SYSC4805A_W20_L1

FINAL REPORT

Project Group: B'Dazzled Blue

<u>Group members:</u>

Samy Ibrahim (101037927)

Muhammad Tarequzzaman (100954008)

Jacob Martin (10100849)

Ahmad Chaudhry (101003005)

Table of Contents

Abstract

This document outlines the final report for our SYSC 4805 maze solving project, throughout the document we will be discussing our objective, deliverables, possible improvements and our approach to the solution. We will also present our timeline for expected tasks and milestones through a Gantt chart and a responsibility assignment matrix. Diagrams depicting the hardware used throughout the project as well as a sequence diagram showing the flow of code is also presented. The document then continues by addressing the watch dog, battery sense and hardware control pin test implementations and finishes off with control charts for the requirements of the project.

Overall Objective

The overall object of this project is to design and implement a robot that can solve a simple maze with multiple paths to get to the end objective using multiple sensors working together to guide the robot, the robot will also print out the successful path upon completing the objective.

Overall Deliverables

The overall deliverables of this project will be detailed in this section. The robot is using multiple sensors to allow it to autonomously navigate through a maze. The mentioned maze consists of multiple paths on the ground drawn out with black tape leading to an objective at the end. The required sensors to complete the maze are photodiode and ultrasonic sensors attached to an Arduino (and robot assembled in Lab 1: consisting of the body, motor, and wheels). Currently, we are using four photodiode sensors and one ultrasonic sensor to perform the task at hand. Two of the photodiode sensors are placed at the front of the robot, and one photodiode sensors is placed on both sides of the robot. The photodiode sensors on the front of the robot is used to navigate the robot forwards and keeping it on the specified black line. The sensors on either side of the robot are used to determine if a right/left turn is coming. The ultrasonic sensor is used to determine if there are any obstacles in the path of the robot, if the robot notices an obstacle, it will assume that it has reached the end of the maze. While solving the maze, the robot saves the path it took and once the robot finds the end of the maze (finds the objective), it will then proceed to output its recorded path from memory onto an LCD screen attached to the robot.

Success Criteria

Multiple test Maze's will be developed as testing scenarios for the robot, the Maze's will be designed in a way where the "objective" can be moved around so that the same Maze can be used multiple times for testing. In general, we plan on conducting

- Unit Testing:
    - Test Technical and Functional aspects of the robot separately
    - Write separate tests for every component in the system (just assert sensors are reading data and assert to 1)
    - Line Following Test: create a test that ensures the robot remains on track
        - Set a time limit for the test, and assert that both motors are never off, the robot should always be following the line (if it is adjusting, one motor may be off but the other must remain on)
    - Sensing Objects Test: create a test to ensure the ultrasonic sensor can determine an object is in its way. We will be using varying heights and distances of obstacles to test further.
    - Determining End of Maze Test: The end of the maze will be specified with white tape on the ground, so this test asserts that if both front photodiode sensors get triggered by the white tape, the robot stops and prints its output to the LCD.
    - Printing Successful Path Test: This test asserts that once the robot solves the maze, it indeed prints out the path, the successful (expected) path can be hard coded as a string and then compared to the output of the robot
- Performance Testing:
    - Response Test: we must ensure that the quick output of our sensors doesn't get overridden by another aspect of the system taking too long (for example, a serial print takes a full millisecond, is it needed?). The test will consist of timers (modules) recording the execution time of all methods and loops in the system. These will be analyzed and important methods (such as turning) will need to be optimized.
    - Full Maze Performance Test: this test will record the time it takes to solve the full maze and report which aspects of the robot are taking the longest (so that those areas can be optimized).
- Stress Testing:
    - Big Maze Test: if the maze is S times the size of our current testing one (S times the intersections and turns), how will the robot perform, will it be able to solve the maze in less than (or equal to) S*(usual maze solving time).
    - No Objective Test: This will test the ability of the robot to handle no Objective found at the end of the maze (to know and not keep looping and looking).
- Integration Testing:
    - Test the entire system working together
    - A full run of the robot testing a simple Maze and ensuring that it can be solved successfully, and that the correct path is printed.

- If the system can pass all these test (with acceptable results in non-unit tests), then it behaves as expected and should successfully find the objective.

Possible Improvements

There are multiple areas in the system that could be improved upon and there were many factors that contributed to this. The algorithm used for the maze solving was not the most intelligent as we did not have much time as a team to work on many upgrades because of the COVID-19 outbreak. The plan was to have a multi-dimensional (2 or 3D) array that would keep track of all the intersections that the robot came across gradually eliminating each option (left, right, or straight) as it discovered that the objective is not there (if the objective were found and the ultrasonic sensor was working, it would simply stop there and print the output of the maze).

Another possible improvement to the project (that was apparent from watching the video demonstration) is to get a better power source for the system (enough power). We were using batteries originally to power the system and things were going smooth until we decided to add the two IR sensors on the sides of the robot (at that point it was 1 Ultrasonic, 4 IR Sensors, LCD, and 2 motors). Obviously, the robot kept having issues and would not perform as we wanted it to, it kept getting stuck for no reason and sensors were unresponsive. We were advised not to leave the house at the time and could not leave the house to buy some external battery packs, and amazon was delaying deliveries by at least two weeks because of the outbreak. Because of that, we had to use a portable power bar (generally used for charging phones) to power it for the time being.

Another possible improvement would be to use a linked list instead of a fixed size array to keep track of all the turns for printing. This was not implemented as it was not crucial and the robot functioned perfectly fine without it, but none the less would be a great improvement in terms of memory management (we create and add a new node when needed instead of having the array allocated at runtime and populated during the execution of the system).

Another possible improvement to the system is to add a color sensor on the front with the Ultrasonic sensor that would work hand in hand. If we used a color sensor, we could use something like an orange or an apple as our "goal" at the end of the maze (really anything of a specific color that could be sensed by the Arduino color sensor). The system would function in the exact same way, but instead of simply reading the distance of an object and deciding whether to stop, the robot would analyze how far the object is and try to analyze its color as well, if it is the expected color (orange if we use an orange), then the robot would stop, otherwise it would turn around and keep searching.

Trigger Type

The Maze Solver we are building is an Event Triggered system meaning that the main flow is controlled by specific events rather than relying on time. The main reason for "Events" in our system is the robot getting off its course (getting off the black line/ maze), if that specific event occurs, then the robot immediately fixes itself (gets back onto the line). Another big event is coming across an intersection (places where the robot can turn), if that happens, the robot will have to remember what decision it made and turn into one of the paths.

Requirements

Based on the overall deliverables of the project the following hardware components will be required to develop the maze solving robot:

- 1 x Robot Physical Hardware With 2 Motor and Wheels x 2
- 1 x Arduino nano
- IR Optical Sensors x 2 + x
- Ultrasonic Sensor x 1
- L9110S Dual H-Bridge motor driver x 1
- Motor Shield board x 1
- Connection Wires
- Switch x 1
- LCD Screen x 1
- Battery Holder x 1
- Battery x 4 + x2

WBS (Work Breakdown Structure)

1. Project Design
    a. Coming up with project idea
    b. Design the project and write the Proposal
2. Implement Simple Robot
    a. Assemble Robot body with the frame, motor, and wheels.
3. Add Maze Solver Specific Hardware and Line Following
    a. Add ultrasonic and photodiode sensors
    b. Design and Implement an algorithm to follow a black line
4. Design Algorithms for Maze Solving
    a. Turn around if obstacle or end of path is found
        i. Implement ultrasonic sensor to sense obstacles
        ii. Add turning functionality
    b. Keep trying paths until objective is found
        i. Avoid already used paths
5. Design Algorithm for Printing Successful Path

    a. Store successful/unsuccessful paths in 2D arrays
    b. Display successful path on LCD screen
6. Functional Testing of Robot Sensors and Functionality
    a. Unit Testing
    b. Integration Testing
7. Non-Functional Testing of Robot
    a. Performance Testing
        i. Time critical system maintained
        ii. Watchdog Timer activation to avoid miss deadline.
        iii. Battery Sense
    b. Stress Testing
8. Verification & Validation
    a. Code verification
    b. Integration of verified code
    c. Functionality and Code validation

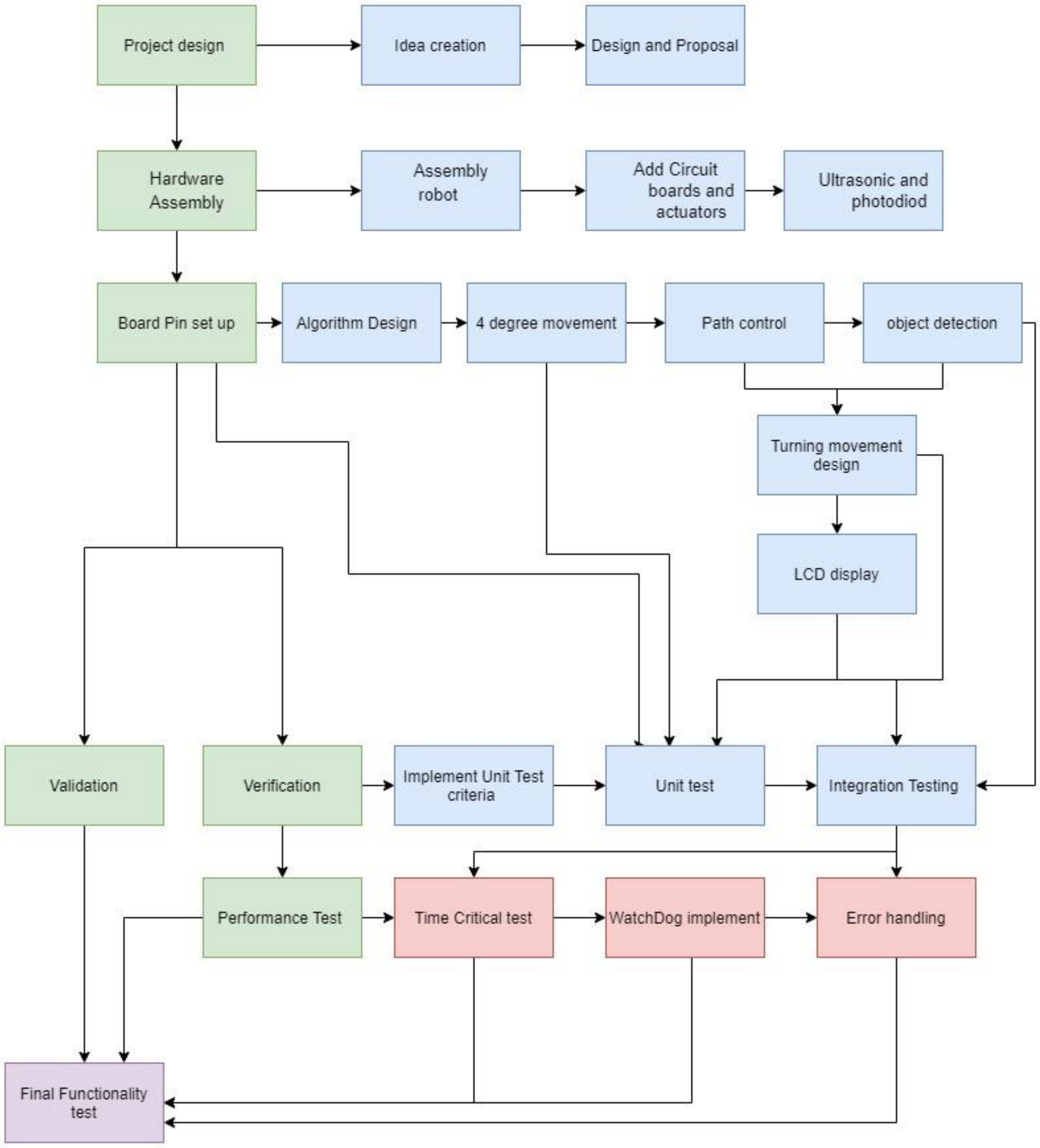


Figure 1: WBS diagram

Schedule Network Diagram

This diagram in fig.1 outlines our schedule for the defined task in respect to each other.

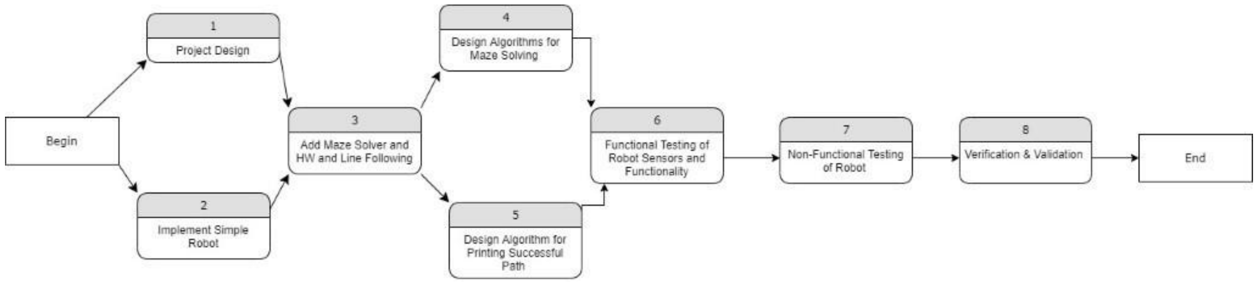

Begin
1
Project Design
2
Implement Simple Robot
3
Add Maze Solver and HW and Line Following
4
Design Algorithms for Maze Solving
5
Design Algorithm for Printing Successful Path
6
Functional Testing of Robot Sensors and Functionality
7
Non-Functional Testing of Robot
8
Verification & Validation
End

Figure 2: Schedule Network Diagram

Gantt Chart

Gantt chart diagram in fig. 2 out lining our task with respect to labs and dates.

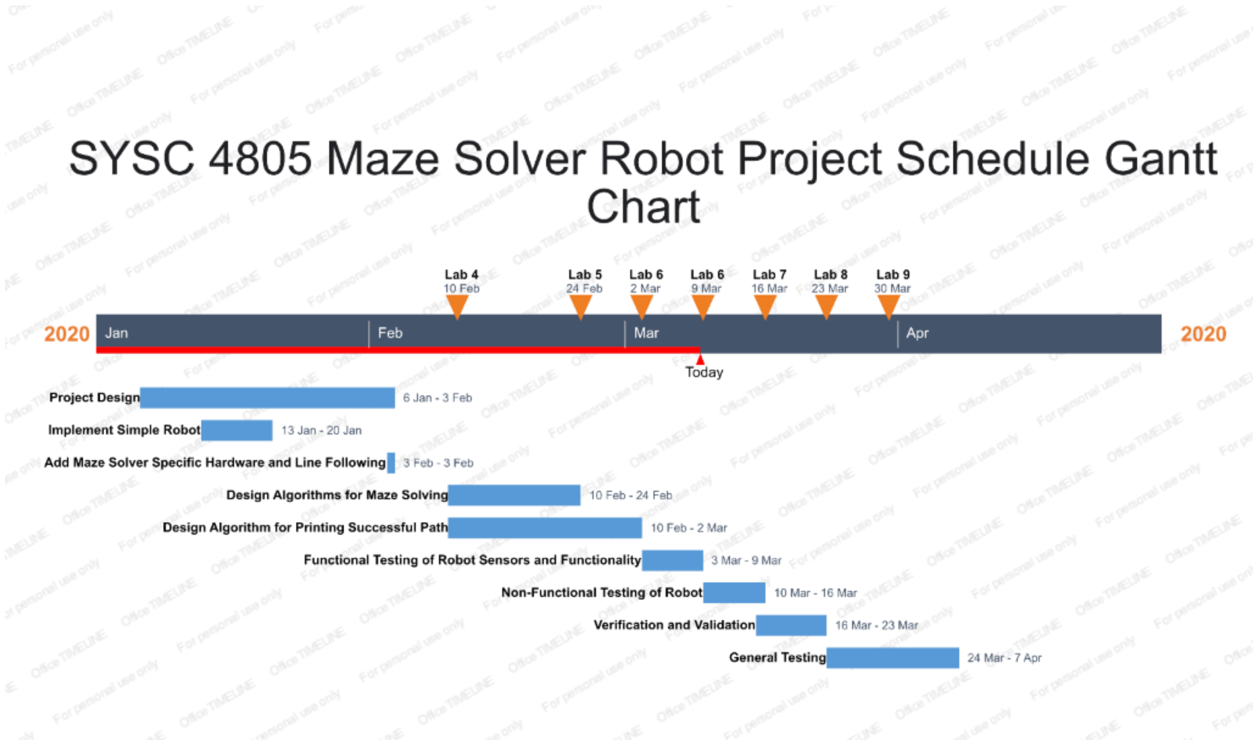


Figure3: Gantt Chart

Responsibility Assignment Matrix

List the key personnel who will be responsible for completion of the project, as well as other personnel involved in the project.

MT: Muhammad Tarequzzaman	AC: Ahmad Chaudhry	SI: Samy Ibrahim	JM: Jacob Martin

ACTIVITY	RESPONSIBE	APPROVER	Contribution	Informed
1.a -Coming up with project idea	JM	SI	AC	MT
1.b -Design the project and write the Proposal	SI	AC	MT	JM
2.a -Assemble Robot body with the frame, motor, and wheels.	AC	JM	MT	SI
3.a -Add ultrasonic and photodiode sensors	JM	AC	SI	MT
3.b -Design and Implement an algorithm to follow a black line	JM	SI	AC	MT
4.a -Turn around if obstacle or end of path is found	JM	SI	AC	MT
4.b -Keep trying paths until objective is found	SI	JM	AC	MT
5.0 – 2d Array Implementation	SI	JM	AC	MT
5.a -Store successful/unsuccessful paths in 2D arrays	SI	JM	AC	MT
5.b -Display successful path on LCD screen	AC	MT	JM	SI
6.a – Hardware Pin Unit Testing	MT	SI	AC	JM
6.b -Integration Testing	AC	MT	SI	JM
6.c – Watchdog	MT	AC	JM	SI
6.d - Battery Sense	MT	AC	SI	JM
7.a -Performance Testing	MT	SI	JM	AC
7.b -Stress Testing	AC	MT	SI	JM
7.b.i- Proper turning during obstacle detection	SI	AC	JM	MT
7.b.ii - Controlling its path while on the line	JM	SI	AC	MT
8.a -Code verification	MT	SI	AC	JM
8.b -Integration of verified code	MT	AC	JM	SI
8.c-Functionality and Code validation	SI	MT	AC	JM

Circuit Diagram

Below is a detailed circuit diagram of the maze solving robot:

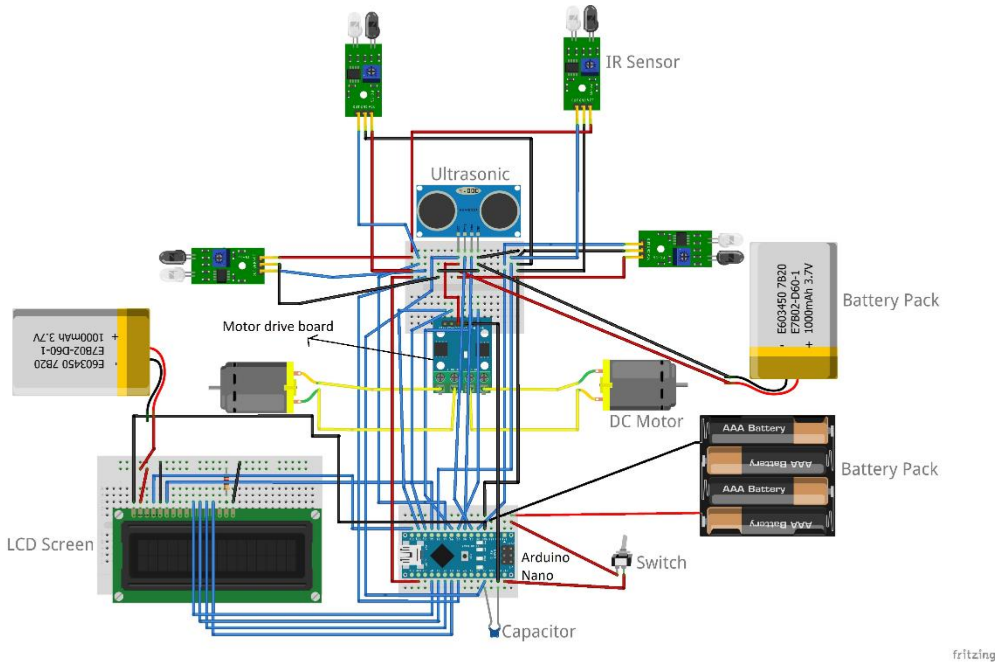


Figure 4: Circuit Diagram

Circuit Schematics

Below is a detailed circuit schematic diagram of the maze solving robot:

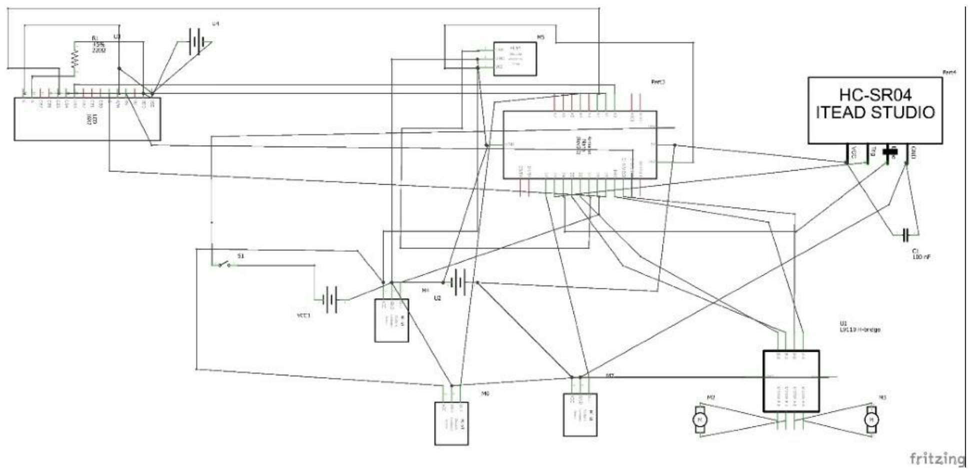


Figure 5: Schematics Diagram

Sequence Diagram

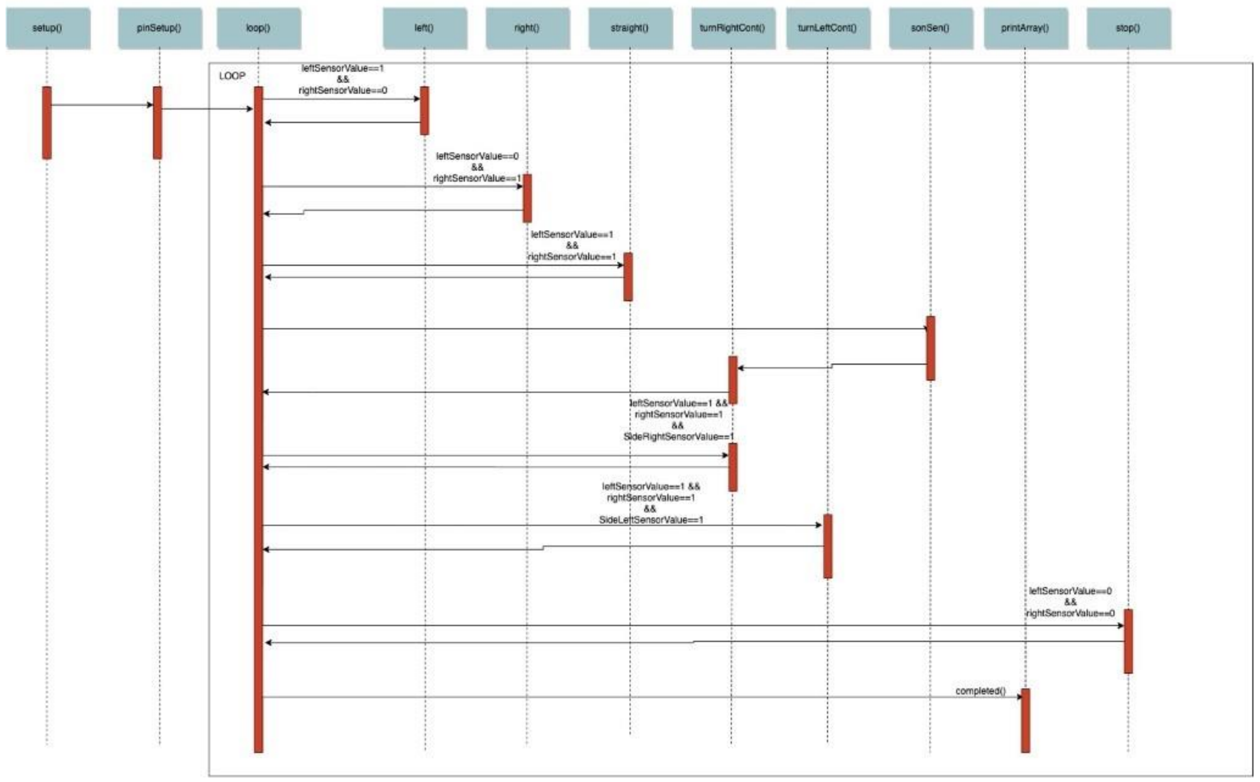

setup()
pinSetup()
loop()
left()
right()
straight()
turnRightCont()
turnLeftCont()
sonSen()
printArray()
stop()
LOOP
leftSensorValue==1 && rightSensorValue==0
leftSensorValue==0 && rightSensorValue==1
leftSensorValue==1 && rightSensorValue==1
leftSensorValue==1 && rightSensorValue==1 && SideRightSensorValue==1
leftSensorValue==1 && rightSensorValue==1 && SideLeftSensorValue==1
leftSensorValue==0 && rightSensorValue==0
completed()

Figure 6: Sequence Diagram

Watchdog:

Watchdog is a crucial part of the MCU fault tolerance design. In order to implement watchdog timer on atmega328p two register value need to be configured, MCUSR and WDTCSR. The WDTCSR has the 4 different type of bits to configure for desired time of reset.

Figure 10-7. Watchdog Timer

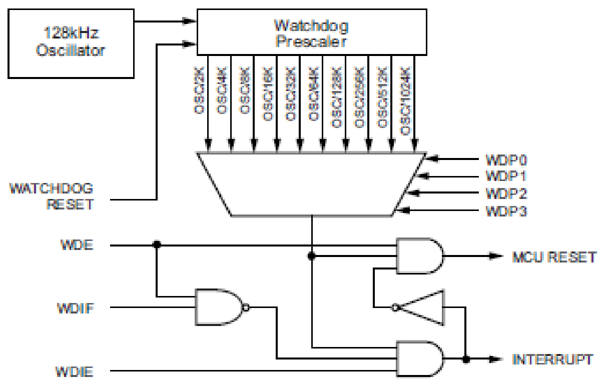


Figure 7: WDT MCU control diagram.

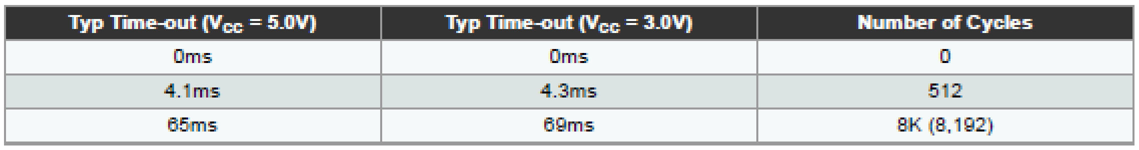
A separate watchdog timer setup function < void WDT_Initializing_ResetTime(uint8_t value) > has been created for this project, in order to implement the timer without any preexisting Arduino library function support. Function gets an input for switch-case such as WDTO_1S and implement the appropriate register value and setup Prescaler value of the oscillator. In Appendix A5 [Code 1] have an example for 1s,2s and 4s timer configuration on WDTCSR.

For the interrupt mode, Interrupt and system reset mode has been configured by using WDCE and WDE bits of the WDTCSR to set 1. Appendix A5 [Code 2] have an example of that setup.

For MCUSR, this register needs to be set 0 before any change must be done on WDTCSR. MCUSR will set to 1 in case of any interrupt call by watchdog and MCU will response that call as a hardware interrupt. Therefore, in order to avoid any setup time fault, a Non- interrupt environment has been made before any set up by calling Disable interrupt function CLI. Once all the setup has been done, SLI call enable the interrupt again. Appendix A5 [Code 3] is an example of cli and sli call.

The WDT timer depends on the frequency of its internal clock (128kHz). One second delay may not show as real time, instead a full one second MCU clock time consider as complete second. Which is 1/frequency of MCU clock. This clock (WDT+SYSTEM timer) cycle is depending on the Voltage input that MCU gets from main battery. According to data sheet Table 8-2, Vcc needs to be on between 5V >= Vcc => 3.0 V to get a reasonable tick count.

Table 8-2. Number of Watchdog Oscillator Cycles



Typ Time-out (V_{CC} = 5.0V)	Typ Time-out (V_{CC} = 3.0V)	Number of Cycles
0ms	0ms	0
4.1ms	4.3ms	512
65ms	69ms	8K (8,192)

Therefore bellow 3.0 V watchdog timer will not able to invoke interrupt at the given timer value.

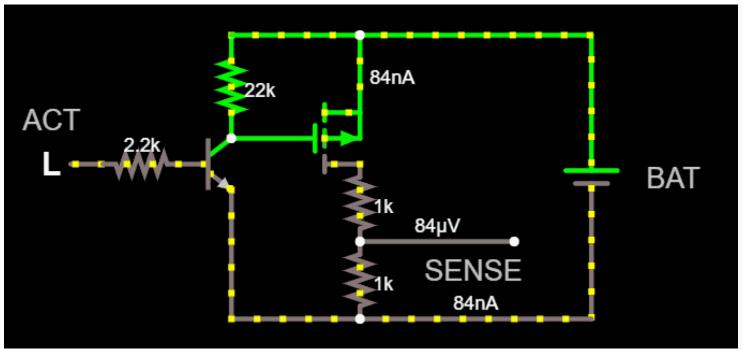
To reset the watchdog, <_asm__ __volatile__ ("wdr”); > needs to be called before ending the loop in the main function. This asm script will invoke “wdr” flag to reset its mode. Fail to invoke the flag on the initialized setup time boundary will trigger watchdog interrupt and system reset.

Battery Sense:

Our system has 3 separate battery with different voltage capacitance and load resistance. 2 of those were 3.1 V and one were 5.0V, that one was connected to the Arduino. Top run smooth system operation, 5.V battery pack always need to be above 4.5V for operational capacity.

To implement a battery sense circuits, we had to design a voltage divider circuits and use Battery Sense Arduino library.

The voltage divider circuits require 2 Register, R1 and R2 and Arduino Analog connection in between those 2 registers. The value of R1 >= R2, not the other way around. The Switch (MOS / BJT) will activate (ACT on the figure bellow) the circuits in order to get an analog voltage sense data.



```
                                  +---------+
BAT+ ---+--------- VIN |         |
        |              |         |
        SW--------- ACT |        |
        |              |         |
        R1             |         |
        |              | Arduino |
        +------- SENSE |         |
        |              |         |
        R2             |         |
        |              |         |
BAT- ------------- GND |         |
                       +---------+
```

Figure 8: Battery sense circuits diagram.

***NOTE**: Due to COVID19 situation we could not able to acquire the BJT and Registers, in order to implement the circuits on the system.*

For the software, we did implement the Battery sense. Appendix A5 [Code 4] shows the setup configuration.

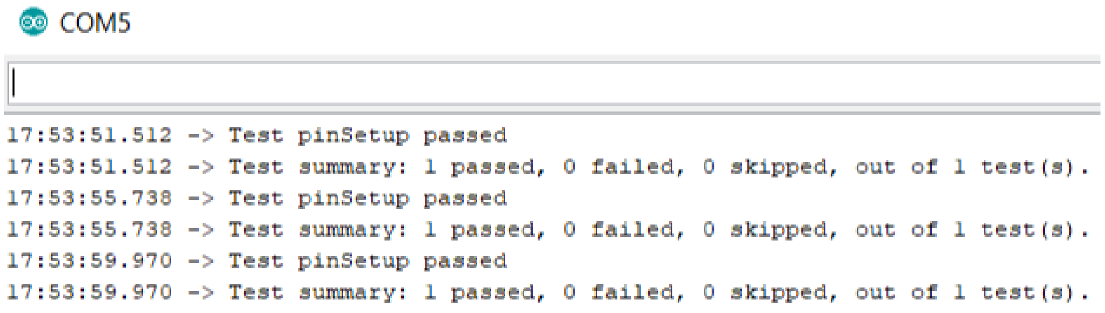
If the circuits were to be implemented after the setup, by calling < Serial.print(battery.voltage());> periodically in the loop using time delay estimate function, will show the battery value of the 5.0V battery pack on the LCD or serial print monitor.

Hardware PIN Unit Test:

For the Unit test of Hardware pin setup and initialization. We developed an assertation test bench script and run it by Arduino Unit test Framework called *ArduinoUnit*. We chose this step to reduce any resource wasting scenario, such as unusual serial transmission. Appendix A5 [Code 5] shows the assertation script for all the initialized pin and their mode of operation.

The < uint8_t readPinMode(uint8_t pin)> function were implemented in order to determine the targeted pin's input / output mode. Appendix A5 [Code 6] shows its implementation.

After implementing the test script, the Test run were implemented inside the setup function using a while loop for 10 cycle. Within this 10 -cycle, test run will generate and display the test summery on serial monitor or LCD. Appendix A5 [Code 7] have the code example and Figure bellow shows an example of the < pinSetup();> functions unit Test,



```
17:53:51.512 -> Test pinSetup passed
17:53:51.512 -> Test summary: 1 passed, 0 failed, 0 skipped, out of 1 test(s).
17:53:55.738 -> Test pinSetup passed
17:53:55.738 -> Test summary: 1 passed, 0 failed, 0 skipped, out of 1 test(s).
17:53:59.970 -> Test pinSetup passed
17:53:59.970 -> Test summary: 1 passed, 0 failed, 0 skipped, out of 1 test(s).
```

Figure 9: TEST summery

Once the test passed, the Arduino will start its other scheduled task and proceed to the main loop function.

Issues Faced

As with all projects there are complications and bugs that appear as the project develops. this section will outline the issues faced during the project development and what actions where faced to correct them.

The largest issue that no one could of predict happening was the effect of covid-19 and the cancellations of face to face tasks. This present a unique challenge for the team to find ways of continuing the development of the project while being separated. Ahmad Chaudhry took possession of the robot and was the key team member in all testing and development. Group calls were made to continue the development in an online environment and peer programming was a key concept used for the development to save time on the process of not having to push code to the GitHub every time we wanted to test. So instead we all sat together and as one person typed, we would all try to solve the problem together.

Another issue faced was the reliance of some components. In specifics our ultrasonic sensor was have issue of reading the correct values during the testing. In the end we needed to continue the development as we were not able to obtain a new part so the code of it working was added but was commented out during the testing face so that it would not cause errors for other parts of the robot. As well the IR sensors attached to the robot would come loose often and interfere with the line detection of the robot causing it to go off path. To overcome this, we were able to hot glue gun on the IR sensors to a position that allowed the robot to sense the correct paths and not all for them to come loose.

The maze itself had to also be specifically designed for the robot to allow the IR sensors to read the paths correct. IF the lines were too big, we would have trouble sensing alternate paths at the robot would think it needs to turn when it doesn’t. If the lines were to small the robot could not travel straight correctly. After many redesigns of the maze we were able to figure out the need thickness of the lines and have a working maze the robot could travel on.

Due the inclusion of many parts on the robot there was a strong draw for power and the inclusion of another outside battery source was needed. If the battery was not included, we noticed that one of the motors that turned the wheels had trouble keeping up with the other and would often shut down when it was supposed to be running.

Control Charts

Ultrasonic Control Chart:

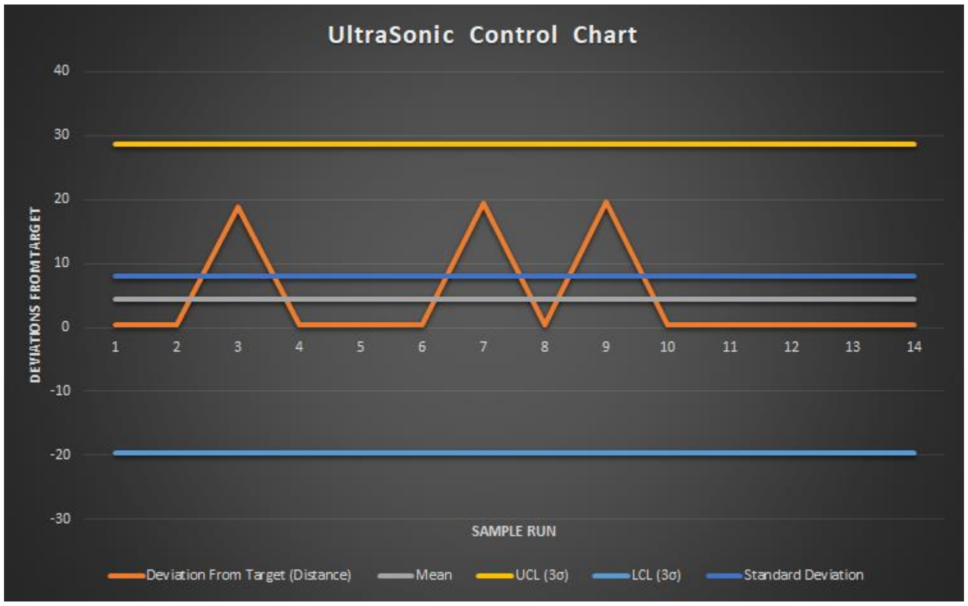


Figure 10: Ultrasonic Control Chart

Sample Run	Deviation from Target (Distance)	Mean	UCL (3σ)	LCL (3σ)	Standard Deviation
1	0.45	4.49	28.60880594	-19.62880594	8.03960198
2	0.44	4.49	28.60880594	-19.62880594	8.03960198
3	18.74	4.49	28.60880594	-19.62880594	8.03960198
4	0.44	4.49	28.60880594	-19.62880594	8.03960198
5	0.44	4.49	28.60880594	-19.62880594	8.03960198
6	0.44	4.49	28.60880594	-19.62880594	8.03960198
7	19.5	4.49	28.60880594	-19.62880594	8.03960198
8	0.45	4.49	28.60880594	-19.62880594	8.03960198
9	19.72	4.49	28.60880594	-19.62880594	8.03960198
10	0.45	4.49	28.60880594	-19.62880594	8.03960198
11	0.44	4.49	28.60880594	-19.62880594	8.03960198
12	0.45	4.49	28.60880594	-19.62880594	8.03960198
13	0.45	4.49	28.60880594	-19.62880594	8.03960198
14	0.45	4.49	28.60880594	-19.62880594	8.03960198

Table 1: Data for Ultrasonic Control Chart

Intersections Turn

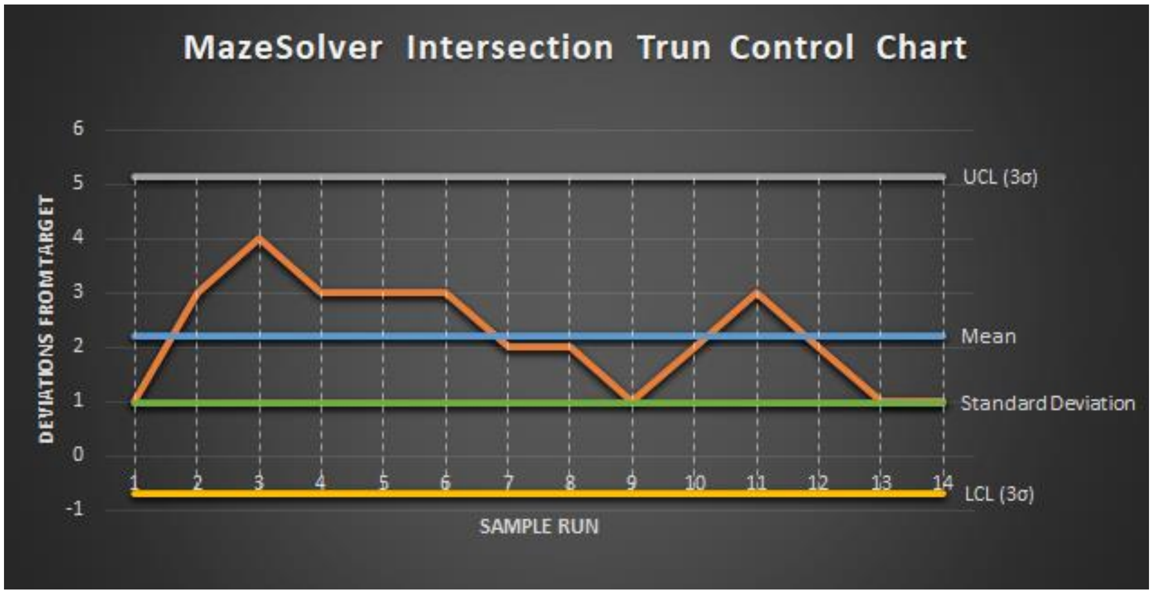


Figure 11: Intersection Turns Control Chart

Sample Run	Deviation from Target (didn't complete intersection properly)	UCL (3σ)	LCL (3σ)	Mean	Standard Deviation
1	1	5.139169	-0.7106	2.214286	0.974961256
2	3	5.139169	-0.7106	2.214286	0.974961256
3	4	5.139169	-0.7106	2.214286	0.974961256
4	3	5.139169	-0.7106	2.214286	0.974961256
5	3	5.139169	-0.7106	2.214286	0.974961256
6	3	5.139169	-0.7106	2.214286	0.974961256
7	2	5.139169	-0.7106	2.214286	0.974961256
8	2	5.139169	-0.7106	2.214286	0.974961256
9	1	5.139169	-0.7106	2.214286	0.974961256
10	2	5.139169	-0.7106	2.214286	0.974961256
11	3	5.139169	-0.7106	2.214286	0.974961256
12	2	5.139169	-0.7106	2.214286	0.974961256
13	1	5.139169	-0.7106	2.214286	0.974961256
14	1	5.139169	-0.7106	2.214286	0.974961256

Table 2: Data for Intersection Turn

LCD Print

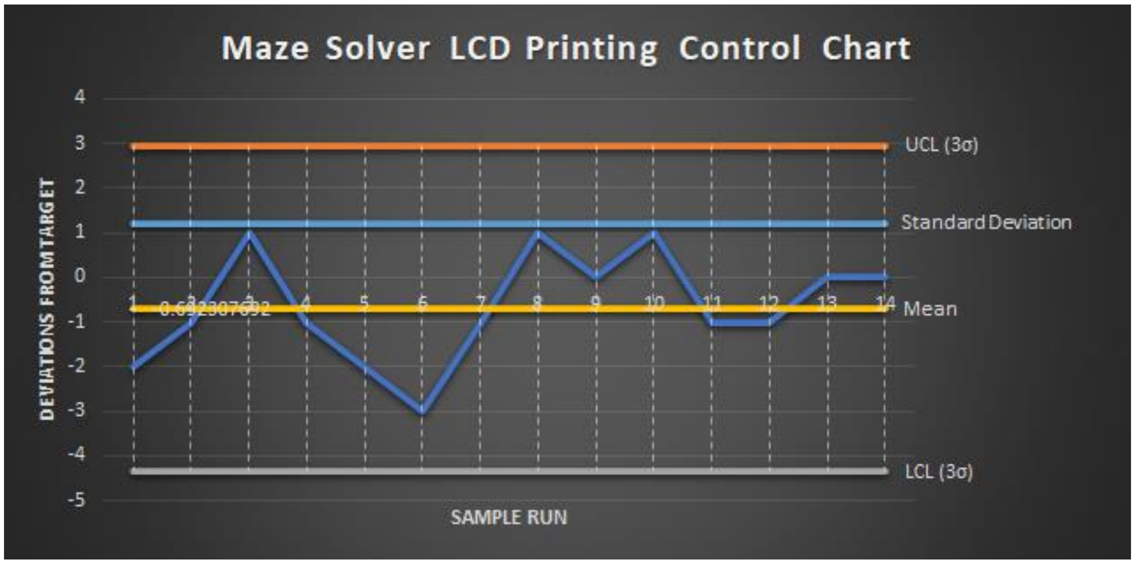


Figure 12: Control Chart for LCD Printing

Sample Run	Deviation from Target (Went off path and needed human correction)	UCL (3σ)	LCL (3σ)	Mean	Standard Deviation
1	-2	2.95491012	-4.339526	-0.69231	1.215739272
2	-1	2.95491012	-4.339526	-0.69231	1.215739272
3	1	2.95491012	-4.339526	-0.69231	1.215739272
4	-1	2.95491012	-4.339526	-0.69231	1.215739272
5	-2	2.95491012	-4.339526	-0.69231	1.215739272
6	-3	2.95491012	-4.339526	-0.69231	1.215739272
7	-1	2.95491012	-4.339526	-0.69231	1.215739272
8	1	2.95491012	-4.339526	-0.69231	1.215739272
9	0	2.95491012	-4.339526	-0.69231	1.215739272
10	1	2.95491012	-4.339526	-0.69231	1.215739272
11	-1	2.95491012	-4.339526	-0.69231	1.215739272
12	-1	2.95491012	-4.339526	-0.69231	1.215739272
13	0	2.95491012	-4.339526	-0.69231	1.215739272
14	0	2.95491012	-4.339526	-0.69231	1.215739272

Table 3: Data for LCD Print

Path Corrections Needed

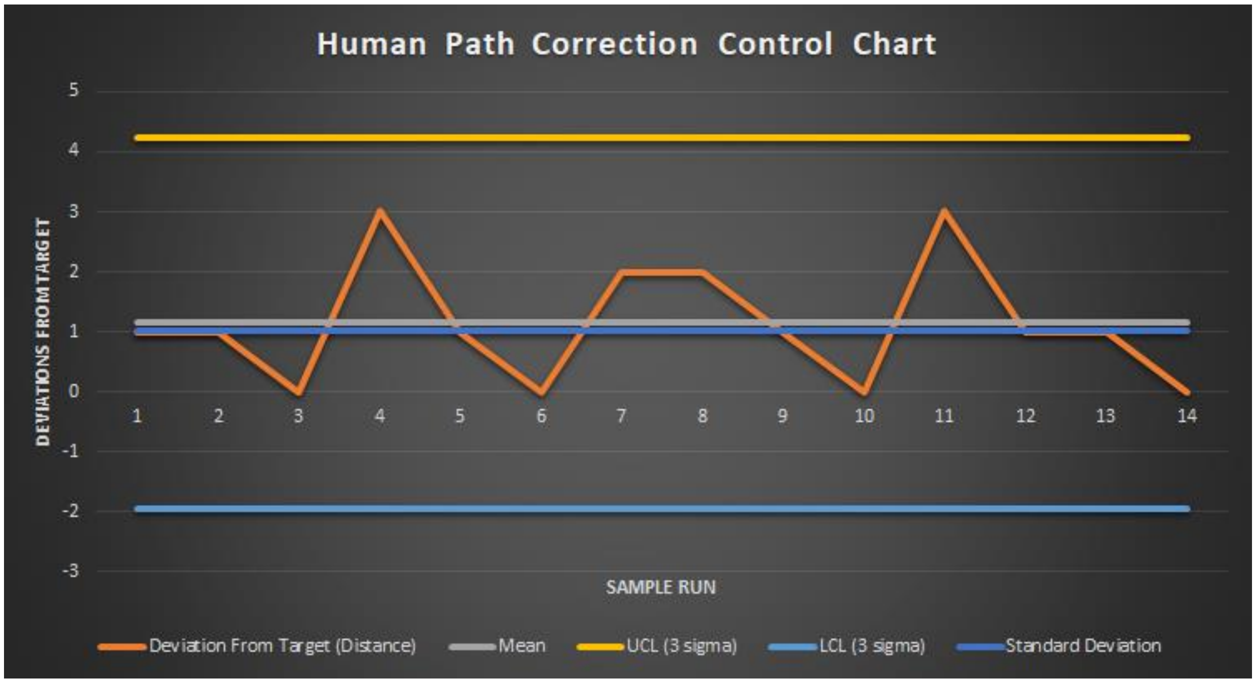


Figure 13: Control Chart for Human Path Correction

Sample Run	Deviation from Target (missed a turn -1, counted extra turn +1)	UCL (3σ)	LCL (3σ)	Mean	Standard Deviation
1	1	4.224172689	-1.938458403	1.1428571428	1.027105182
2	1	4.224172689	-1.938458403	1.1428571428	1.027105182
3	0	4.224172689	-1.938458403	1.1428571428	1.027105182
4	3	4.224172689	-1.938458403	1.1428571428	1.027105182
5	1	4.224172689	-1.938458403	1.1428571428	1.027105182
6	0	4.224172689	-1.938458403	1.1428571428	1.027105182
7	2	4.224172689	-1.938458403	1.1428571428	1.027105182
8	2	4.224172689	-1.938458403	1.1428571428	1.027105182
9	1	4.224172689	-1.938458403	1.1428571428	1.027105182
10	0	4.224172689	-1.938458403	1.1428571428	1.027105182
11	3	4.224172689	-1.938458403	1.1428571428	1.027105182
12	1	4.224172689	-1.938458403	1.1428571428	1.027105182
13	1	4.224172689	-1.938458403	1.1428571428	1.027105182
14	0	4.224172689	-1.938458403	1.1428571428	1.027105182

Table 4: Data for LCD Print

Appendix

Provide supporting material for your proposal here. It may be:

A1: Component Cost Reference:

- Robot framework x 1 (Comes with kit)
- Arduino nano x 1 (Comes with kit)
- IR Optical Sensors x2 (Comes with kit) + x2 (bought)
- Ultrasonic Sensor x 1 [https://www.sparkfun.com/products/15569] ($3.95)
- L9110S Dual H-Bridge motor driver x 1 (Comes with kit)
- Motor drive board x 2 (Comes with kit)
- Wires (Comes with kit)
- Switch x 1 (Comes with kit)
- Wheels x 2 (Comes with kit)
- LCD Screen x 1 (Pre-Owned)
- Battery Holder x 1 (Comes with kit)

A2: Total Cost:

$3.95

A3: Datasheets:

- Arduino nano x 1 [https://culearn.carleton.ca/moodle/pluginfile.php/3697721/mod_resource/content/1/ATmega168Data-Sheet.pdf]
- IR Optical Sensors x 2 (Comes with kit)
- Ultrasonic Sensor x 1 [https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf]
- L9110S Dual H-Bridge motor driver x 1 [https://culearn.carleton.ca/moodle/pluginfile.php/3697718/mod_resource/content/1/Motor_control_driver_chip-l9110.pdf]

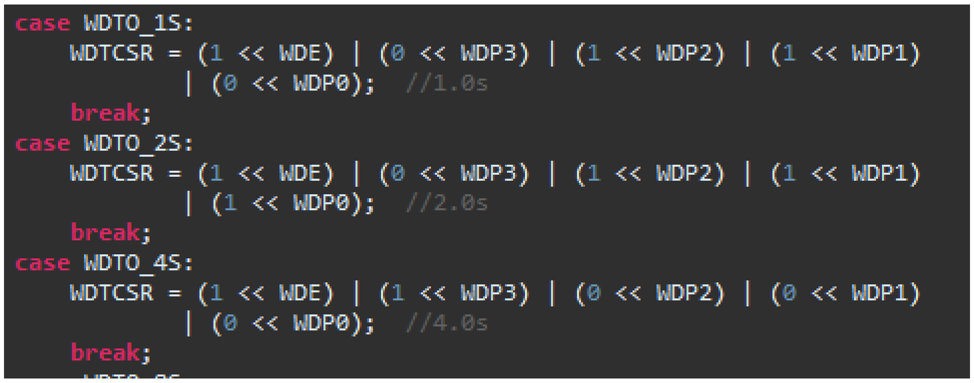
A4: Research materials:

1. This resource provides us with an example on how to interface with the ultrasonic sensor and how the physical component works providing us with a better understanding of how to use the component within the project.
9. https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
2. Below are few possible references that we can use as possible solutions for the maze solving algorithms that can be implemented into the robot.

- https://www.cs.bu.edu/teaching/alg/maze/
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.4944&rep=rep1&type=pdf
- https://www.researchgate.net/publication/331481380_A_Review_of_Various_Maze_Solving_Algorithms_Based_on_Graph_Theory

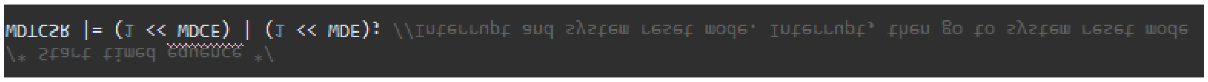
3. For testing we will use "ArduinoUnit" Eclipse IDE to develop Unit test for Arduino Hardware functionality and code functionality, Bellow is the documentation,
   - **ArduinoUnit**: https://github.com/mmurdoch/arduinounit
   - Example: https://maker.pro/arduino/tutorial/using-unit-test-frameworks-with-arduino
4. Watchdog Timer: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
5. Battery Sense: https://github.com/rlogiacco/BatterySense

A5: CODE snap



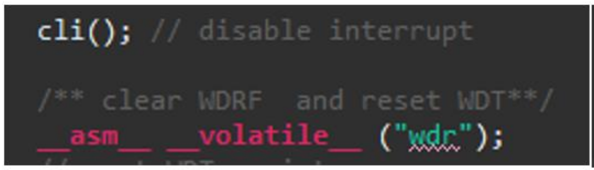
```
case WDTO_1S:
    WDTCSR = (1 << WDE) | (0 << WDP3) | (1 << WDP2) | (1 << WDP1)
            | (0 << WDP0);  //1.0s
    break;
case WDTO_2S:
    WDTCSR = (1 << WDE) | (0 << WDP3) | (1 << WDP2) | (1 << WDP1)
            | (1 << WDP0);  //2.0s
    break;
case WDTO_4S:
    WDTCSR = (1 << WDE) | (1 << WDP3) | (0 << WDP2) | (0 << WDP1)
            | (0 << WDP0);  //4.0s
    break;
```

Code: 1: WDT Pre-scaler set up

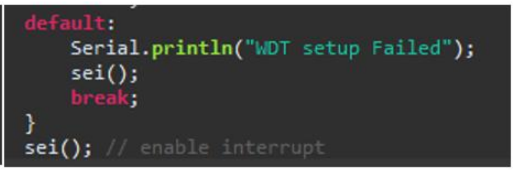


```
WDTCSR |= (1 << WDCE) | (1 << WDE); //interrupt and system reset mode. Interrupt, then go to system reset mode
/* start timed sequence */
```

Code: 2: WDT interrupt and system reset mode.

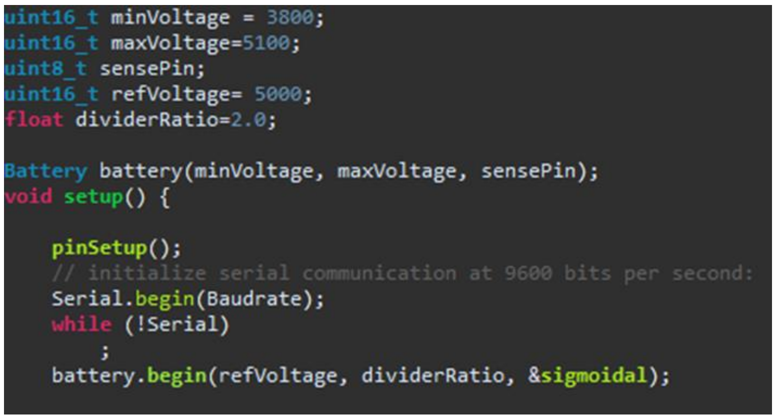


```
cli(); // disable interrupt

/** clear WDRF  and reset WDT**/
__asm__ __volatile__ ("wdr");
```



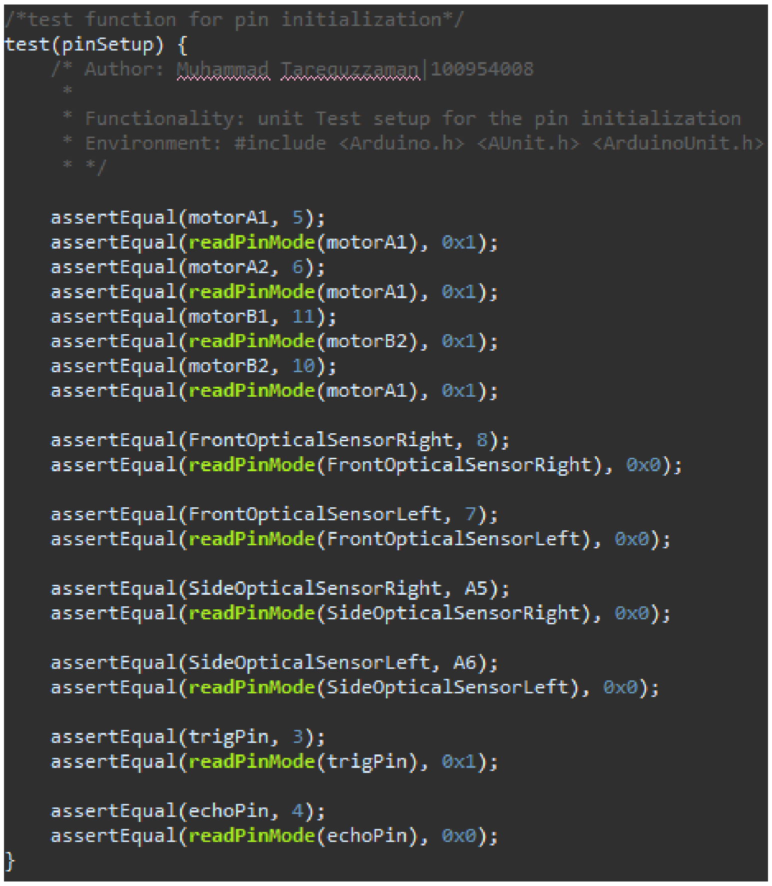
```
default:
    Serial.println("WDT setup Failed");
    sei();
    break;
}
sei(); // enable interrupt
```

Code: 3: cli and sli setup inside the WDT setup



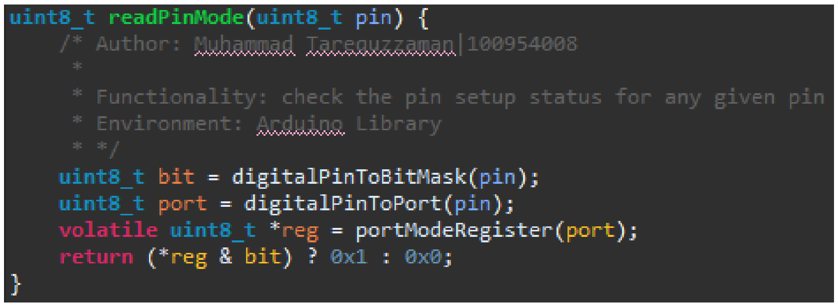
```
uint16_t minVoltage = 3800;
uint16_t maxVoltage=5100;
uint8_t sensePin;
uint16_t refVoltage= 5000;
float dividerRatio=2.0;

Battery battery(minVoltage, maxVoltage, sensePin);
void setup() {

    pinSetup();
    // initialize serial communication at 9600 bits per second:
    Serial.begin(Baudrate);
    while (!Serial)
        ;
    battery.begin(refVoltage, dividerRatio, &sigmoidal);
```

Code: 4: Battery Sense setup



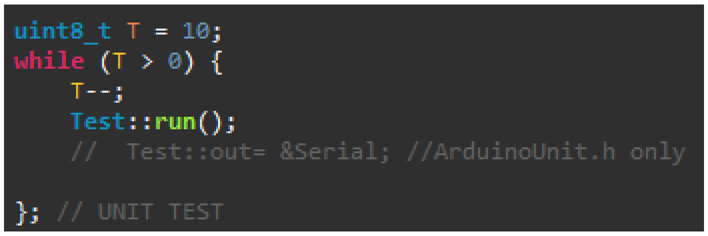
```
/*test function for pin initialization*/
test(pinSetup) {
    /* Author: Muhammad Tarequzzaman|100954008
     *
     * Functionality: unit Test setup for the pin initialization
     * Environment: #include <Arduino.h> <AUnit.h> <ArduinoUnit.h>
     * */

    assertEqual(motorA1, 5);
    assertEqual(readPinMode(motorA1), 0x1);
    assertEqual(motorA2, 6);
    assertEqual(readPinMode(motorA1), 0x1);
    assertEqual(motorB1, 11);
    assertEqual(readPinMode(motorB2), 0x1);
    assertEqual(motorB2, 10);
    assertEqual(readPinMode(motorA1), 0x1);

    assertEqual(FrontOpticalSensorRight, 8);
    assertEqual(readPinMode(FrontOpticalSensorRight), 0x0);

    assertEqual(FrontOpticalSensorLeft, 7);
    assertEqual(readPinMode(FrontOpticalSensorLeft), 0x0);

    assertEqual(SideOpticalSensorRight, A5);
    assertEqual(readPinMode(SideOpticalSensorRight), 0x0);

    assertEqual(SideOpticalSensorLeft, A6);
    assertEqual(readPinMode(SideOpticalSensorLeft), 0x0);

    assertEqual(trigPin, 3);
    assertEqual(readPinMode(trigPin), 0x1);

    assertEqual(echoPin, 4);
    assertEqual(readPinMode(echoPin), 0x0);
}
```

Code: 5: Test pinsetup assertation



```
uint8_t readPinMode(uint8_t pin) {
    /* Author: Muhammad Tarequzzaman|100954008
     *
     * Functionality: check the pin setup status for any given pin
     * Environment: Arduino Library
     * */
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *reg = portModeRegister(port);
    return (*reg & bit) ? 0x1 : 0x0;
}
```

Code: 6: readPinmode code snap



```
uint8_t T = 10;
while (T > 0) {
    T--;
    Test::run();
    //  Test::out= &Serial; //ArduinoUnit.h only

}; // UNIT TEST
```

Code: 7: TEST Run setup inside the main setup function.