Software Requirement Specifications

Info

Project Name	Maze Solving Robot
Date	31-07-2025
Version	0.1.0
Author	Thribhu Krishnan

Revision History

Version	Author	Version Description	Date Completed
0.1.0	Thribhu Krishnan	Implemented the primitive features of the maze following robot	21/08/25

Review History

Approving Party	Version Approved	Signature	Date

Approval History

Reviewer	Version Reviewed	Signature	Date

Project Link

TempeHS:2025SE-Mechatronics-Thribhu.K

Contents

1. Introduction

1.1. Product Scope

List the benefits, objectives and goals of the product

This product is a maze following robot that is coded in Python with the OOP paradigm as its main idea. It is a very primitive prototype for what could be a potential wall following maze solver, used in different industries. It utilises different components such as Raspberry Pi Pico 2, Servos, Sensors and Displays. On top of that, it also has safety features so it would not collide or step out of line.

This robot can be used for industries like emergency services and warehouses.

1.2. Product Value

Describe how the audience will find value in the product The product will provide value for industries like emergency services and warehouses.

For emergency services, it can aid with locating any victims in any spaces that a normal human being cannot reach, especially during crucial times.

For warehouses, the managers can benefit from the decreased human labour, saving money for salaries as well as 24/7 automation.

1.3. Intended Audience

Write who the product is intended to serve

As specified earlier, it is intended to aid warehouse workers and rescuers in their daily tasks. This is because of the fact that it is automated.

1.4. Intended use

Describe how will the intended audience use this product

Its original intended purpose is to follow a wall and solve a maze. Most likely, the code will be changed and tweaked to fit the usecases of the audience. For example, an emergency rescuer may change the speed to 100% to ensure there is enough time to locate and navigate through the maze.

1.5. General description

Give a summary of the functions the software would perform and the features to be included.

One function should control the robot and get everything started, specifically MazeRunnerStateMachine.update(). Inside that MazeRunnerStateMachine, there is code for updating and pushing the values of the wheel duties to the servos, fetching teh values of the ultrasonic sensor, the logic/controller itself and the lgiht sensor + display.

I expect the robot to navigate around a maze by following a wall.

2. Functional Requirements

List the design requirements, graphics requirements, operating system requirements, and constraints of the product.

The Raspberry Pi Pico 2 was first announced in August of 2024. It follows the same guidelines as its predecessor (Rapberry Pi Pico). It has an improved power efficiency, higher RAM (memory), and higher storage which is useful for large codebases.

The specific chip used is the RP2350, successing the RP2040 used on the original Pico. It is used to power the newer boards like the Raspberry Pi 5. The chip is also RISC-V instead of ARM, which can provide higher performance and lower costs due to lack of licensing from ARM.

The Raspberry Pi Pico 2 runs on the MicroPython language, which is a subset of Python with smaller library sizes fit for microcontrollers.

In terms of hardware, the chassis is built from laser engraved wood, built custom by Mr Jones (our teacher) and carved/engraved at the school. There are also 3D printed components such as a holder for the Pico MCU. On top of that, there is a custom made PCB that connects the different servos, displays, sensors with components such as diodes, capacitators, voltage regulators, fuses, and more, soldered (by hand) by the students themselves (me [Thribhu], Max and Owen).

For the servo, we are provided with a DR15SMG, which is a significant improve from iSTEM's same challenge (which used cheap wheels and badly configured servos).

For the ultrasonic sensors, we used the PiicoDev_Ultrasonic module to read from the values. I made the bound 100mm.

For the light sensor, we used PiicoDev's VEML6040 sensors, which can measure RGB. This was used for sensing the green "victims" (tiles) and stopping when sensed.

For the display, we used PiicoDev's SSD1306, which is a 128x64 pixel screen. I displayed different values, as well as current robot states.

3. External interface requirements

3.1. User Interface Requirements

Describe the logic behind the interactions between the users and the software (screen layouts, style guides, etc.).

For the UI, they display the current state, which is text that is rendered onto the screen. It swaps between the different screens in real time because of the functions being asynchronous.

When a victim is sensed/detected, it will stop in its tracks and start blaring an alarm using the inbuilt speaker (not implemented, will not implement), letting the user know of its aid.

3.2. Hardware interface requirements

List the supported devices the software is intended to run on, the network requirements, and the communication protocols to be used.

So far, the software is tested on the following:

- Raspberry Pi Pico 2
- PiicoDev VEML6040
- PiicoDev SSD1306
- PiicoDev Ultrasonic
- DR15SMG Servo

No network is required as the MCU does not support networking.

For communicating with the different components, they use the I2C protocol to communicate digital data between eachother. I2C uses a master-slave configuration, where the MCU is the master and the component is the slave.

3.3. Software interface requirements

Include the connections between your product and other software components, including frontend/backend framework, libraries, etc.

The MCU's language of choice is Python, specifically MicroPython which is small enough to be placed on a Raspberry Pi Pico 2. There are libraries available for Python, however the main ones are provided by PiicoDev and their team. Their libraries are the backbone for the maze runner python script.

No backend or frontend frameworks are used in the software, and everything communicates using the Pin and PWM classes.

3.4. Communication interface requirements

List any requirements for the communication programs your product will use, like emails or embedded forms.

Due to the robot's MCU not having any sort of wireless capabilities (including radio), it does not communicate at all with **other robots**. With humans, it can communicate perfectly with its beautiful UI on the side of the robot (on the display). It can show the current states on the LCD screen on the side.

4. Non-functional requirements

4.1. Security

Include any privacy and data protection regulations that should be adhered to.

This robot is off the grid, absolutely offline. The MCU does not support any sort of wireless communication such as Bluetooth, WiFi or any other protocol available. Due to this, it would be practically impossible to do an OTA attack, nor be able to share any data with anyone else making it perfectly private.

4.2. Capacity

Describe the current and future storage of your robot

For such a small factor robot (both internally and physically), it does not require a lot of space.

Internally: The codebase is ran on one python script (and libraries), which all fit under the 100mb of storage, perfect for an MCU of a small size.

Physically: The robots chassis is small enough to be stashed in a plastic box in a cupboard and stacked horizontally and vertically.

4.3. Compatibility

List the minimum hardware requirements for your software.

The robot requires an MCU, two servos for wheels, one display for status, two sensors (one for the side, one for the front), a light sensor for detecting green tiles and basic power (which can be supplied by batteries).

4.4. Reliability

Calculate what the critical failure time of your product would be under normal usage.

Under normal time, there is a small margin of error, with there being 10% amount of failures, which is exceptionally good for such an autonomous robot with primitive code

4.5. Scalability

Calculate the highest workloads under which your software will still perform as expected.

In terms of scalability, this robot has a higher chance of crashing due to the code running with Python, which is a garbage collected language and in such a case of high memory usage, it can break and cause the robot to stop working.

4.6. Maintainability

Describe how continuous integration should be used to deploy features and bug fixes quickly.

As stated previously, no OTA updates means that managers cannot update the software in it. The only way to update the robot's internals would be to manually connect to it with a cable, which can be an issue if the manager does not have the required technical knowledge to update the internals.

On the flip side, batteries are easy to replace as it is a basic battery cover. Flipping it open and

4.7. Usability

Describe how easy it should be for end-users to use your software

The entire codebase runs on a fascade pattern, with there being one function (technically two if you init) that you have to call that does everything for you.

For the average Joe, they would not have to interact with the robot, as it does everything for you.

4.8. Other

List any additional non-functional requirements

5. Planning, Modelling and Simulations

5.1. UML Class Diagram

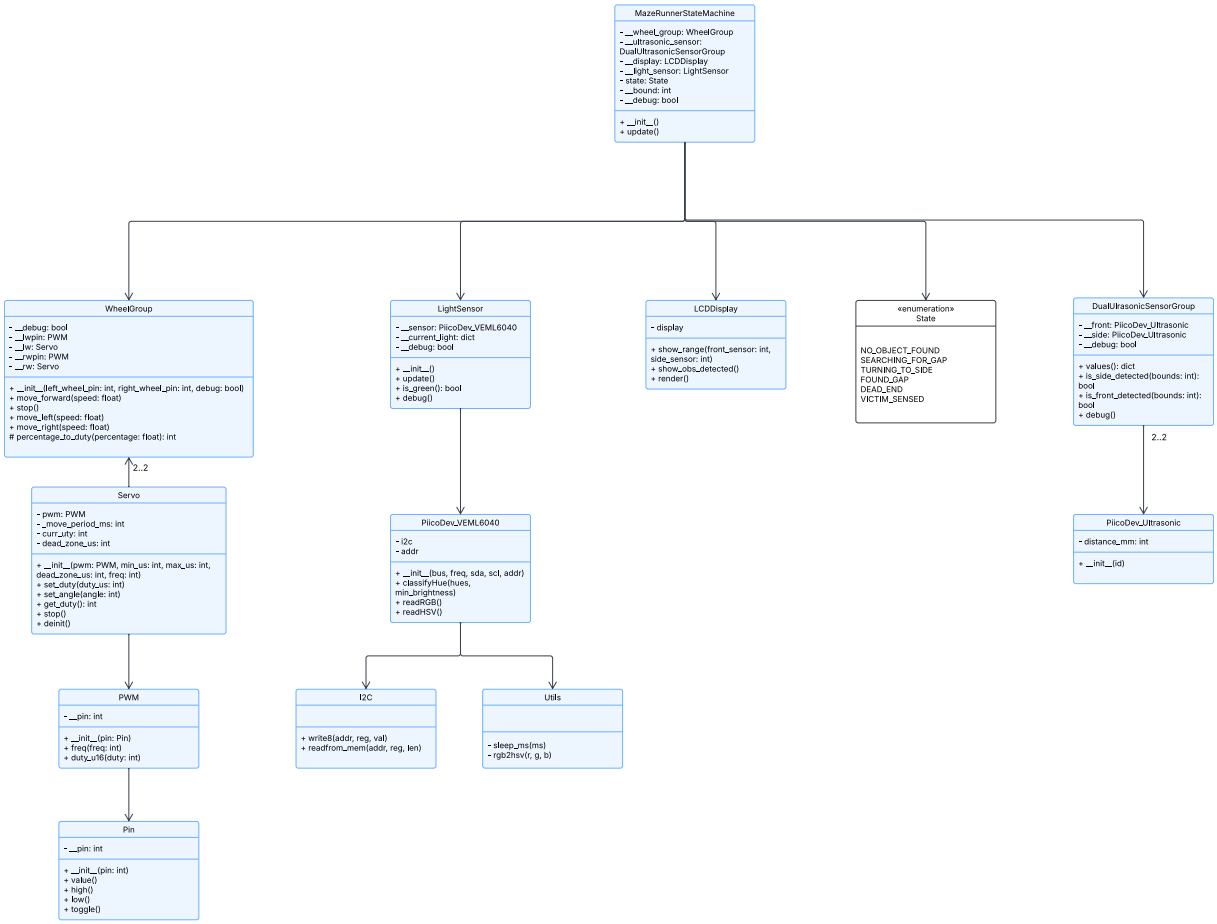


Figure 1: UML Class Diagram

5.2. Data Flow Diagrams

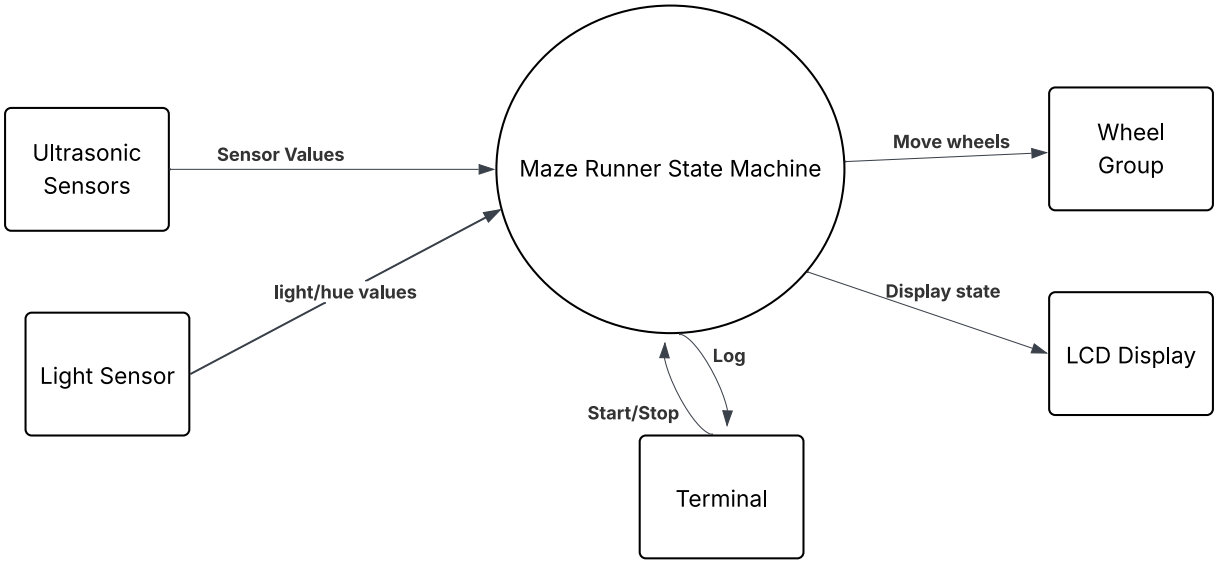


Figure 2: Level 0 Data Flow Diagram

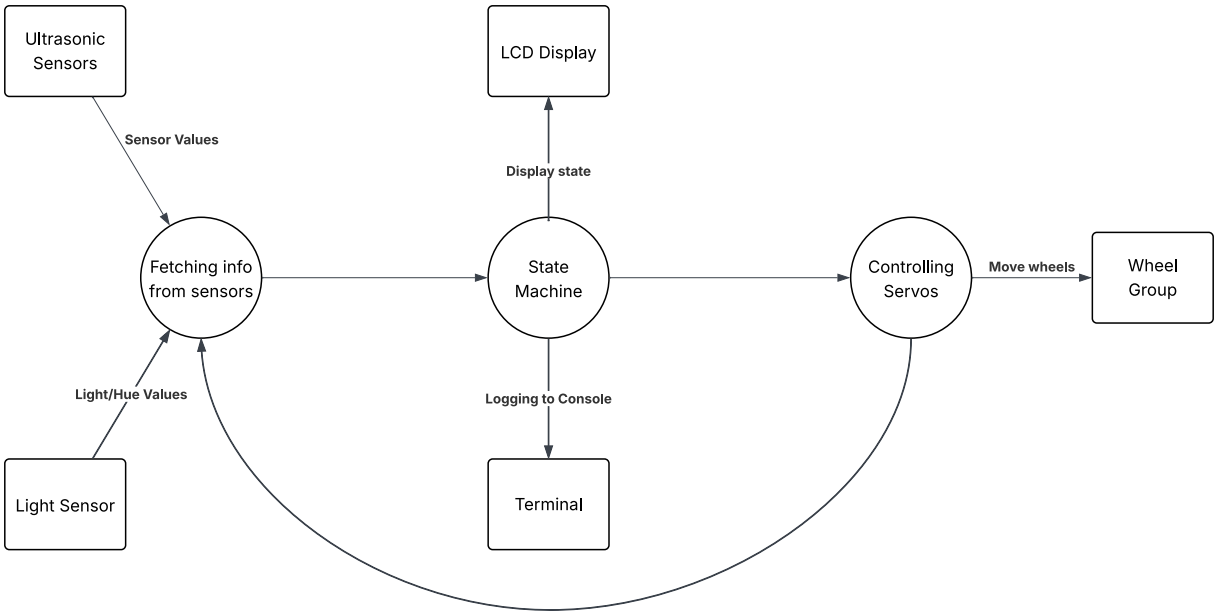


Figure 3: Level 1 Data Flow Diagram

5.3. Flowchart

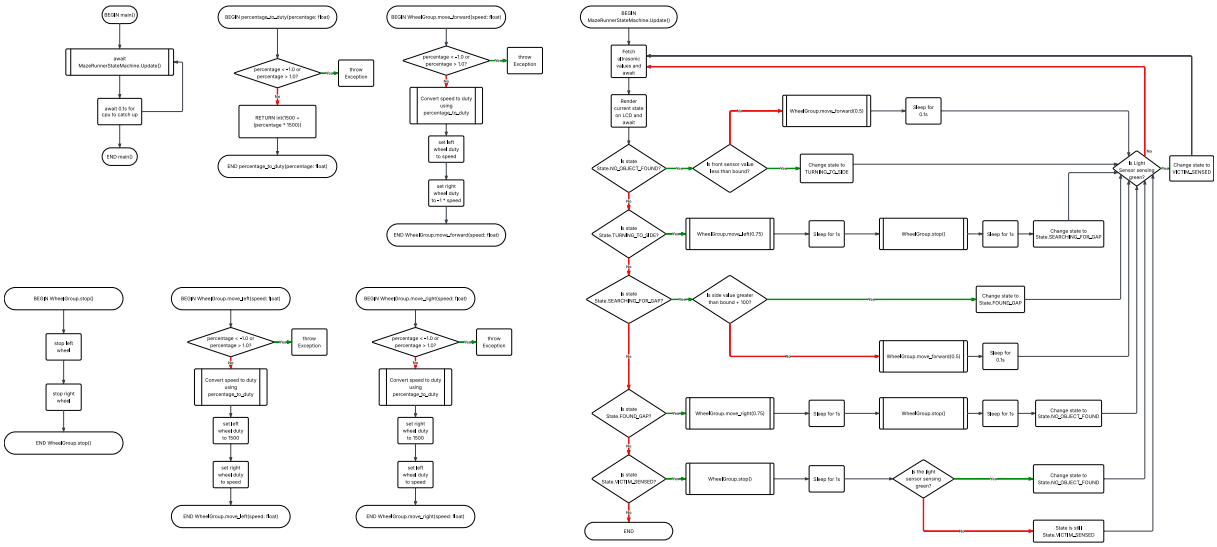
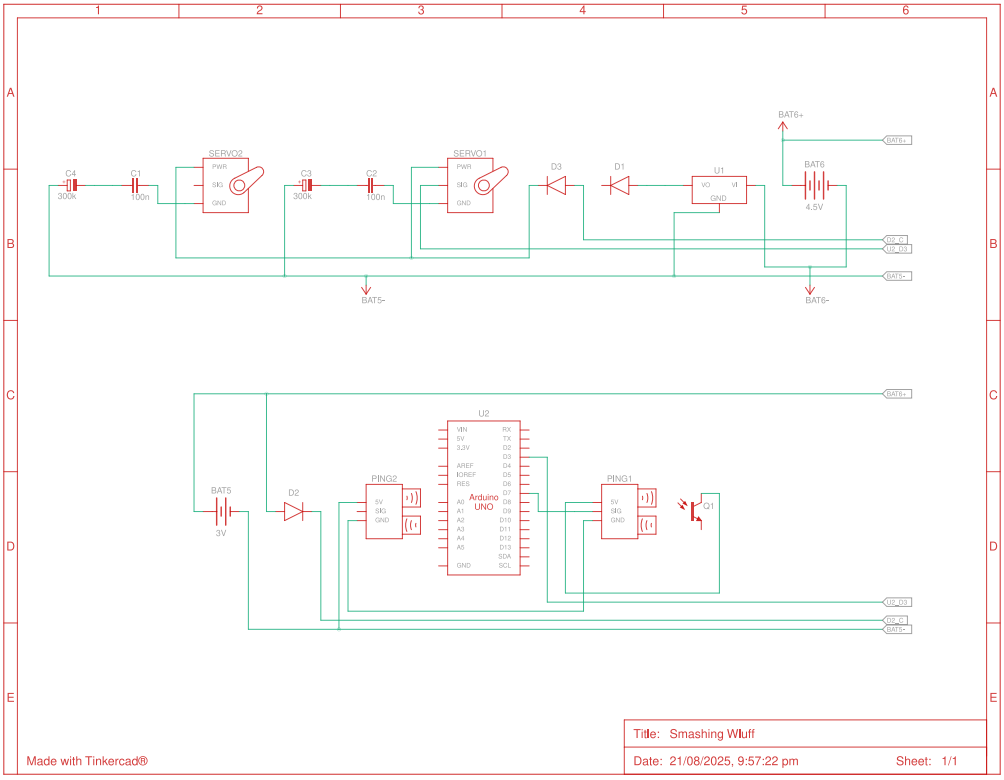


Figure 4: Logic Flowchart

5.4. Wiring Diagram



5.5. Material Components List

- 1x Raspberry Pi Pico 2
- 1x Custom made PCB
  - 2x 100 µf Electrolytic capacitor
  - 2x 100 pf Ceramic capacitor
  - 2x Terminal Blocks
  - 3x Diodes
  - 1x Voltage Regulator
  - 1x Fuse
  - 25x Pin Headers
- 2x RCWL-1601
- 2x PiicoDev_Ultrasonic modules
- 1x Battery Pack + 2x Batteries
- 1x PiicoDev OLED Module SSD1306
- 1x Light Sensor
- 2x Servos
  - 2x Wheels
- 1x PiicoDev Expansion Board
- 1x Omnidirectional Wheel
- Chassis of choice

5.6. Power Supply Diagram and Calculations

Battery Pack (2x 3.7V batteries) = 7.4V ⟶ Capacitators (7.4V - (0.5*2) = 6.4V)

Deamplifier (6.4V to 5V) ⟶ Servos (5V intake)

5.7. Online Simulations

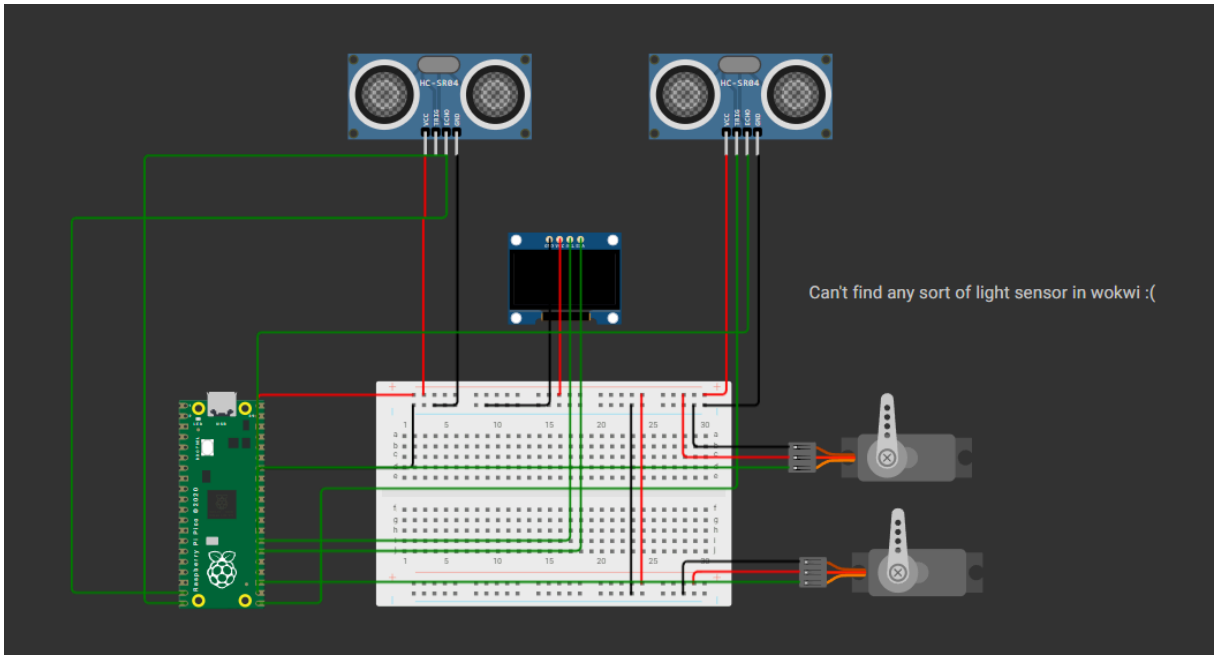


Figure 5: Online Simulation using Wokwi

6. Definitions and acronyms

Word	Definition
MCU	Microcontroller Unit, just an short hand word for the Raspberry Pi Pico 2
OOP	Object Orientated Programming - A paradigm that uses classes to create a “blueprint” that contains data and functions
PWM	Pulse Width Modulation - A method of controlling analog devices and sending power digitally