




---

# Autonomous Line Tracking and Object Detection Robot

Sayuri Moodley, Talon Sewnath and Revashan Soobiah



---

# Table of Contents

- 01** Introduction
- 02** System Level Design
- 03** Electrical Design
- 04** Motion Control Design, Implementation, and Results
- 05** Line Sensing Design, Implementation, and Results
- 06** Treasure Maze Solving Algorithm Design, Implementation, and Results
- 07** Conclusion

# 01 Introduction

The autonomous line-following robot project is a multi-faceted opportunity to engage with sub-system design, computer simulation, and practical testing. These aspects were tested through the development of the robot's ability to navigate a maze using its various sensors, input code, and a microcontroller. More specifically, the robot was intended to follow a black line which guides the robot, by means of its line sensors, and detect objects from a designated point. Once the maze is completed, the robot will navigate back to the object with the shortest distance from the detect line.

Physically, the robot consisted of several separate components which required assembly. Of all the given components, certain components were eliminated based on their redundant functionality to lower power consumption overall. The Arduino Nano 33 IoT board contained 256kB of memory which allowed for the detailed and sizable simulations to be uploaded from MATLAB. The on-board processor is the ARM Cortex-M0 32-bit SAM21 which was ideal as it has low power consumption. The ultrasonic sensor HC-SR04 used ultrasound reflection as a means of measuring the distance of the object directly in front of it. More specifically, it measures the travelling time of the ultrasonic waves to the object and uses the time multiplied by the speed of the waves to calculate the distance. 2 bi-directional logic level converters were included because certain components require a lower input voltage, i.e., 3.3V instead of 5V. To drive the 2 brush motors, a dual H bridge was connected to allow for the direction and speed of the wheels to be controlled. These motors were placed inside the housing of the 2WD mobile platform and connected to its wheels.

The first stage of this project engaged with how the robot would theoretically solve the maze, by dividing the robot's capabilities into subsystems and expanding on each aspect. Comprehensive UML diagrams were developed for subsystem interactions and the first draft of the schematic was created. In the second stage, the first computer algorithms were produced through simulation. The first draft of functions such as turning at set angles, line following, and object detection were generated. Although these original functions were not used in the final testing, this exercise aided immensely in developing the correct thinking patterns. The last stage of this project was the physical testing of the robot. Many unforeseen issues arose which could not have been predicted through simulation such as the hardware failure or accounting for friction. This stage included soldering the necessary headers to Veroboard along with placing microchips and connecting all the separate hardware to the board. The final algorithms included detailed state flow maps developed on Simulink which were uploaded to the Arduino.

# 02 System Level Design

## Sensor Selection

For the robot to perform all the necessary tasks, 2 sensors were utilised. The first being the line sensor and the second being the ultrasonic sensor. 4 line sensors were chosen from a possible 5 as 4 was deemed the most efficient for the task at hand. The sensors were placed such that the outer two sensors were equally spaced from the central two sensors which were next to each other. The ultrasonic sensor was used to measure and transfer the distance recorded. Encoder sensors were provided however were not actively used for maze solving.

### Justification

The topology of the sensors allowed for smooth line following. The central two sensors being next to each other allowed for early path correction and the outer sensors were used to identify turns. The different sensors detecting also determined the angular velocity required for correction. The line sensors were also convenient to use as they have an input voltage range between 3.3V and 5V which is easily accessible in the designed circuit. The ultrasonic sensor was used since it provided easy distance detection. It also functions with an input of 3.3V to 5V and outputs a voltage safe for the Arduino to process. The encoders were not used as the maze didn't require any distance control and was purely controlled by line sensors.

## Mechanical Assembly

Headers were mounted on the Veroboard for the ICs to plug into. Connector headers were also soldered onto the board next to the Arduino, logic level shifters and sensor signal outputs.

The Veroboard was mounted firmly on top of the robot to ensure sturdy positioning using two-way tape. The H bridge was also mounted next to the Veroboard to allow for wires to easily connect to the Veroboard's headers. The ultrasonic sensor was placed in between the shell of the robot and held firmly using a mould of Prestik.

### Justification

Headers were mounted to prevent the solder from burning the components. The connector headers were used to allow for testing, troubleshooting and extra connection channels. The Veroboard, H bridge and ultrasonic sensors were mounted to prevent connections becoming loose and shorts occurring if components touch.

## Electrical Connection and Pin Allocation

Headers were soldered on instead of wired connections between different components on the Veroboard to reduce the effects of shorts and potentially blowing the Arduino however ground was connected through soldered wires. 3.3V was taken out from the Arduino to reduce the need for a voltage regulator. Digital pins were used for the ultrasonic sensor and H-bridge which included PWM pins. 2 digital pins were used by the ultrasonic sensor. 6 digital pins were used by the H bridge, 2 of which were PWM pins, and 4 analogue pins were used by the line sensors.

### Justification

Headers were used instead of soldered wires as this provides freedom to change pin connections and test for electrical failures. Ground was connected through soldered wires to reduce the number of wire headers for ease of debugging whereas power supply was connected through soldered wires and wire headers to be able to supply power to both the Arduino and H-bridge. Due to the ultrasonic and H-bridge taking up majority of the digital pins, the line sensors were forced to be analogue pins. The ultrasonic sensor

# 02 System Level Design

requires two PWM pins for the trigger and echo signals to be able to send out signals and read distance. The H bridge requires PWM pins to enable A and B and the 4 control pins require digital pins.

## Control Algorithm

MATLAB was the chosen software for simulation, specifically Simulink. A state flow was used as it provided a high-level interface for implementing logic. The central node was used for line following containing go straight, adjust right and adjust left actions. There are 3 transitions from the main node, the first being checkLeft, the second being checkRight and the last being the EndOfMaze. The check actions check whether it is a turn or a 90° bend. From each checkTurn there are 2 transitions, one to the main branch after completing the bend and the second to the turn procedure. The turn procedure causes the robot to move along the turn, stop, sense, turn around and re-enter the maze with the transition going back to the central state. The last transition from the central state is to find the treasure.

### Justification

MATLAB was used as it provided lots of tools for developing the control algorithm of the robot. Simulink provided pre-existing blocks that simulate the hardware components. State flow provided an easy interface to develop the algorithm for maze solving.

## Power Supply

A single line was used for ground, battery power and 5V power with drill holes between them to prevent shorts. A connector from the Arduino GND pin connected to the GND line. Only one switch was used for both the Arduino and the H bridge. The Arduino provides the 3V3 output to the line sensors and ultrasonic sensor. The H bridge provides 5V output. The logic level shifters and encoders were not used.

### Justification

A single ground section was convenient as wires were taken and soldered to wherever it was needed to complete the ground connection between components. The battery was connected to a line for the same purpose as ground as connections were taken from power to the Arduino and H-bridge. One switch was used due to soldering and connection problems. It was found that the connection between the switches would break, which could potentially damage the robot infrastructure and wiring. The 3V3 output from the Arduino was used instead of a regulator because the Arduino is more reliable in the sense that it does not fluctuate in the output. It was found that the ultrasonic sensor was able to operate on the 3V3 voltage which allowed for the converters to be disregarded. This made it easier for debugging as less wire headers were needed, and the flow of wires was better to see.

# 03 Electrical Design

## Arduino Nano 33 IoT

Symbol	Description	Min	Typ	Max	Unit
$V_{IN\ max}$	Maximum input voltage from $V_{in}$ pad	-0.3	-	21	V
$V_{USB\ max}$	Maximum input voltage from USB	-0.3	-	21	V
$P_{max}$	Maximum power consumption	-	-	TBC	mW

## Motor Driver Dual H-Bridge

Symbol	Description	Min	Typ	Max	Unit
$V_{drive}$	Voltage needed to drive the motors	2	-	5	V
$I_{drive}$	Current needed for driving the motors	0.5	-	2	A
$P_{max}$	Maximum power consumption ( $P = VI$ )	2.5	-	70	W

## Encoder

Symbol	Description	Min	Typ	Max	Unit
$V_{drive}$	Voltage needed to drive the motors	3	-	5	V
$I_{drive}$	Current needed for driving the motors	5	-	20	A
$P_{max}$	Maximum power consumption ( $P = VI$ )	15	-	100	W

## Line Following Sensor

Symbol	Description	Min	Typ	Max	Unit
$V_{drive}$	Voltage needed for sensor	3.3	-	5	V
$I_{drive}$	Maximum current needed for sensor	200	-	880	mA
$P_{max}$	Maximum power consumption ( $P = VI$ )	660	-	4400	mW

# 03 Electrical Design

## Logic Level Converter

Symbol	Description	Min	Typ	Max	Unit
$V_{IN\ max}$	Voltage needed for reference	-0.3	-	21	V
$I_{drive}$	Current needed for converter	-0.3	-	21	V
$P_{max}$	Maximum power consumption ( $P=VI$ )	-	-	TBC	mW

## Battery

Symbol	Description	Min	Typ	Max	Unit
$V_{drive}$	Voltage	-	3.7	5	V
$I_{drive}$	Current	-	2000	300	mA
$P_{max}$	Maximum power consumption ( $P = VI$ )	-	7400	1500	mW

## Ultrasonic Sensor

Symbol	Description	Min	Typ	Max	Unit
$V_{in}$	Voltage for sensor	3.3	4	5	V
$I_{in}$	Current needed for sensor	250	270	300	mA
$P_{max}$	Maximum power consumption ( $P = VI$ )	825	1080	1500	mW

# 03 Electrical Design

## Final Total Power Subsystem

The Encoder and Logic level converter was not used in the final design for the power submodule, hence why it was not included in the final calculations.

Symbol	Voltage Supply (V)	Min Current (mA)	Typ Current (mA)	Max Current (mA)	Min Power (mW)	Max Power (mW)
Arduino Nano 33 IoT	7.6	7	20	35	53.2	266
Motor Driver Dual H-Bridge	7.6	500	1500	2000	3800	15200
Line Sensors (x4)	3.3	2	6	10	$4(V)(I) = 26.4$	$4(V)(I) = 80$
Ultrasonic sensor	3.3	250	270	300	825	990
Total					4704.6	16536

## Microcontroller Resource Allocation

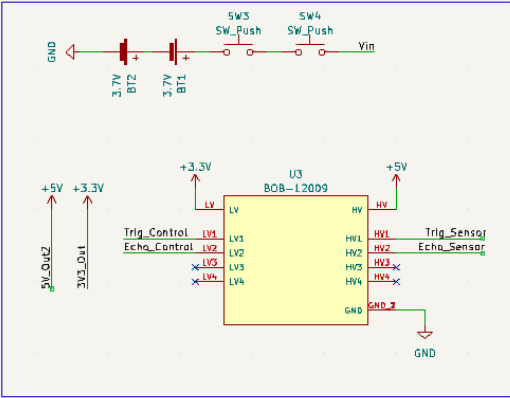
- Voltage of 7.6V was fed into Vin of the Arduino directly from the batteries.
- 3.3V was taken out of the Arduino in order to power the Line sensors with the required voltage.
- Enable A and Enable B of the H-Bridge required PWM pins of the motors in order to send a pulse to the motors, therefore digital pins 9 and 2 were used, respectively, to incorporate this.
- In 4 and In 3 are used for Enable B therefore, for ease of debugging, digital pins 3 and 4 were used respectively next to Enable B at pin 2.
- Similarly In 1 and In 2 are used for Enable A therefore digital pins 8 and 7 were used respectively next to Enable A at pin 9.
- The Ultrasonic sensor needs to use two digital pins with PWM to be able to send and retrieve signals and calculate distance. Therefore digital pins 5 and 6 were used for the trigger and echo , respectively, signals to and from the sensor.
- In order to show that the ultrasonic sensor was able to sense an object, the use of the internal LED on the Arduino board was used. A signal was sent to D13 that flashed the orange LED on the Arduino.
- Digital pins 2 and 4 were used for the encoders however since the final design did not use the encoders, the pins were freed for the use of the H-bridge.
- Due to the majority of digital pins being used for other components. It was decided that Line sensors were easiest to convert from analogue to digital. Therefore analogue pins 1,2,6 and 7 were used. A space between analogue pin were left for ease of debugging to know which two sensors were left most opposed to the right most sensors.



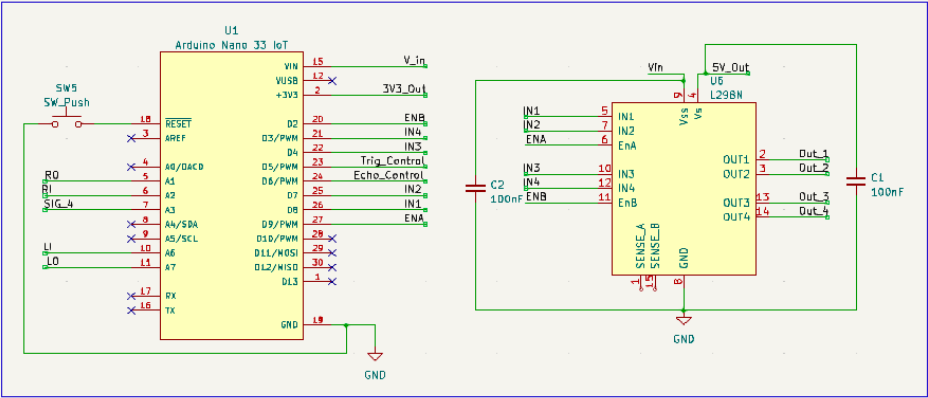
# 03 Electrical Design

## Final PCB Schematic

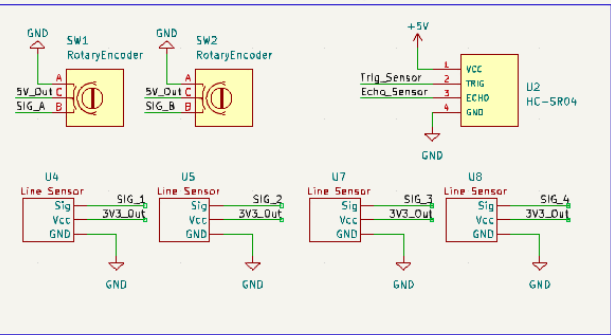
The power module consists of the battery power and the bidirectional converter. The battery power is connected to the micro-controller via two switches. The bidirectional converters allow dual voltage capability.



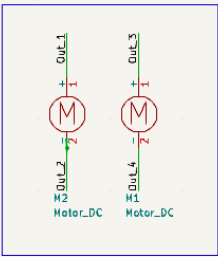
The Arduino Nano is responsible for sending out trigger signals to sensors and receiving digital converted data from sensors. The Nano is also used to regulate voltage into 3.3V and 5V from the 7.4V. The trigger signal to operate the ultra sonic sensor.



The 4 line sensors are connected to the microcontroller. They receive power from the Arduino and send back data for processing. The rotary encoders are also connected to the Arduino but receive 5V power. The ultra sonic sensor uses echo as an output and trigger as an input from the MCU.



The motor drives receive power from the H bridge and the direction of each wheel is controlled by the H bridge.



Author 3:Talon Sewnath  
Author 2:Revashan Soobiah  
Author 1:Sayuri Moodley  
**University of Cape Town**

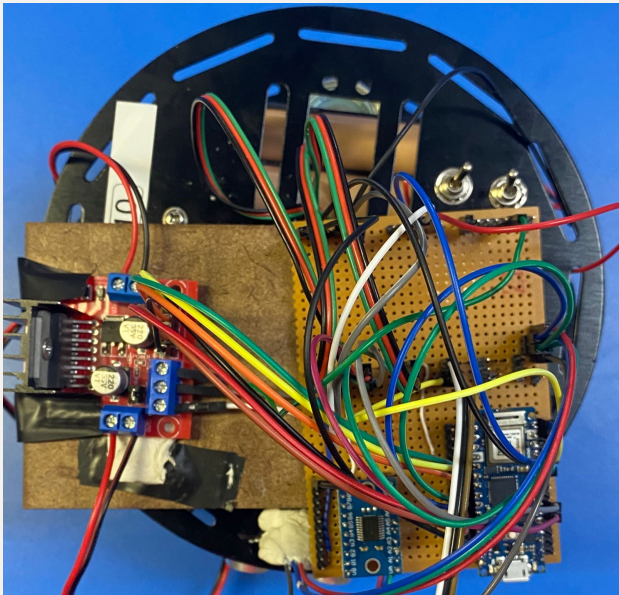
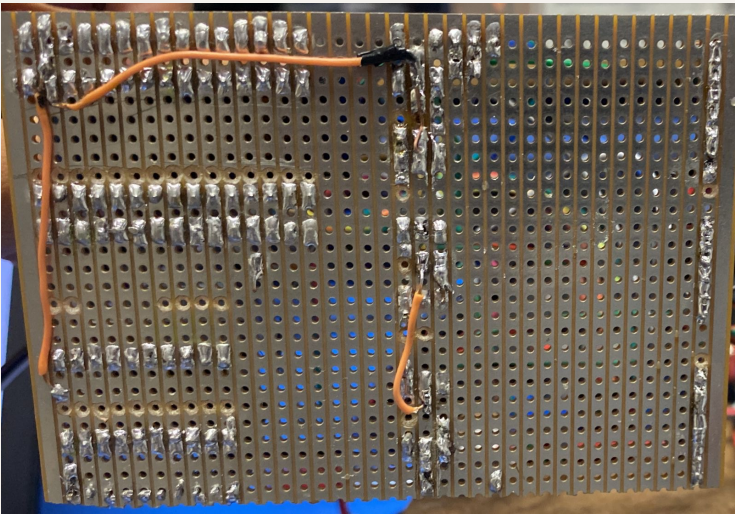
Sheet: /  
File: EEE3099\_Paper\_Design.kicad\_sch

**Title: Paper\_Design**

Size: A4 Date: 2023-08-18

KiCad E.D.A. kicad 7.0.1-0

Rev: v0.01  
Id: 1/1



# 04 Motion Control Design, Implementation, and Results

## H Bridge Pseudocode

Receive  $w_L$

Receive  $w_R$

if  $w_L \geq 0$

    Arduino output  $\rightarrow$  IN1

If  $w_L < 0$

    Arduino output  $\rightarrow$  IN2

Convert  $w_L$  &  $w_R$  to magnitude

Input magnitudes to LUT L&R

Receive PWM outputs from LUT

Output PWMs to H bridge ENA and ENB

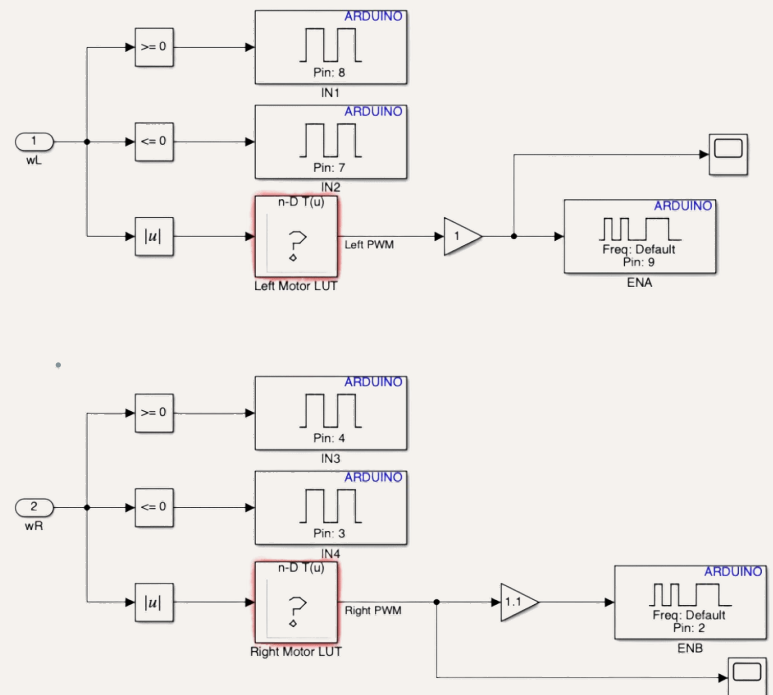


Figure 1a: H bridge Simulink model

## Encoder Pseudocode

Receive signal from L&R Encoders

Receive  $w_L$  &  $w_R$

If  $w_L/w_R > 0$

    scaleFactor = 1

Else

    scaleFactor = -1

Detect current change

Scale current change with scaleFactor

Integrate scaled Ticks

Output L/R ticks

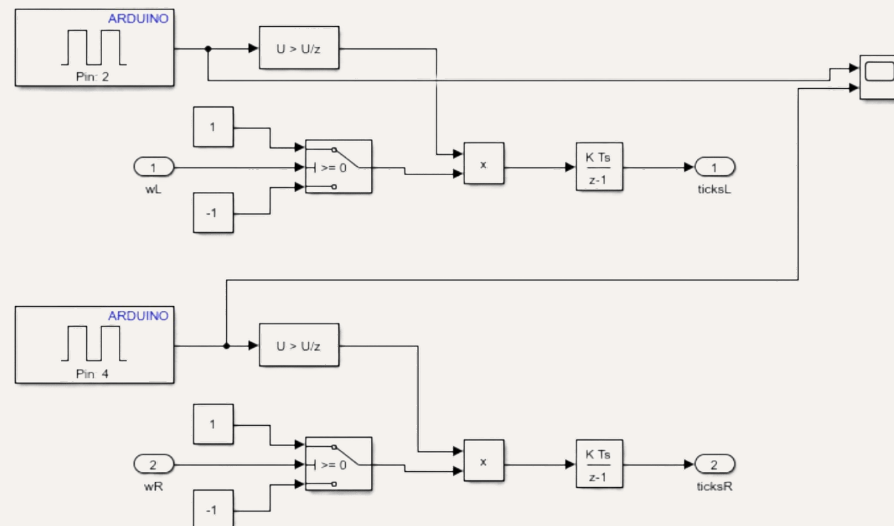


Figure 1b: Encoder Simulink model

# 04 Motion Control Design, Implementation, and Results

$$w_L = (v - w * axleLength/2)/wheelRadius$$
$$w_R = (v + w * axleLength/2)/wheelRadius$$

or in matrix notation,

$$[w_L; w_R] = (1/wheelRadius) * [1 \ -axleLength/2; 1 \ axleLength/2] * [v; w]$$

### Motion Control Pseudocode

Receive v, w  
Convert v, w to wl and wr  
Scale wl and wr by 1/wheelRadius  
Output wl, wr

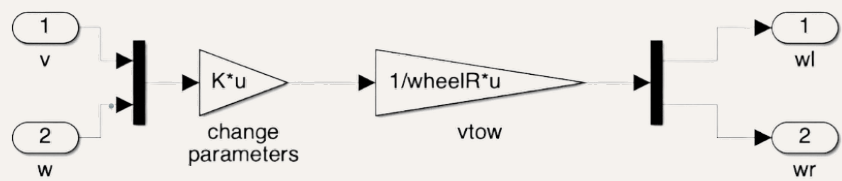


Figure 1c: Motion control Simulink model

Result	Justification
The practical motor block was saturating at voltages greater than 7.5V. Even though the input to the motor was a max of 5, the motors were saturating when testing.	Lookup tables were used to convert the digital signal to a voltage sent to the motor. The practical motor blocks were replaced by PWM blocks to prevent saturation.
The H bridge outputs two unequal enable voltages to the motors resulting in one wheel being faster than the other. Typically, the left motor received an enable voltage that was 1.1 times greater than the right motor.	This was not a connection error as the H bridge and motors were switched out for new components. A 1.1 gain block was used to scale the right motor up to compensate for the practical error. The 1.1 scalar sends equal voltages to the H bridge.
The encoder ticks per rotation was found to be 12. The calculated value for ticks per 1m travelled was 94 ticks. In testing, 94 ticks on the scope resulted in a 10-metre distance travelled.	The number of ticks was scaled down by 10 to allow the robot to travel the required distance.
Linear velocities greater than 0.3 resulted in a noticeable overshoot and angular velocities greater than 5 resulted in the robot not being able to catch lines.	The linear velocity used was restricted to under 0.25 for pure straight motion. The angular velocity was restricted to 5 rad/s for the left motor and 8 rad/s for the right motor for pure rotation. When executing path correction, a small linear velocity assisted the angular velocity to allow smooth correction.
The motors had to be re-attached as every time the robot was placed down, the speed immediately dropped.	The motors were touching since the connection was loose and they were cancelling each other out.

# 05 Line Sensing Design, Implementation, and Results

## Line Sensor Conversion Pseudocode

A2D():

Threshold = 2000

If LineSensor input > Threshold

Output = 1

Else

Output = 0

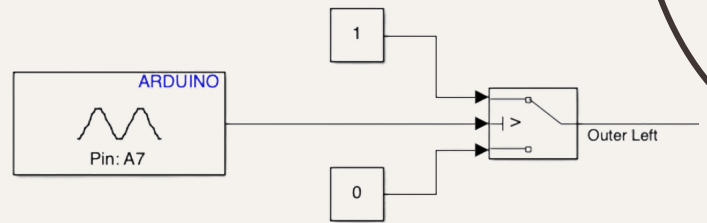


Figure 2a: Line sensor Simulink model

## Path Correction Pseudocode

LineFollow():

Read sensor data

While (1)

If L2 = white & L4 != black

v = moderate linear speed

w = moderate negative angular speed  
(small correction to the right)

If L3 = white & L1 != black

v = moderate linear speed

w = moderate positive angular speed  
(small correction to the left)

If L4 = black

v = small linear speed

w = big negative angular speed  
(big correction to the right)

If L1 = black

v = small linear speed

w = big positive angular speed  
(big correction to the left)

End

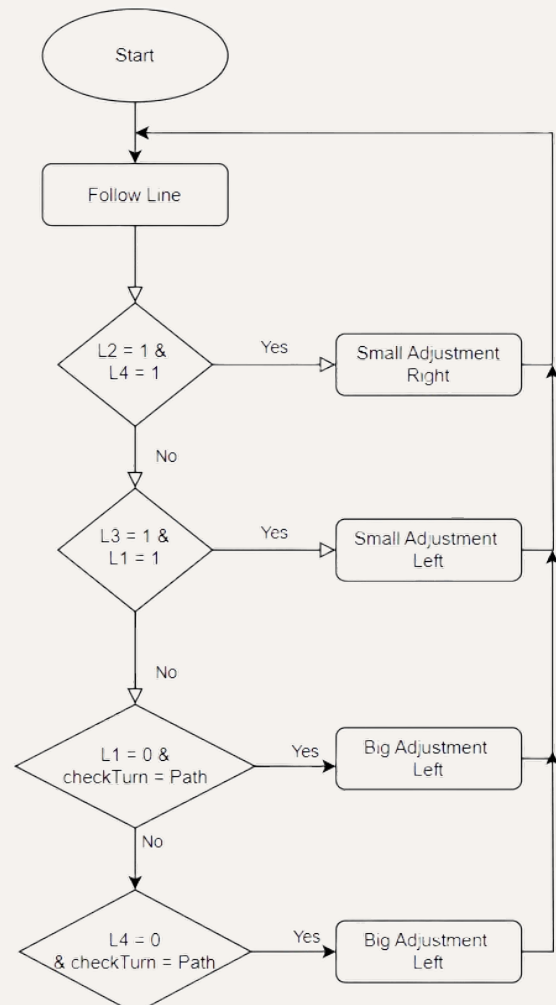


Figure 2b: Path correction flowchart

# 05 Line Sensing Design, Implementation, and Results

## Mode Recognition Pseudocode

Given robot on main path:

If L1&L2&L3 = black

mode = Left turn or 270° Path change

Else If L2&L3&L4 = black

mode = Right turn or 90° Path change

Else if L1&L2&L3&L4 = black

mode = End

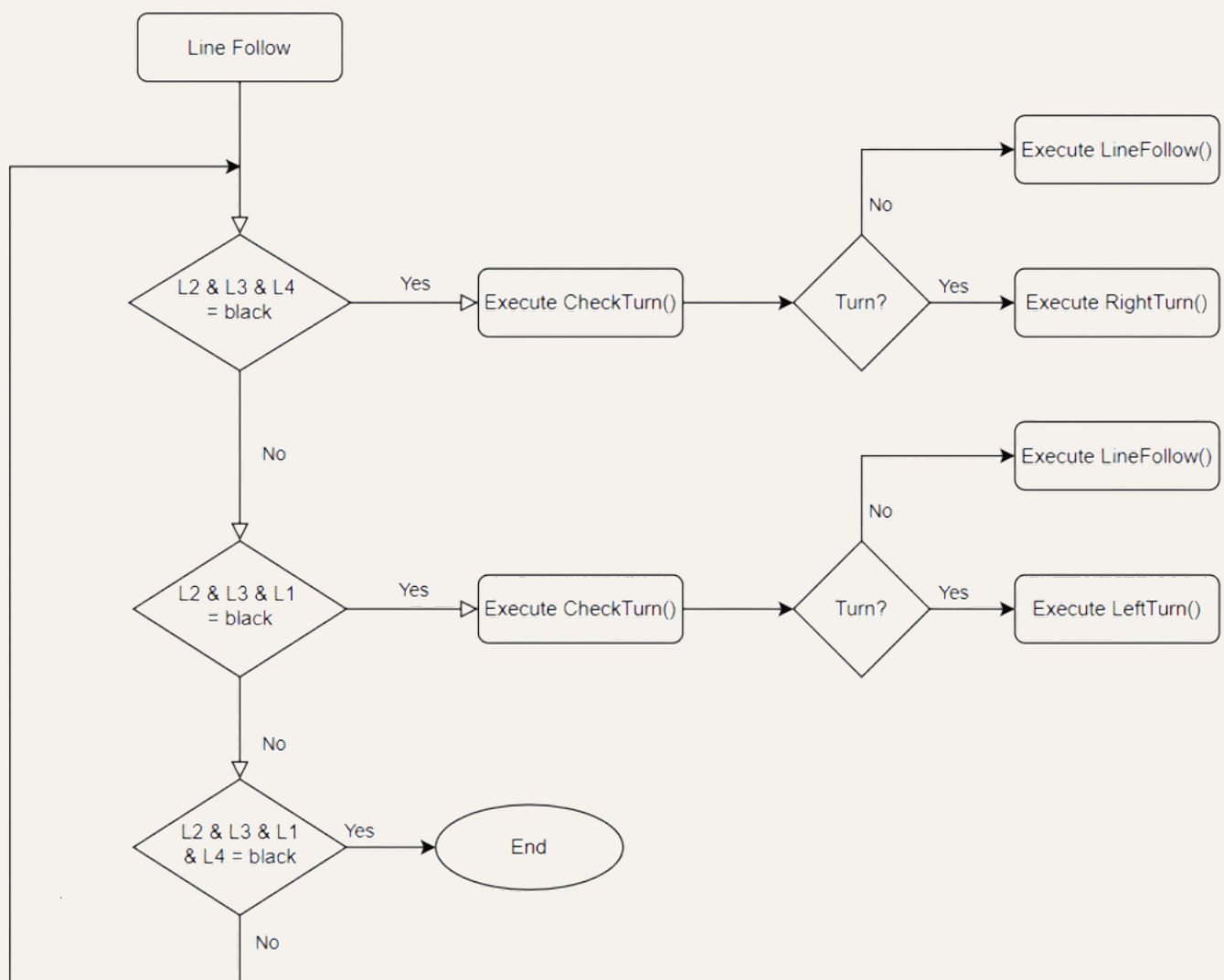


Figure 2c: Mode recognition flowchart

# 05 Line Sensing Design, Implementation, and Results

Result	Justification
The threshold for reading white was set to greater than 2000 and black for less than 2000. Using this logic, the analogue pins were converted to a digital equivalent.	The line sensors were tested on connected io to a display and the threshold was determined to be greater than 2000 for white and less than 2000 for black. The reason for converting to digital was for ease of use in the state flow.
The inner sensors were used for path correction and the outer sensors were used for turns.	The outer sensors resulted in the robot constantly deviating from the path whereas the central sensors provided quick correction.
There was an issue with turns and 90° paths as the robot could not differentiate between the two. In both cases, the same three sensors read black.	A test was done where the robot moves forward. If it reads white then it's a 90 degrees path and if it reads black, it's a turn.
The robot had to halt at terminal nodes for more than 0.2 seconds.	The ultrasonic sensor could not register if the robot did not stop. The halt time had to be greater than twice the sampling time.
The use of the ultrasonic sensor resulted in the after conditions in the state flow to stop working.	The sampling parameters were changed and all the components sampling time had to be synchronised to better the performance of the after conditions.



# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## Check Turn Right Pseudocode

CheckTurnR():

If L2&L3&L4 = black

Move forward

If L2&L3 = white

Return line following

Else

Return Right turn

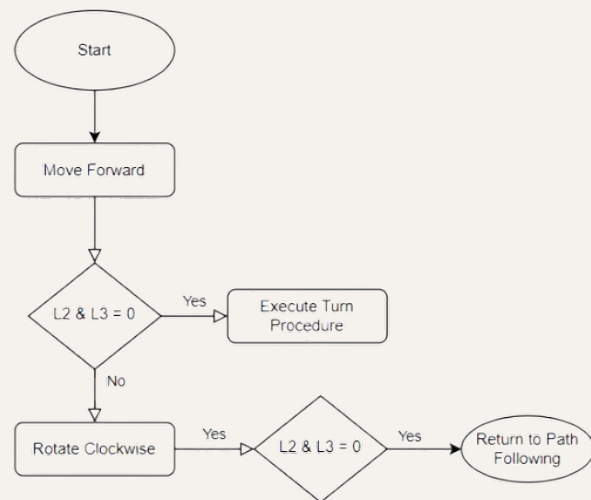


Figure 3a: Check turn right flowchart

## Check Turn Left Pseudocode

CheckTurnL():

If L2&L3&L1 = black

Move forward

If L2&L3 = white

Return line following

Else

Return Left turn

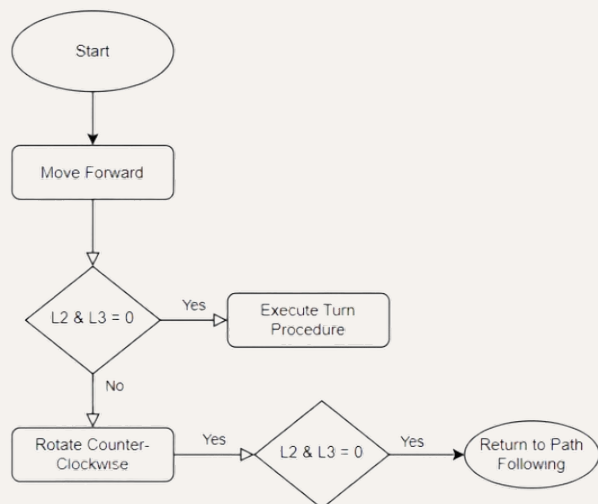


Figure 3b: Check turn left flowchart

## Ultrasonic Pseudocode

Set sampling time to 0.1s

If sensedVal > 0.5

Dist = 0

Else

Dist = sensedVal

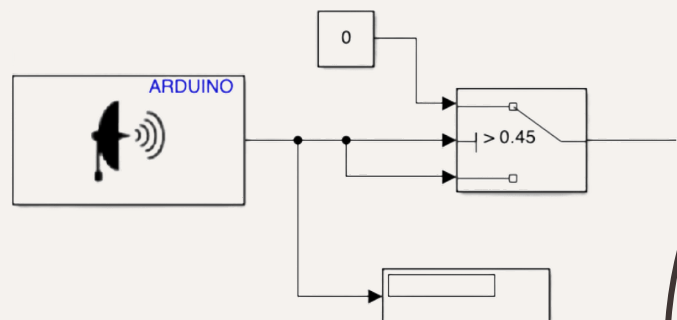


Figure 3c: Ultrasonic Simulink model

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## Right Turn Execution Pseudocode

Given L2&L3&L4 = black

rightTurn():

Increase turn count

Move forward as adjustment

Rotate clockwise until L2&L3 = black & L1&L4 = white

Perform line following until L1&L2&L3&L4 = black

Stop for half a second

Check minimum ultrasonic distance

Set minimum branch

Reverse for half a second

Rotate until L2&L3 read black

(condition that must happen after 0.5 seconds to ensure turning happens)

Perform line following until L1&L2&L3&L4 = black

Move forward as adjustment

Rotate clockwise until L2&L3 = black

(condition that must happen after 0.5 seconds to ensure turning happens)

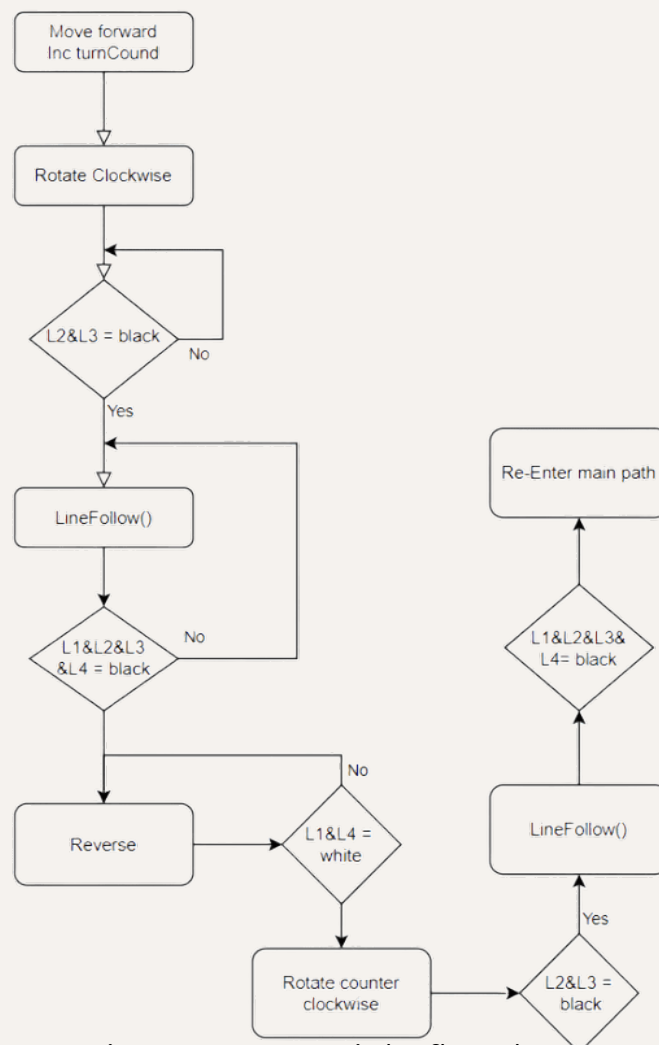


Figure 3c: Turn right flowchart



# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## Left Turn Execution Pseudocode

Given L1&L2&L3 = black

rightTurn():

Increase turn count

Move forward as adjustment

Rotate anticlockwise until L2&L3 = black & L1&L4 = white

Perform line following until L1&L2&L3&L4 = black

Stop for half a second

Check minimum ultrasonic distance

Set minimum branch

Reverse for half a second

Rotate until L2&L3 read black

(condition that must happen after 0.5 seconds to ensure turning happens)

Perform line following until L1&L2&L3&L4 = black

Move forward as adjustment

Rotate anticlockwise until L2&L3 = black

(condition that must happen after 0.5 seconds to ensure turning happens)

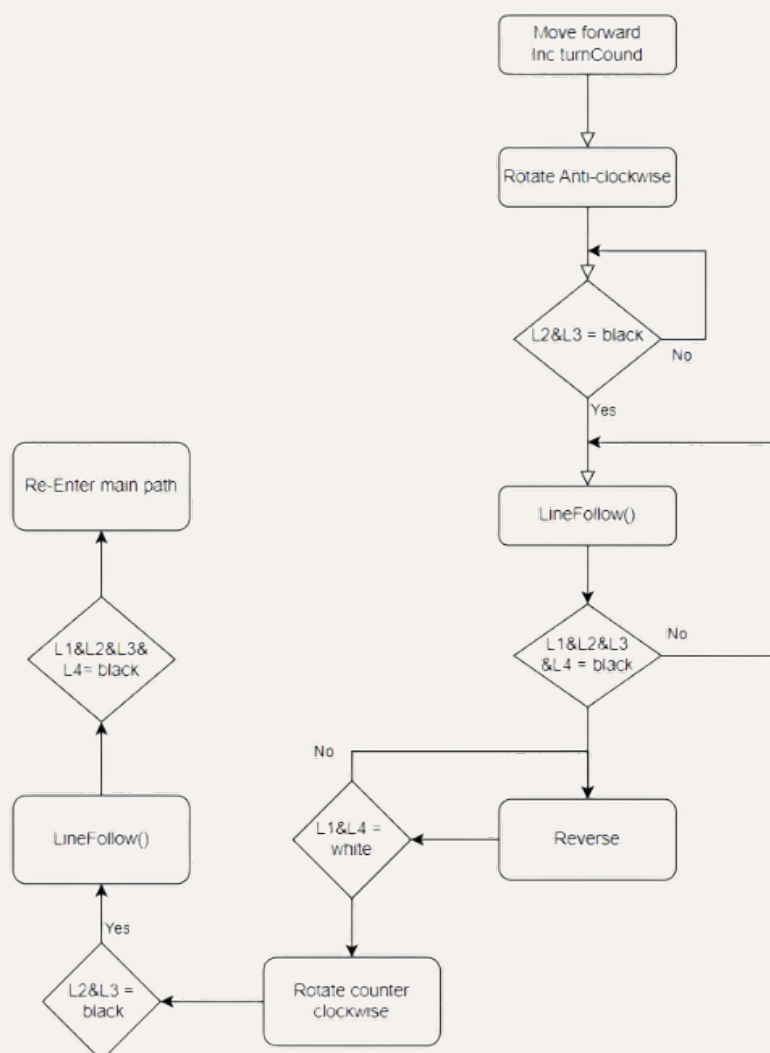


Figure 3d: Turn left flowchart

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## Maze Solver Pseudocode

While not at end

Linefollow()

if L4 & L3 & L2 & L1 = black

Stop

Wait to start treasure solving

if L4 = black & L1 != black

checkRightTurn()

If checkRightTurn = Turn

rightTurn()

return to main line following

Else

lineFollow()

if L1 = black & L4 != black

checkLeftTurn()

if checkLeftTurn = Turn

leftTurn()

return to main line following

Else

lineFollow()

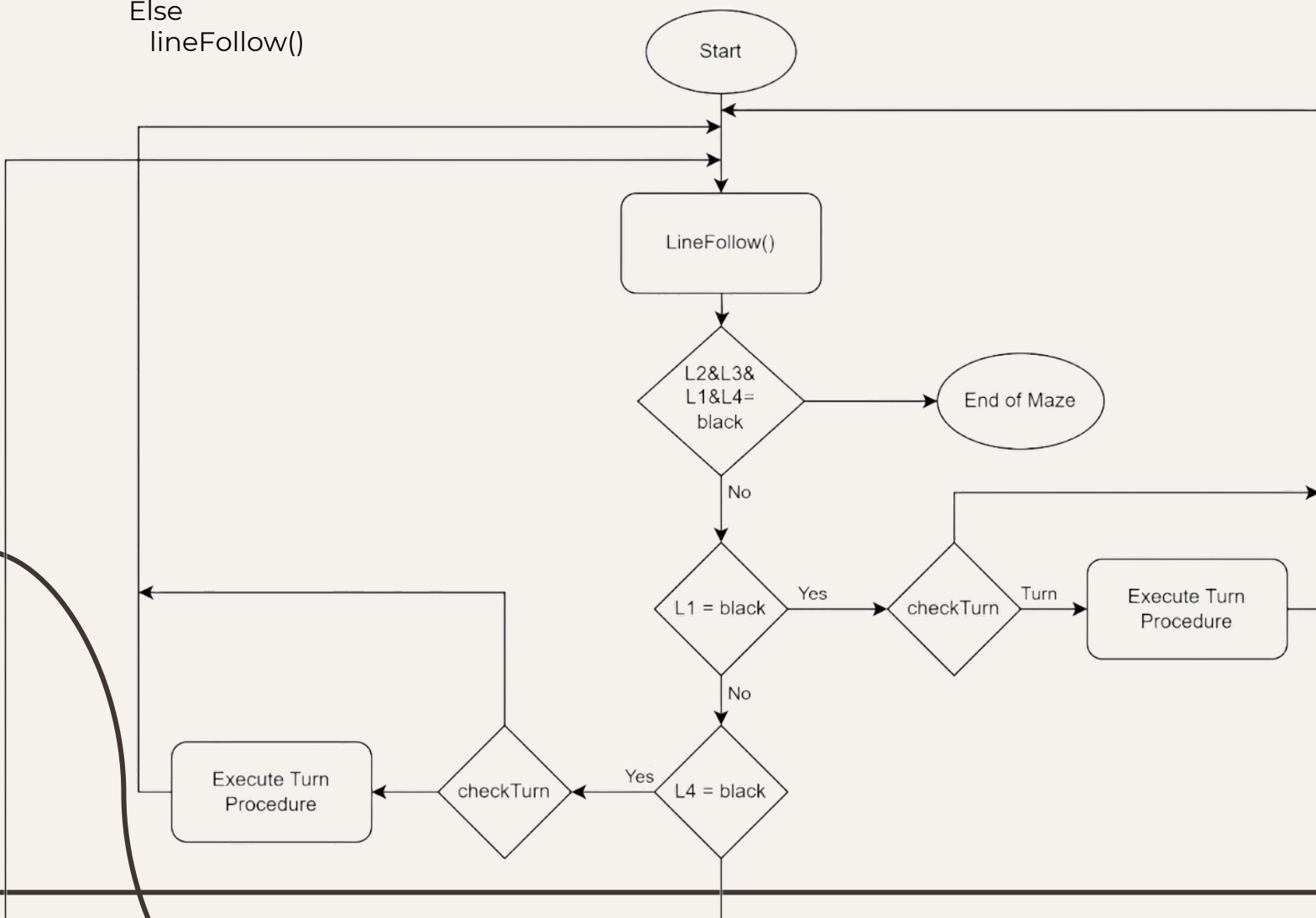


Figure 3e: Maze solver flowchart

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

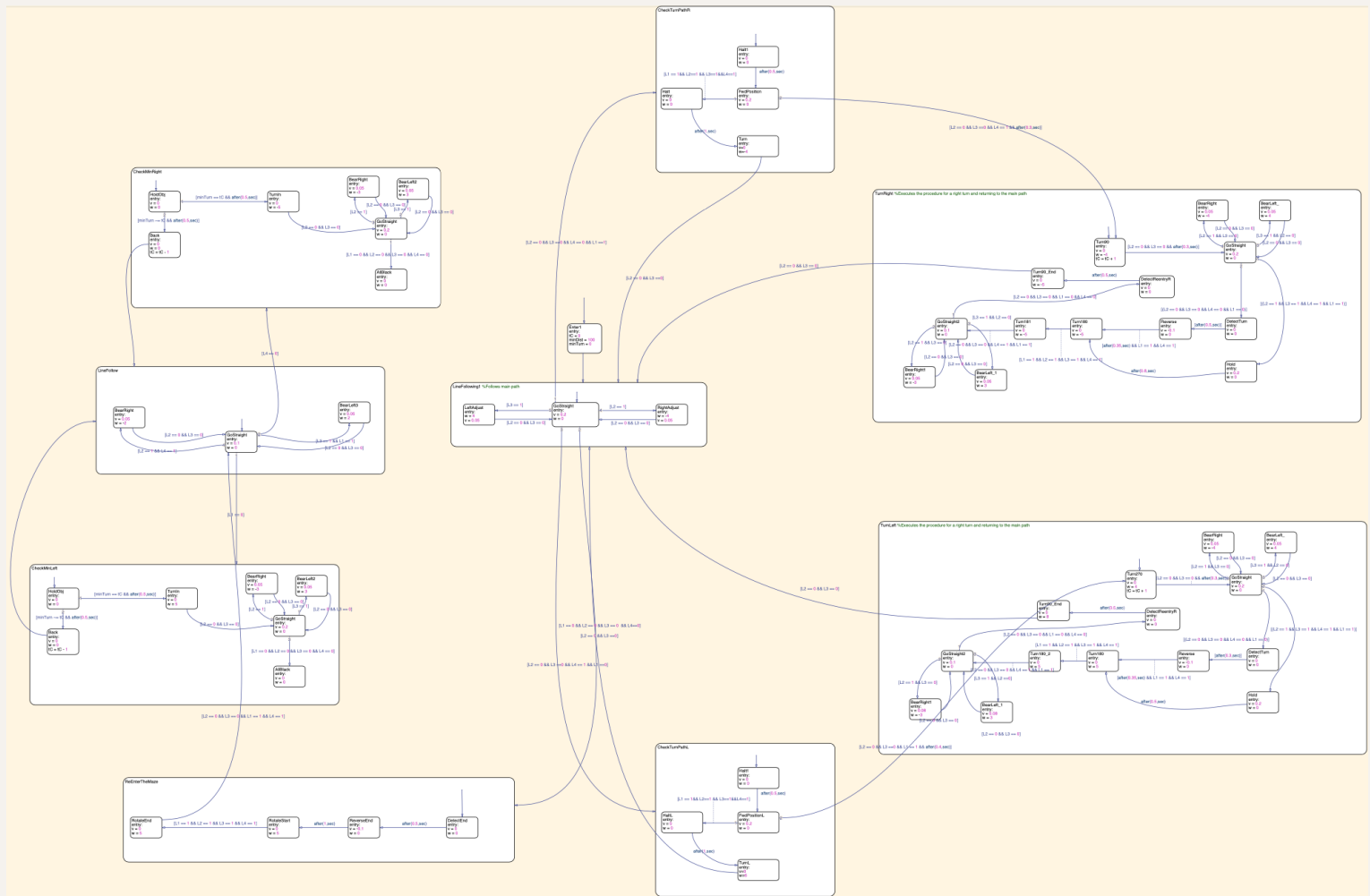


Figure 4a: Entire state flow

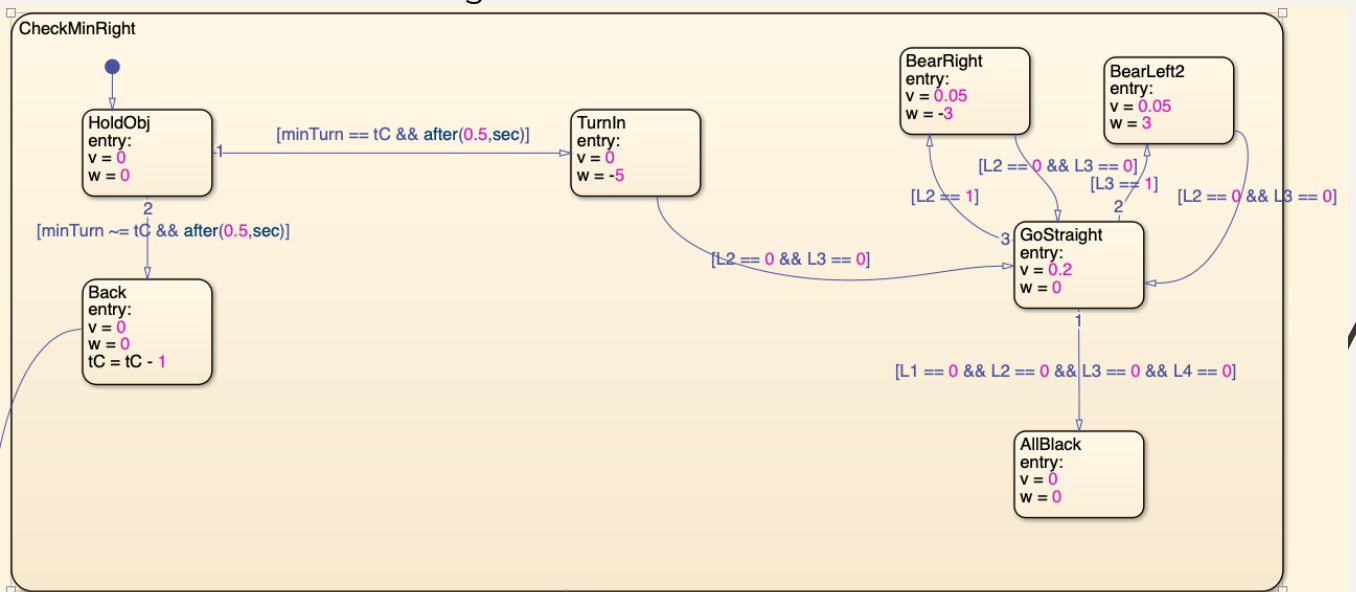


Figure 4b: CheckMinRight state flow

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

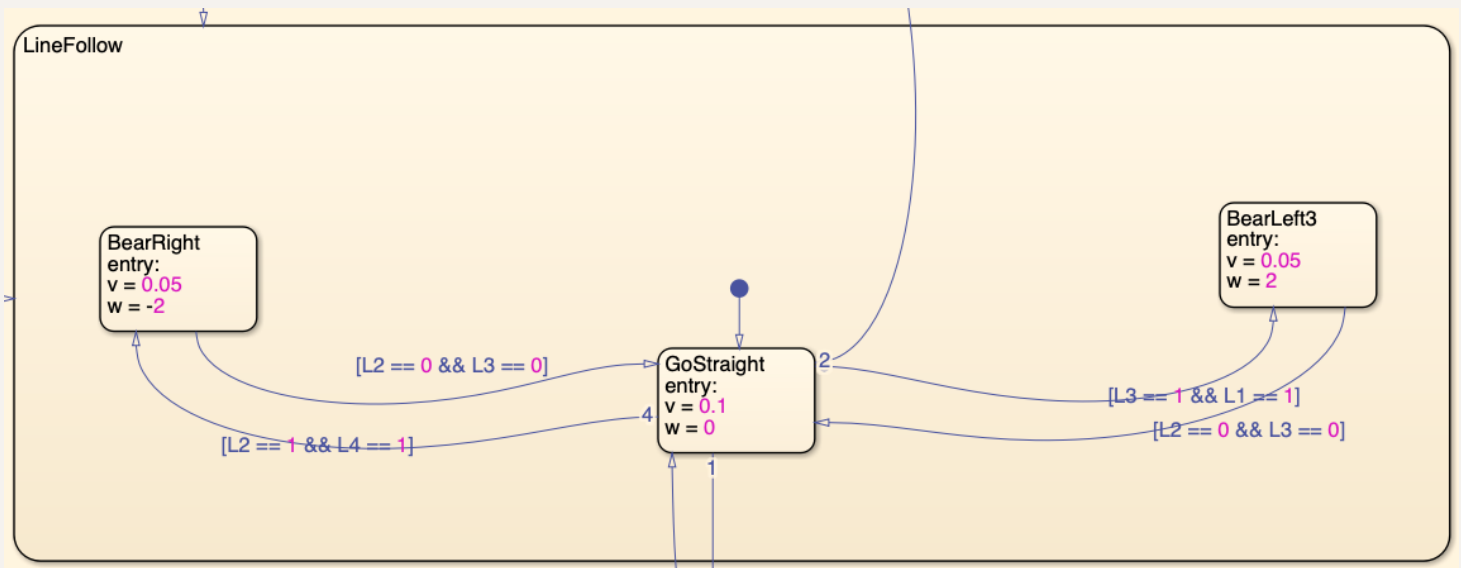


Figure 4c: LineFollow state flow

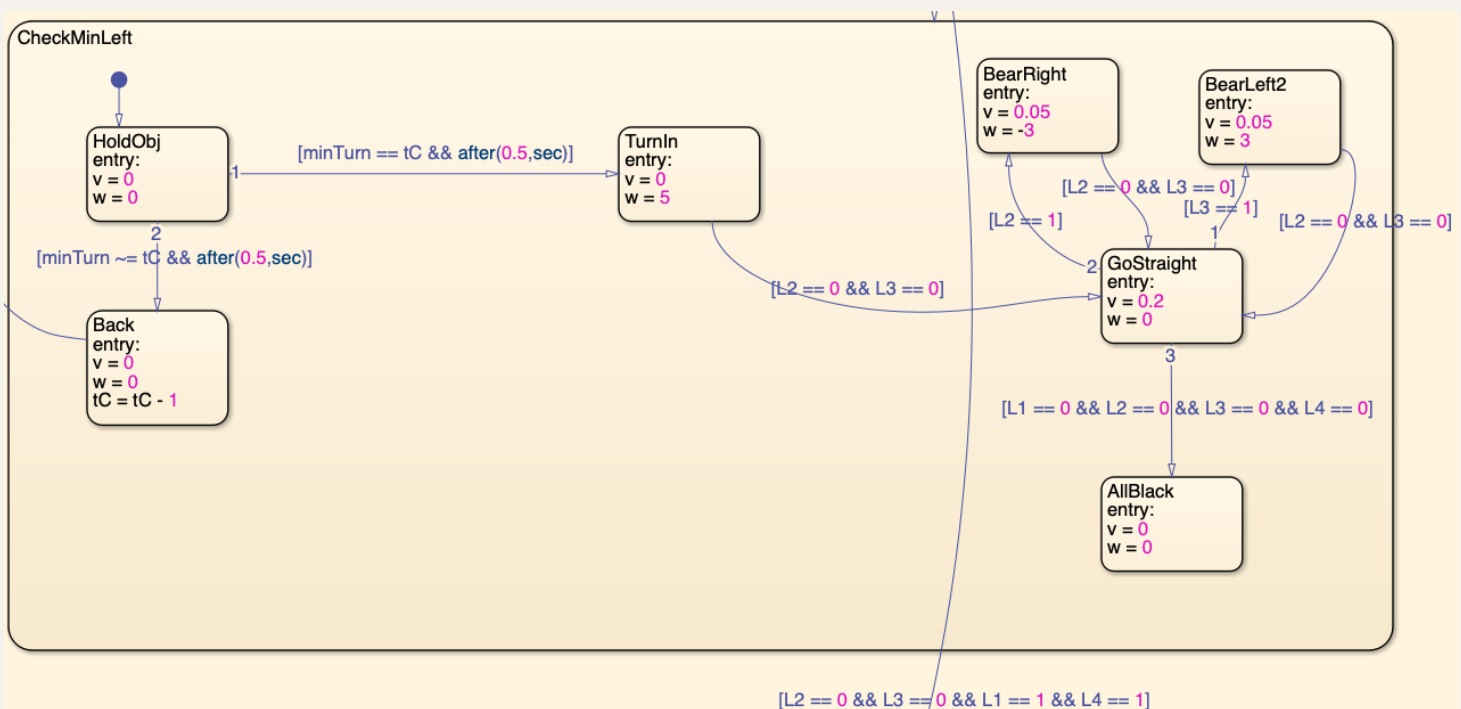


Figure 4d: CheckMinLeft state flow

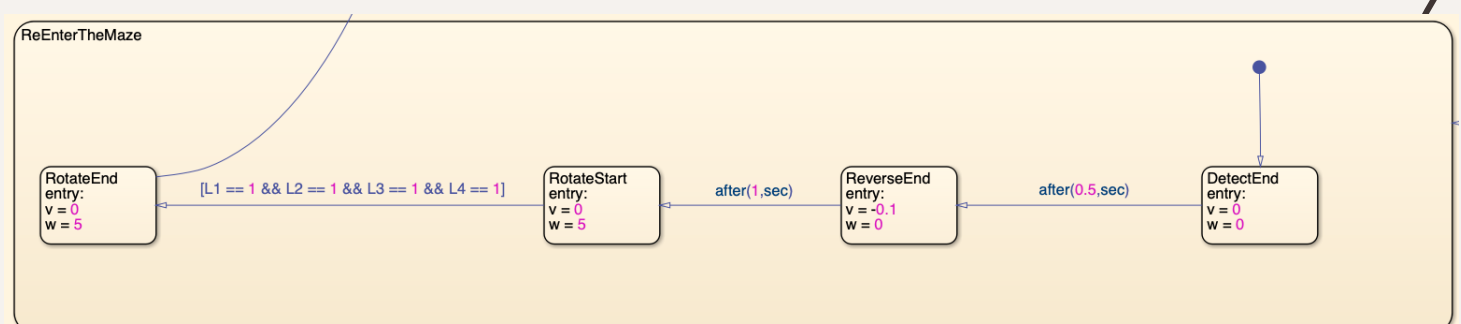


Figure 4e: ReEnter state flow

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

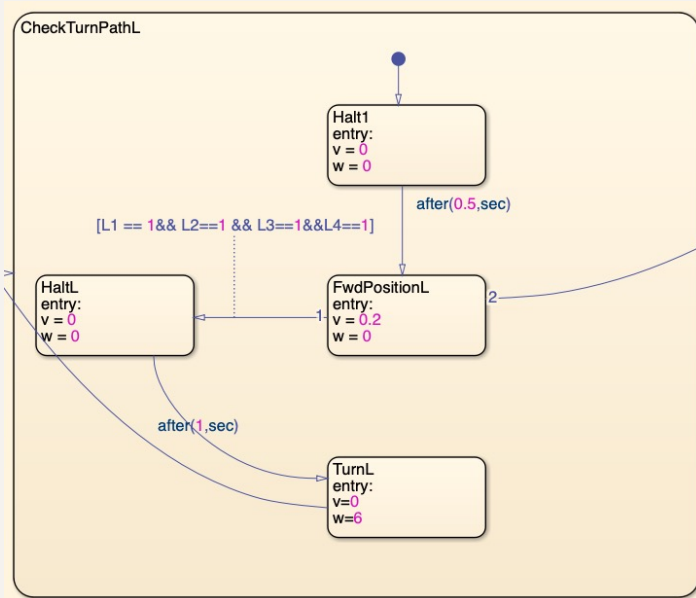


Figure 4f: CheckTurnPathL state flow

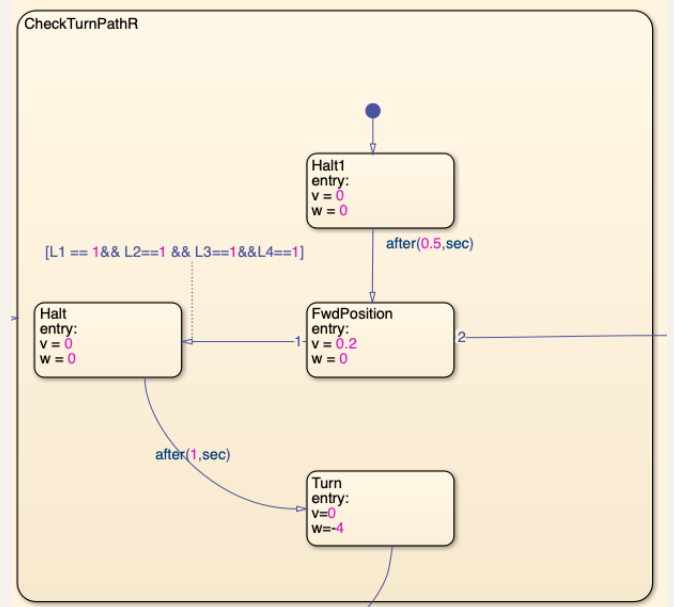


Figure 4g: CheckTurnPathR state flow

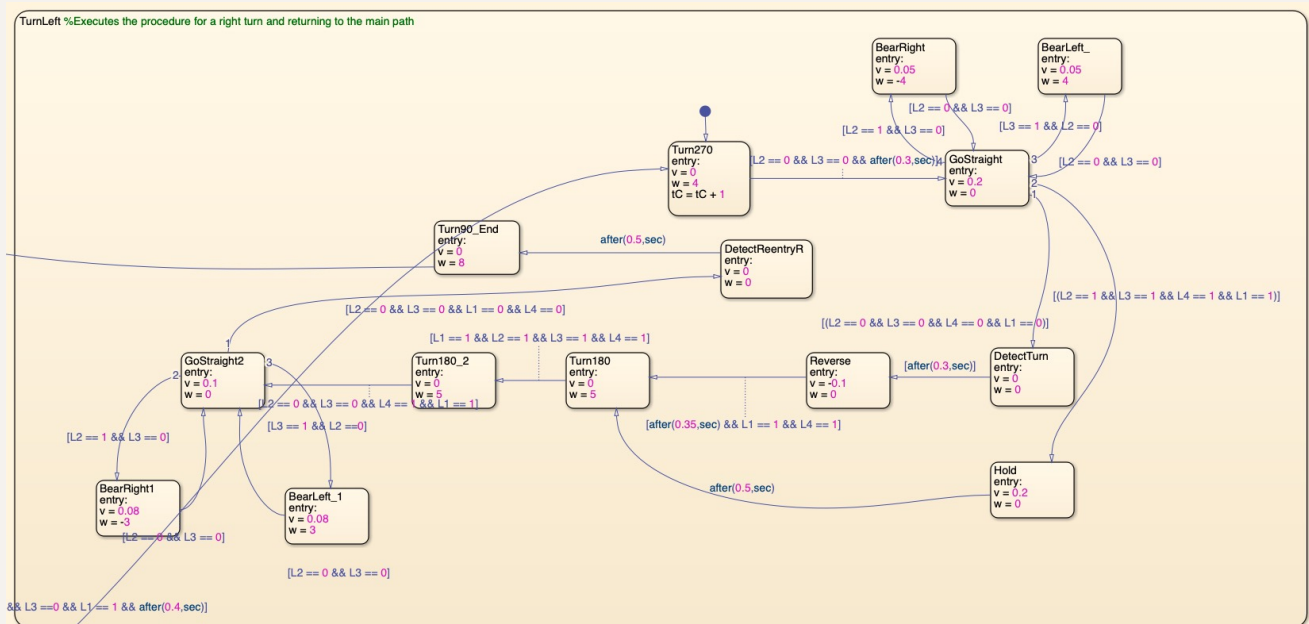


Figure 4h: TurnLeft state flow

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

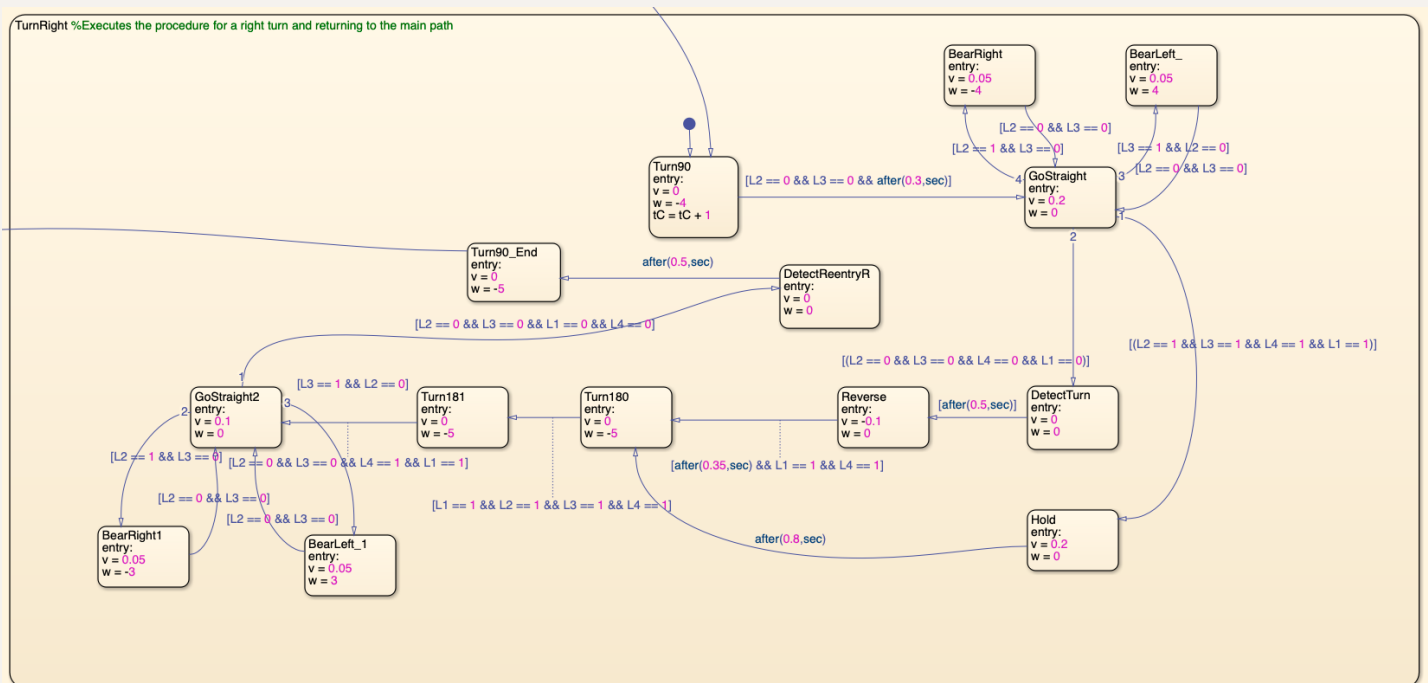


Figure 4f: TurnRight state flow

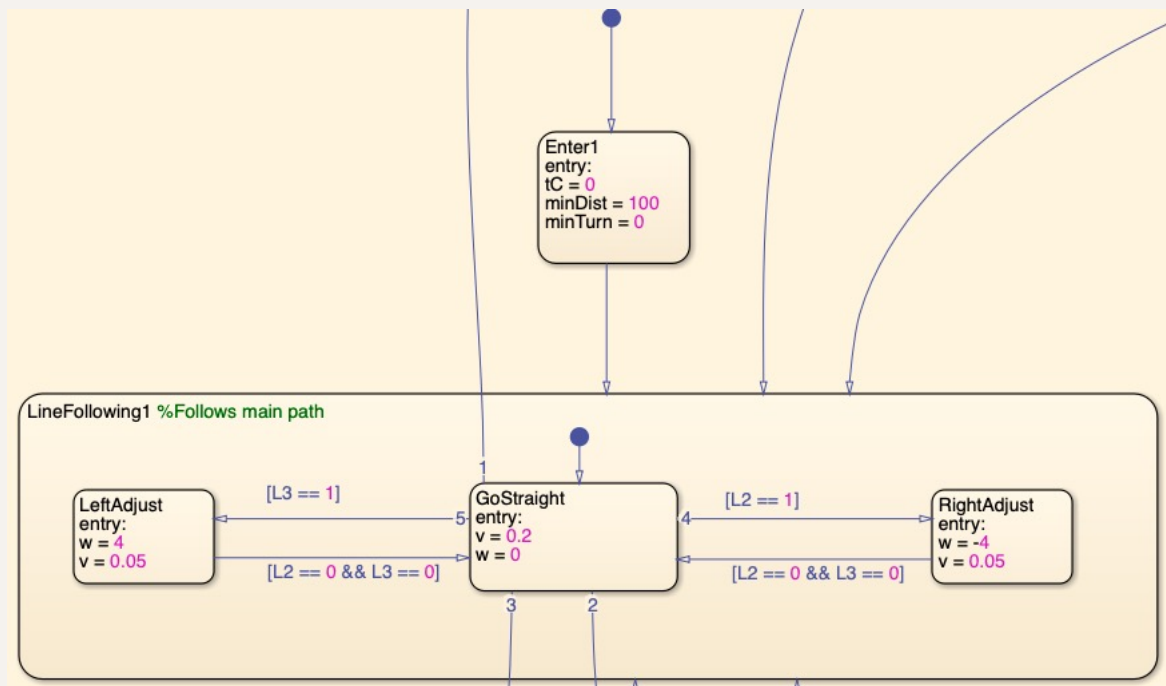


Figure 4f: LineFollowing1 state flow

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## Closest Object Pseudocode

```
ReturnToClosest():  
  Get tC, minBranch, minDist  
  Reverse until L1&L4 = white  
  Rotate 180 degrees  
  while not found  
    LineFollow()  
    if L4 = black  
      if tC = minBranch  
        rightTurn()  
        Stop when all read black  
      Else  
        decrease tC  
    if L1 = black  
      if tC = minBranch  
        leftTurn()  
        Stop when all read black  
      Else  
        decrease tC
```

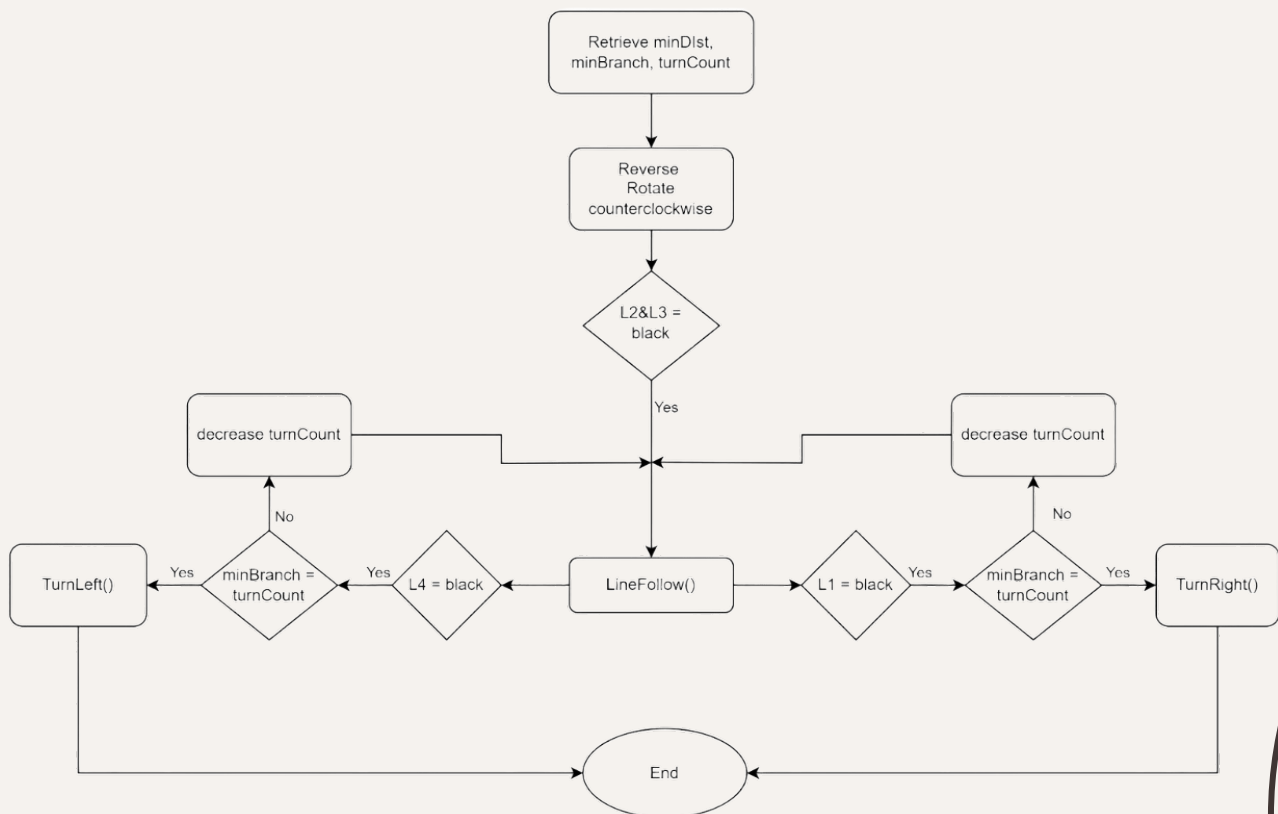


Figure 5a: Ultrasonic Simulink model

# 06 Treasure Maze Solving Algorithm Design, Implementation, and Results

## **Modular Design of Software Down to Function Calls and Including Interrupt/Real-Time structure**

The way functions are called and re-executed are shown in the state flow figure.

Functions used:

- LineFollow() - used to follow the line. The line follow function is repeatedly needed when turning into a left branch, right branch, following the main path and navigating to the treasure.
- TurnRight() - used to turn into the branch detect and turn out. This function is used to make turns and to turn into the closest object. An interrupt is added in the final turn to make sure it stays there instead of turning out. TurnLeft() uses the same logic as TurnRight().
- CheckTurn() - used in the main line following to determine whether it's a path or a 90 degree bend. It's used multiple times, for left turn, right turn and checking in the navigation to the closest object.



# 07 Conclusion

## Software

The MATLAB version greatly affects how the files load. Each member had to ensure that the same version of MATLAB was used or else not all the files would be compiled. Several drivers needed to be installed multiplied or else MATLAB would not load correctly. In addition to this, the ports to the Arduino had to be configured manually through Arduino IDE otherwise it would not be recognised correctly. Sampling time was an issue because it needed to be high enough for each sensor. The robot moved relatively quickly which was why the line sensor needed dew positioning data every few milliseconds. The discrete fixed step and line sensors needed the same sampling time while the Ultrasonic sensor needed 10 times the line sensors' sampling time. Another software issue was how much memory MATLAB relies on. On multiple occasions, one member's laptop needed to be restarted as a result of MATLAB's processing. The last software issue was tracking the states of the robot. Often one could not see that the robot made an error but not understand which state the robot was in because IO Connected testing was much slower than real-time.

## Hardware

Several components of the robot were damaged or no longer worked as intended. The first H-Bridge that was used delivered a lower voltage to the left wheel. The first Arduino burnt out and was no longer recognised by any PCs. The second time this occurred, an Arduino Leonardo was used as a replacement but because it was limited by its memory, it was no longer an alternative. The Leonardo's memory stores 32kB whereas the Arduino Nano 33 IoT stores 256kB. Another hardware issue was the right motor which stopped rotating completely. All these components were replaced with spare components collected from White Lab.

## Final Design and Recommendations

The final design of the robot comprised of the Arduino Nano, a H-Bridge, 4 line sensors, an ultrasonic sensor. The decision to exclude the encoders, and logic converters was made in the interest of low-power consumption and the fact that 3.3V could be sourced from the Arduino itself. Most of the final algorithms were not derived from the initial ones, as the simulation code was almost completely different the physical testing code. One recommendation is using LEDs to track the state of the robot. One LED can be assigned to each state which would enable the programmer to trace which state the robot is in externally. Another recommendation is newer equipment, most of our hardware problems can be attributed to faulty equipment. This would have been more time-efficient.

# Task Allocation

	Tasks
Talon	<ul style="list-style-type: none"><li>• Motion control design, implementation, and results</li><li>• Line sensing design, implementation, and results</li><li>• Treasure Maze solving algorithm design, implementation, and results</li></ul>
Revashan	<ul style="list-style-type: none"><li>• Electrical Design</li><li>• Power calculations</li></ul>
Sayuri	<ul style="list-style-type: none"><li>• Compiled document</li><li>• Introduction</li><li>• Conclusion</li></ul>

# Research References

- The Pi Hut. (2019). HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi. [online] Available at: <https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi> [Accessed 20 Oct. 2023]..
- docs.arduino.cc. (n.d.). Nano 33 IoT | Arduino Documentation. [online] Available at: <https://docs.arduino.cc/hardware/nano-33-iot> [Accessed 20 Oct. 2023]..
- www.robotics.org.za. (n.d.). Logic Level Converter Bi-Directional - Micro Robotics. [online] Available at: <https://www.robotics.org.za/LEVEL-4P>.
- wiki.dfrobot.com. (n.d.). Line\_Tracking\_Sensor\_for\_Arduino\_V4\_SKU\_SEN0017-DFRobot. [online] Available at: [https://wiki.dfrobot.com/Line\\_Tracking\\_Sensor\\_for\\_Arduino\\_V4\\_SKU\\_SEN0017](https://wiki.dfrobot.com/Line_Tracking_Sensor_for_Arduino_V4_SKU_SEN0017).
- wiki.dfrobot.com [Accessed 20 Oct. 2023]..  
(n.d.). Wheel\_Encoders\_for\_DFRobot\_3PA\_and\_4WD\_Rovers\_\_SKU\_SEN0038\_-DFRobot. [online] Available at: [https://www.dfrobot.com/wiki/index.php/Wheel\\_Encoders\\_for\\_DFRobot\\_3PA\\_and\\_4WD\\_Rovers\\_\(SKU:SEN0038\)](https://www.dfrobot.com/wiki/index.php/Wheel_Encoders_for_DFRobot_3PA_and_4WD_Rovers_(SKU:SEN0038)) [Accessed 20 Oct. 2023].