maximum current and the actual current used by our system as found practically is: 0.26A Average with the very highest voltage recorded at 0.28A.

The measured voltage was 7.56V (Although this will fluctuate depending on the capacity of the batteries).
Therefore, the average power consumption is 0.26A*7.56V = 1.9656W = 1.97 Watts.

Max power consumption is 0.28A*7.56V = 2.1168W = 2.12 Watts.

It was found that some batteries encountered overcharging – due to using state flow along with time values derived from trial and error to achieve turning (90°, 180° etc) – it was found that the overcharged batteries caused overshoot. To avoid issues and maintain simplicity it was ensured that batteries used were charged to specification. This issue could be largely avoided by using encoders to manage turning.

	Overcharged batteries (8.3V)	Normal batteries (7.6V)
H-Bridge regulated out	5.06V	5.06V
Minimum voltage to motors (Low speed)	1.33V	1.20V
Maximum voltage to motors (High speed)	2.68V	2.36V

Table 2 - Overvoltage values

In terms of safety measures for the power supply design and schematics within the Arduino Nano-based system:

1. We eliminated the need for an additional voltage regulator since the Arduino Nano already features an integrated voltage regulator.

2. The H-bridge incorporates a built-in 5V voltage regulator, allowing for the supply of voltage exceeding 5V. It stabilizes the input to a constant 5V, which is essential for powering the ultrasonic sensors and encoders.

3. We incorporated level shifters to convert the 5V to 3.3V for interfacing with the Arduino Nano, as it can only accept 3.3V without risking damage. Furthermore, level shifters were employed to convert 3.3V to 5V for other system components.

4. Originally to ensure a stable voltage and safeguard against voltage spikes that could harm sensitive equipment like the Nano, a Zener Diode was going to be integrated into the design. However, due to the system being powered by batteries – under normal circumstances a Zener Diode would not be needed. Therefore, to slightly reduce circuit complexity and marginally increase efficiency – the Zener Diode was not included within the Vero board circuit.

Below is the fully assembled and soldered Veroboard circuit:

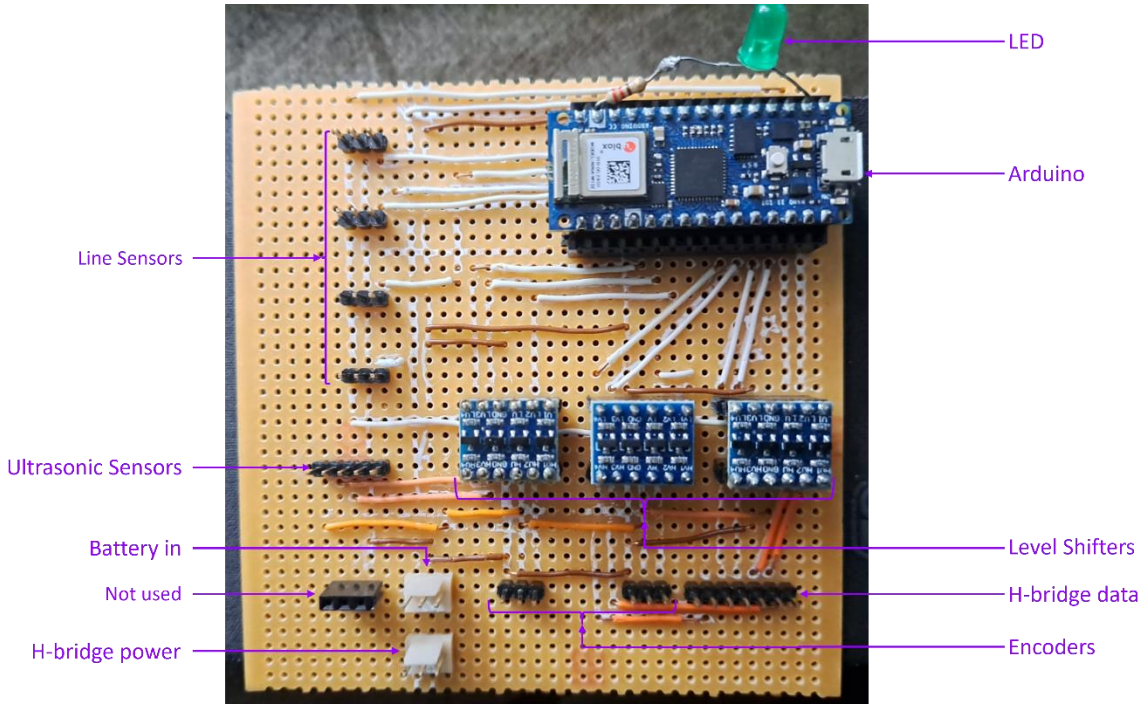


Figure 1: Vero Board fully assembled.

Legend:
Orange Wire - 5V
White Wire - 3.3V
Brown Wire – GND

Algorithms

Distance and angle control

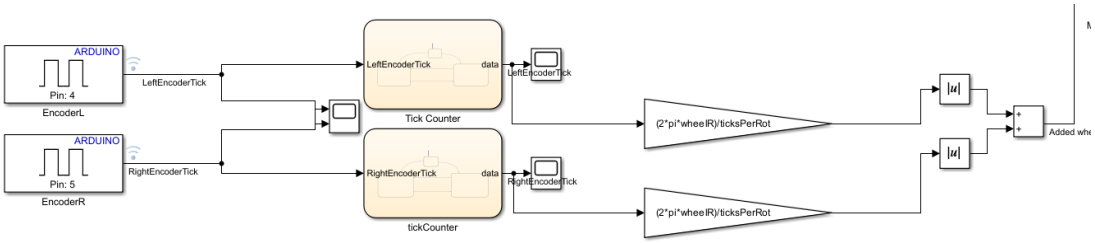


Figure 2: Encoder sensor distance calculation

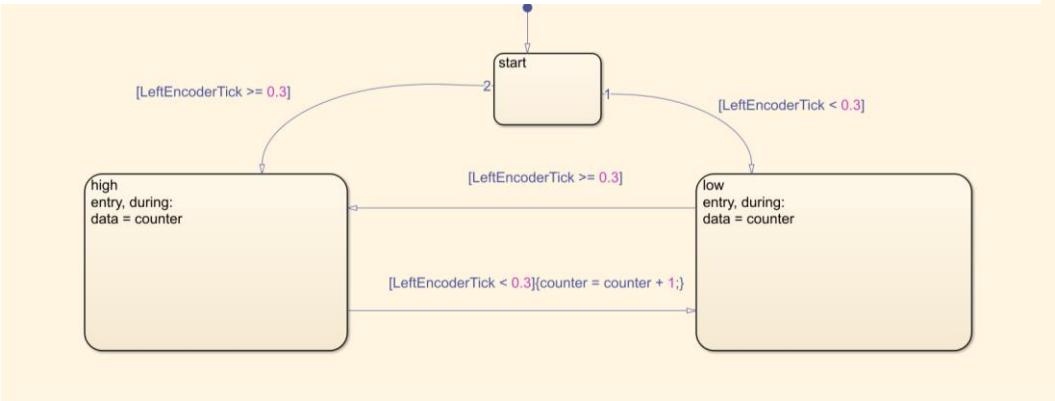


Figure 3: Falling edge tick counter stateflow

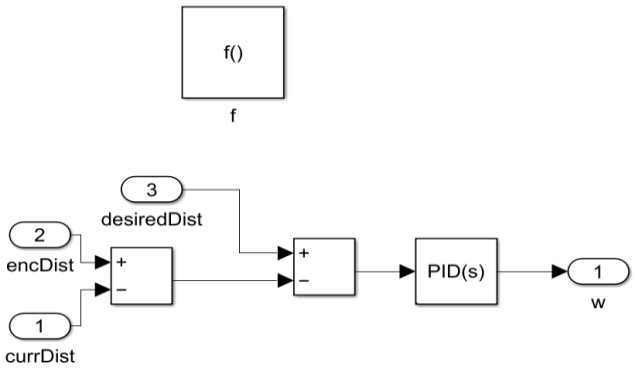


Figure 4: Distance controller

The distance and angle control algorithm's purpose are to find out the distance the robot moves and the angle. This is measured and recorded via encoder sensors. The number of rotations is used to figure out the distance travelled.

Calculations

$$\text{Number of rotations} = \frac{\text{Total encoder ticks}}{\text{ticks for one revolution}}$$

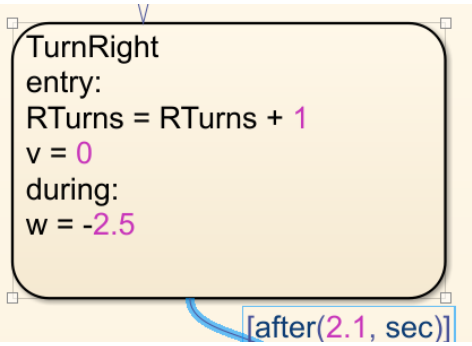
$$\text{Distance} = \frac{2\pi R}{\text{ticks for one revolution}}$$

Straight line distance control and angle

1. Encoder sensor output goes through falling edge tick counter and distance is calculated using the previous equation and input into the main stateflow algorithm.
2. Due to the distance increasing constantly throughout, current distance is subtracted from encoder distance to get the distance from the start of the specific control so it works with any given setpoint at any given time.
3. Subtracted distance to get distance travelled is subtracted from specified setpoint to get an error value
4. Error value is put through PID controller and output is input as w into the wrwl conversion block
5. Each wrwl is put through a lookup table and output to the motors as a PWM signal
6. wrwl is checked and the respective pins are set high and low in order to set the wheels direction
7. As the robot moves forward the error value will decrease and will be 0 at desired distance ensuring exact distance control of the robot

For purpose of the final demo and practical implementation, angle control using PID controllers was not used as it proved to be very buggy and non-functional on the hardware. So it was decided that state flow motion control will be used to rotate.

To turn 90 degrees angular velocity is set to 2.5 rad/s for 2.1 seconds
To turn 180 degrees angular velocity is set to 3.5 rad/s for 2.5 seconds



Line following

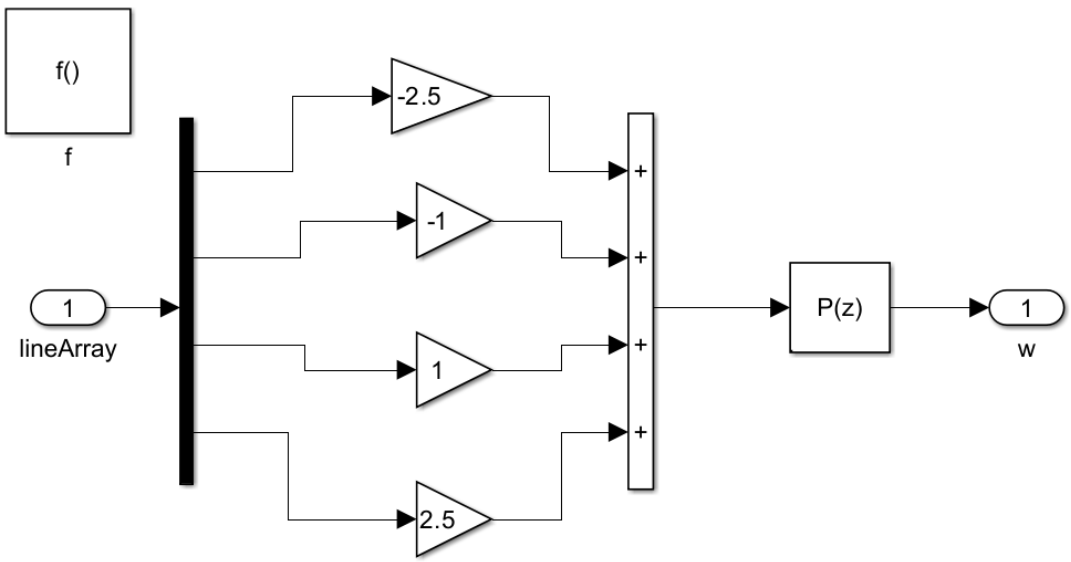


Figure 5: 2 Line sensor gain array

Line following algorithm's purpose is to keep the robot on a straight path. Four line sensors are used. Five sensors were an option but for is better for even parity of the robot. The sensors emit light and detect while lines on a black surface. The algorithm checks if the value from the sensor is more or less than the threshold value. It then makes the robot turn left if the sensor has a greater value than threshold value or if it is less, it will turn right.
This will fluctuate but mostly prevent error on the line following path.

Stepped description: Line following algorithm

1. Use a block with the parameters being:
2. Four-line sensors in this block. The output of this line sensor outputs an array of the sensors of the line values distributed.
3. Use a demux block to show the individual sensors. Two sensors on the outskirts and two on the inside.
4. The outer sensors have 2.5 and -2.5 gains while the inner has gains of 1 and -1. The four sensors are then passed through a summation block and through a P block which has a constant gain of 1.2.
5. In normal circumstances innermost and outermost sensors would produce a value that would cancel each other out – producing a w = 0.
6. When the line deviates from the centre the error increases (as by default if the sensors have the same value, then their outputs would cancel each other out), this error is then multiplied by the gain of the specific sensor.
7. The output of the P block is the angular velocity (w) input of the wlwr block.
8. During the robot movement the velocity is set to 0.12m/s. This velocity is input into the wlwr block.
9. The wlwr block is connected to the LUT for the left and right motor.
10. The output of each respective LUT is put through an absolute block before being put into an Arduino PWM block which outputs a signal that first gets level shifted from 3.3V to 5V and is then sent to the H-bridge which then powers the motors.

Pseudocode: Line following algorithm

```
// Step 1: Initialize parameters

float lineSensors[4];

float gains[4] = {2.5, 1, -1, -2.5};

float error, angularVelocity, velocity;

float LUT_left, LUT_right;

float leftMotorSignal, rightMotorSignal;


// Step 2: Read data from line sensors

readLineSensorData(lineSensors);
```

```
// Step 3: Demux block to access individual sensors
float outerSensors[2], innerSensors[2];
outerSensors[0] = lineSensors[0];
outerSensors[1] = lineSensors[1];
innerSensors[0] = lineSensors[2];
innerSensors[1] = lineSensors[3];

// Step 4: Process sensor data
float sumOfOuterSensors = outerSensors[0] + outerSensors[1];
float sumOfInnerSensors = innerSensors[0] + innerSensors[1];
float P_gain = 1.2;

error = (sumOfInnerSensors - sumOfOuterSensors) * P_gain;

// Step 5: Calculate angular velocity
if (error == 0) {
   angularVelocity = 0;
} else {
   angularVelocity = error;
}

// Step 6: Set robot velocity
velocity = 0.12;

// Step 7: Connect to wlwr block
LUT_left = angularVelocity;
LUT_right = angularVelocity;

// Step 8: Calculate motor signals
leftMotorSignal = abs(LUT_left);
rightMotorSignal = abs(LUT_right);
```

```
// Step 9: Power the motors

setMotorSpeed(leftMotorSignal, rightMotorSignal);
```

Maze solving, localisation and object detection.

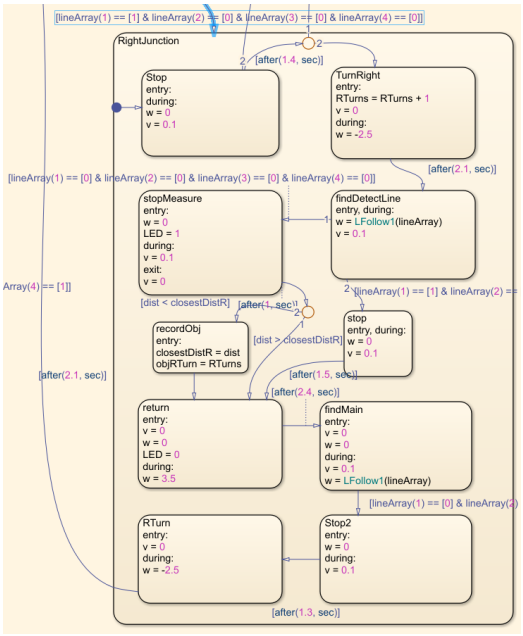


Figure 6: Right Junction Simulink

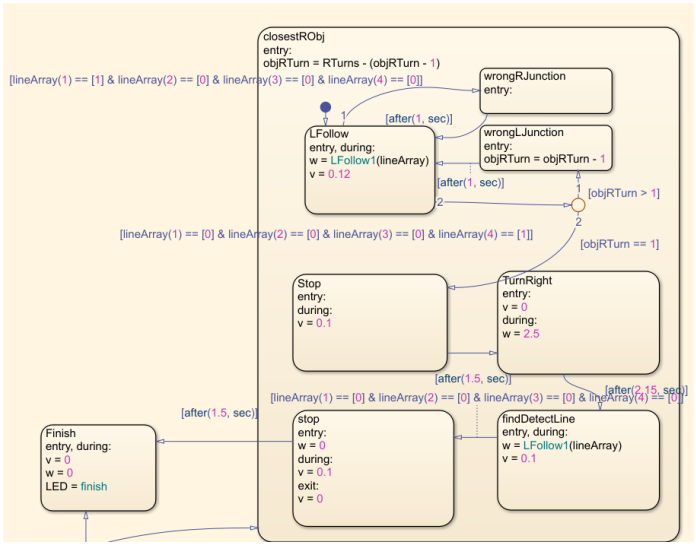


Figure 73: Closest Object Simulink

All the previous algorithms will be used and test the robot's ability to recall its position and previous steps it took. This algorithm is the most important for the robot to follow the previous algorithms. The line following algorithm is implemented so that the robot stays on the straight path. Object detection is then implemented using the ultrasonic sensor. This sensor measures the distance from the detect line to the object. This value is the measured distance. This is all repeated until the robot finishes the maze.

Maze solving algorithm

1. State flow Simulink chart block is used having variables as such:
    - Dist
    - closestDistL
    - closestDistR
    - ObjLturn
    - ObjRturn
    - RTurns
    - LTurns

2. Robot starts maze at the beginning on the main path and in main path state.

3. Robot line follows at a v = 0.12 until a junction is detected by the 3 left or right sensors going low
4. As a junction is detected the robot slightly moves forward before turning and checks if the main road continues after the junction to confirm branching off from the main road.
5. Robot turns into branch, increments RTurns or LTurns respectively to count the number of branches and enters the respective right or left junction state and line follows till a detect line is found(lineArray = [0, 0, 0, 0]) or line ends(lineArray = [1, 1, 1, 1])
6. Detect line is found the robot stops, LED turns on and saves the output of the ultrasonic sensor to closestDist(L/R) if object Dist is closer than current closest value(Dist < closestDist(L/R)) and saves the current (R/L)Turns to the respective Obj(L/R)turn to remember which turn the closest object is on.
7. After object dist is measured or no detect line is found the robot turns 180 degrees and line follows back until the main road is sensed(lineArray = [0, 0, 0, 0]) and turns back onto main road.
8. Repeats for every junction until the end of maze is reached. Robot turns 180 degrees, line follows and checks if the closest left or right calculates which turn the closest object is on for the inverse as we are travelling back on the maze(objRTurn = RTurns - (objRTurn - 1)).
9. Robot goes along main road skipping the wrong junctions by entering an empty state for a second till the robot passes the junction as to not effect line following and obj(R/L)Turn is decremented till it = 1 and robot has found the correct junction with the closest object
10. Robot turns onto junction and stops at detect line to finish.

Conclusion

In the realm of mechatronics and robotics, the development of our autonomous maze-solving robot, equipped with semi-advanced algorithms, represents a testament to the ingenuity of modern engineering. Our mission was to create a robot capable of navigating a maze using line tracking and detecting objects through ultrasonic sensors. In this report, we have detailed the journey of bringing this concept to life.

Our project started with simulating the system in MATLAB, gaining insights from various sources, including MATLAB's YouTube channel. We then transitioned from the simulation to practical implementation. This involved designing the electrical components and integrating them into the system. The key components included the Arduino Nano 33 IoT microcontroller, level shifters, encoder sensors, ultrasonic sensors, motors, and an H-bridge, all interconnected on a Vero Board. The robot was designed to complete the maze, detect objects, follow lines with an LED indicator, and identify and return to the nearest object.

The system-level design involved creating a robust line-following algorithm that allowed the robot to maintain its path. Four line sensors were strategically placed on the robot, helping it to stay on course and make necessary adjustments when the innermost sensors deviated from the line and the outermost sensors crossed the line. Additionally, the robot was equipped with the ability to detect T-junctions aswell as detect objects using ultrasonic sensors, and it could even retrace its steps using localization.

The electrical design considered voltage and current requirements for the various components, ensuring stable operation. We relied on the integrated voltage regulator of the Arduino Nano 33 IoT and used level shifters to interface between 3.3V and 5V components. To maintain circuit simplicity, we decided not to include a Zener Diode due to the use of batteries.

Despite the initial plans to use encoder sensors for precise control, they proved challenging to work with during practical implementation, and we opted for a state flow motion control approach instead.

The maze-solving, localization, and object detection algorithms brought the robot's capabilities full circle. These algorithms combined the line-following capabilities with object detection using ultrasonic sensors, creating a maze-solving robot that could remember its path and return to the nearest object.

During our practical demonstration, the robot successfully navigated the maze, adeptly detected all objects, followed lines precisely, and obediently responded to detect lines with a lit LED. Moreover, it demonstrated the ability to identify and return to the nearest object upon completing the maze. However, an unexpected issue marred the final demonstration, thwarting the robot's return to the closest object. This underscores the crucial significance of robust design principles in real-world applications.

In conclusion, our project showcases the potential of mechatronics and robotics to create semi-intelligent, autonomous systems.