Mechatronics 2

Assessment 5

By Jeong Bin Lee – 12935084

E: 12935084@student.uts.edu.au

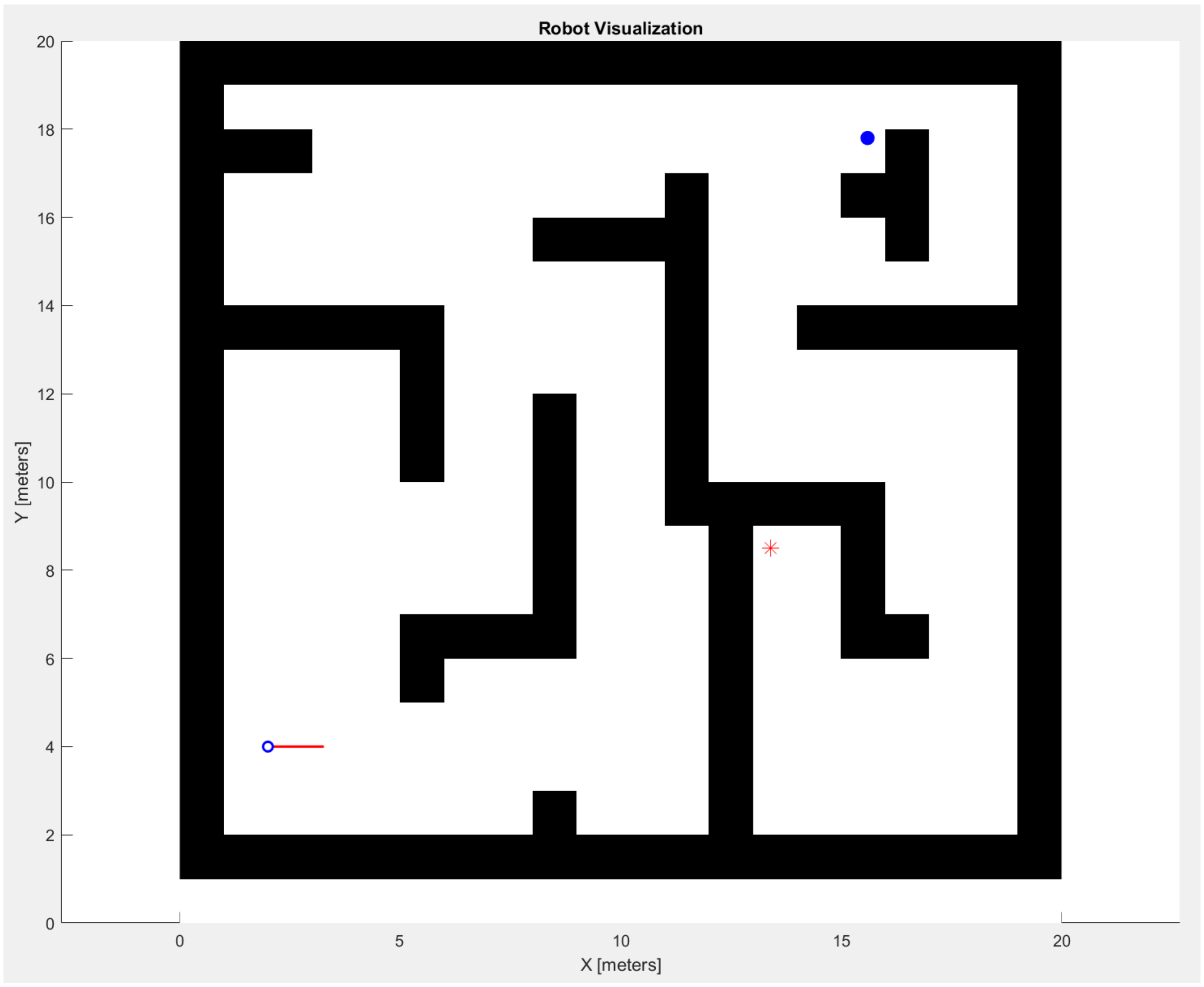


Table of Contents

1. Introduction

The aim of the task is to find two goals, one is a bucket of water and the other is fire. The bucket water must be collected before fire can be put out. After the fire is put out, the robot must return back to the starting position. There are some limitations, such as the robot must be at least within 0.5m of the goal itself which goes for both water and fire, in any circumstances the robot must not collide with the wall and after returning home the robot must be within 2m of the starting position. The assessment also encourages the robot to finish this task as fast as possible by competing with other robots.

The robot can move forward and backwards and turn clockwise and counter-clockwise as commanded by the programmer with some level of noise. In addition to this, there is an IR sensor that can detect and give back the distance of walls in a 360° and a ping that can give the distance of the goal but not the direction. Using this as a tool to get around obstacles to find both goals as fast as possible without colliding with walls, requires some level of knowledge in programming, mathematics, and logical thinking.

2. System Overview

2.1 System Flowchart

When the user presses the select button on the LCD shield, the program will start by sending the start command. Then the robot will begin to move to the first set node on the map. After navigating to the node, the robot will then ping for any nearby goals within the set radius of the node. If there is a goal nearby, it will approach it to 2m until the goal can be identified. If that goal is fire it will store the current node number into a variable to come back to it, if that node is water, it will collect it and move on to the next node until fire goal is found. After all the goals have been achieved, the robot will return to the starting position by backtracking its original path, however not visiting all the nodes, or pinging for any goals. After the goal is within 2m of its original position, the close command will be sent, that the robot has finished its primary objectives.

The flowchart of this entire systems is given in figure 2.1.

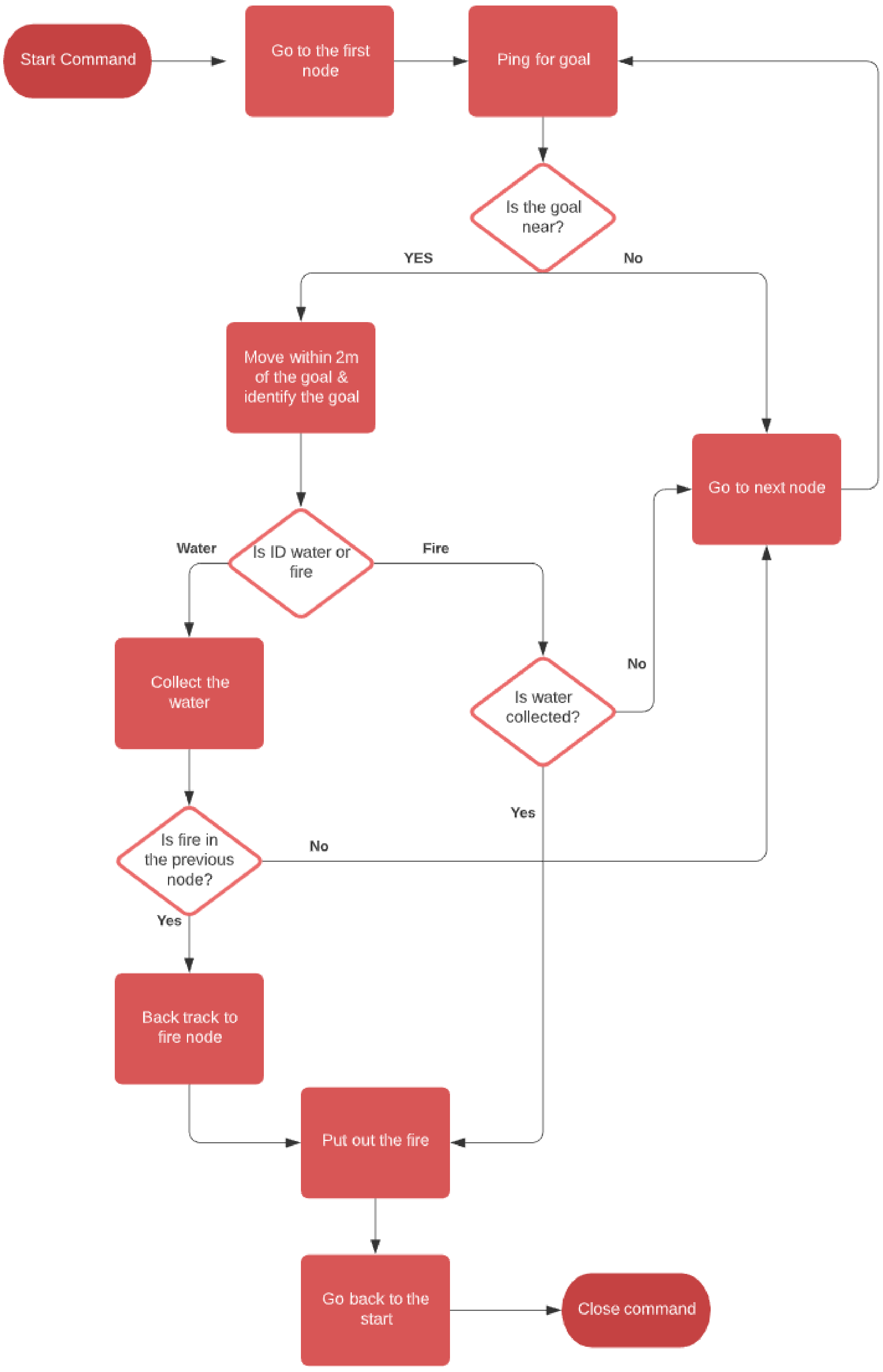


Figure 2.1. System Flowchart from start to the end.

2.2 Implemented Functionalities

There are 8 major functions implemented to achieve the goals:

1. **Movement functions** are to move forward and backward with precision having an error of 0.002m.

2. **Turning functions** are to rotate the angle the robot is facing with precision having an error of less than 0.001°.

3. **Scanning for goals** is to ping the goal and identify if the goal is within the specified radius.

4. **Find water** functionality is to align the robot horizontally and vertically, to the goal until the goal is reached.

5. **Back to node** functionality is to return the robot to the node, so the current path can be continued.

6. **Find fire** is only used after finding water, which also considers if the fire is in one of the previously visited nodes, or if the robot should continue the current path.

7. **Move path** is for the robot to follow a path from the starting point, which is row 4, column 2 to the end point, which is row 7, column 14 and every other node in between. Each time the robot reaches a node, goal is pinged to check for any goals nearby within a radius.

8. **Take me home** is the functionality to directly go back the path the robot came from. This function is used to return to the starting point and also to go back to the identified fire node.

In addition to the functionality mentioned above, the original plan to move around was to use a modified version of A Star Algorithm which worked on a separate .c file, however the RAM in Arduino Uno is only 2kB, which wasn't enough to host the 2 different 2D array. The code worked for a short distance, but long distance required additional attention to the logic of the programming. But it did not matter as the hardware could not support the software. Figure 4.3.1 shows an example of the described modified A Star Algorithm where the right shows the numbers indicating which direction to go to approach the goal, and left shows the actual path the robot would have taken.

3. Hardware Overview

3.1 Hardware Design Proposal

The proposed hardware is given in table 3.1.

Image	Component	Quantity	Use	Link
	Wheels	4	The wheels will be attached to the DC stepper motor to move the robot around.	https://www.thingiverse.com/thing:862438
	DC stepper motor	5	The stepper motors are used to accurately move around the robot. One motor will be used to move the IR sensor 360°	https://core-electronics.com.au/small-reduction-stepper-motor-5vdc-512-step.html?utm_source=google_shopping&gclid=CjwKCAjw8-78BRA0EiwAFUw8LLz9A5AdEwEh78l8XRdloGx3N9Uf4qRc89eF3rspkurmIks-PV2pDBoCWhgQAvD_BwE
	IR Sensor	1	The IR sensor will be used to detect the wall.	https://www.robotgear.com.au/Product.aspx/Details/272-Sharp-Analog-IR-Distance-Sensor-GP2Y0A02YK0F-20cm-150cm
	Thermal Sensor	1	The thermal sensor will be used to detect the fire goal and the water goal depending on the change of temperature.	https://au.element14.com/omron-electronic-components/d6t-8l-09h/thermal-ir-sensor-5-200deg-c-i2c/dp/3296706?gclid=CjwKCAjw0On8BRAgEiwAincsHKRqeEekSezGc46SJUngvhkk_auGjj4eLR4dAgkaZDgUua1o6mnyxhoClMoQAvD_BwE&mckv=s_dc|pcrid|432287894076|pkw||pmt||slid||product|3296706|pgrid|102205443804|ptaid|aud-471864486199:pla-900374783411|&CMP=KNC-GAU-GEN-SHOPPING-SENSORS-AND-TRANSDUCERS
	Arduino Uno	1	The Arduino Uno is the brains of the components that brings everything together.	https://core-electronics.com.au/uno-r3.html?utm_source=google_shopping&gclid=CjwKCAjw8-78BRA0EiwAFUw8LPL1XNBDgc9IspY4kKc2iv7ZfhXxM2JjM1n__yh230bweWzxMnysrxoCwt4QAvD_BwE
	9V batter pack	1	The battery pack powers the entire module and allows Arduino to operate as a portable device	https://core-electronics.com.au/9v-battery-holder-with-switch-5-5mm-2-1mm-plug.html?utm_source=google_shopping&gclid=CjwKCAjw8-78BRA0EiwAFUw8LGhYyuxkEt4SX1HmQLAbdriH8s5hi-yaWuFlDvdczfgmy2KftBBqfhoCP8kQAvD_BwE

Table 3.1. Hardware Proposal.

3.2 Functionality

The functionality of the robot will work same as the simulated version on MATLAB. It will have 4 wheels that can move forwards and backwards, to turn the robot in place, the two left wheels will move in one direction while the two right wheels will move in the other direction. There will be a motor attached to the IR and thermal sensor for a full 360° view for the sensor to scan. A 9v battery will allow the Arduino Uno to be portable, and all the components will be attached to a metal sheet to be protected by fire.

4. System Components

4.1 Calibration

Before diving into the project, since there was a lot of noise and uncertainty that the robot will respond as directed, some experiments and calibration was done on excel. Figure 4.1.1 shows the calibration for the motor movement. Passing a command of 1m from Arduino to MATLAB gave a distance difference of 1.04m and it was fairly consistent from 0.5m to 5m. Additional data gathered is shown in Figure 4.1.2. Using all the gathered data, the function "void moveUp(void)" and "void moveDown(void)" was obtained. Although the movement was limited from 0.5m to 5m, it outweighed the amount of error produced by noise and anything else. After a while, 0.5m was not small enough, 0.25m was added in addition.

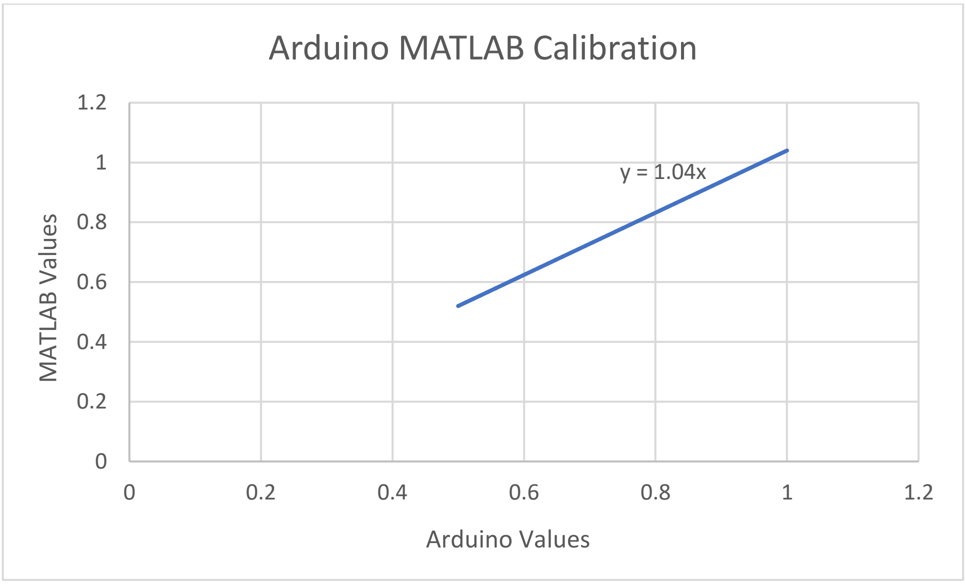


Figure 4.1.1 Arduino to MATLAB movement calibration.

Arduino Distance	Actual distance	Changed Arduino Dist	error correction 1	error correction 2	Error correction 3
0.5	0.52	0.499	0.001	0.001	0.001
1	1.04	0.998	0.002	0.002	0.002
1.5	1.56	1.498	0.002	0.002	0.002
2	2.04	1.997	0.003	-0.007	0.001
2.5	2.55	2.448	0.052	0.001	0.001
3	3.06	2.938	0.062	0.001	0.001
3.5	3.57	3.437	0.063	0.001	0.001
4	4.08	3.927	0.073	0.002	0.002
4.5	4.59	4.417	0.083	0.002	0.002
5	5.1	4.906	0.094	0.002	0.002

Figure 4.1.2 Additional data gathered for calibration.

After increasing the accuracy of the movement, the angle needed correction as it also had noise which impacted the movement of the robot. However, same strategy could not be used for the angle. Instead, the angle was changed in small increments of 0.5°. For example, to turn 90°, a for loop will be prompted 180 times, with each iteration printing a command to turn 0.5°. Turning 90° using this method gave little to no error. Sometimes, turning clockwise and then counter-clockwise allowed the robot to correct itself.

4.2 Maps

The next phase of this project was to implement a 2D array map accurate enough for the robot to move around freely without any collision with walls. The map was initially drawn on excel spreadsheet on a 20x20 shown in figure 4.2.1, where each square would represent 1m of the map. Soon after, the map was changed to 40x40 to ensure that the map was able to cover 0.5m movements. The figure in 4.2.1 also shows an upside-down version of the map. This was due to the array taking the lowest values in the array first, hence why the figure 4.2.2 is also upside-down. A map of 60x60 was also created to increase the accuracy, however, was not included in the end. In fact, the Arduino Uno did not have enough RAM to use more than 1 2D array of 20x20 map (see section 4.3 for more information).

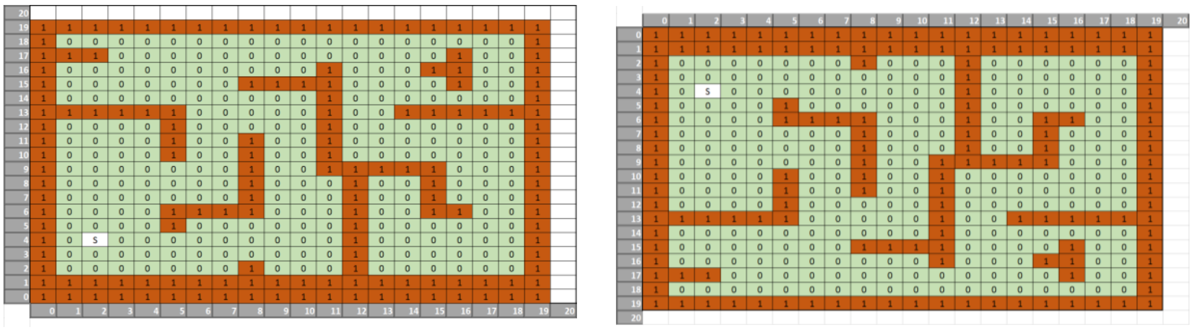


Figure 4.2.1 20x20 representation of the map.

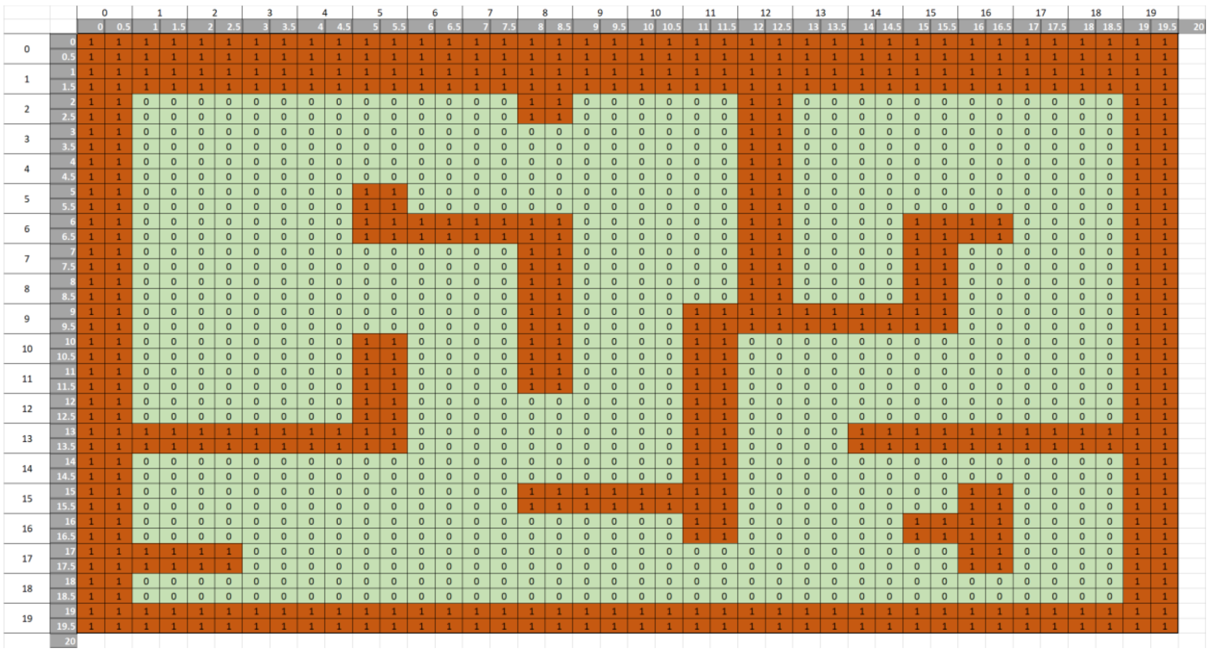


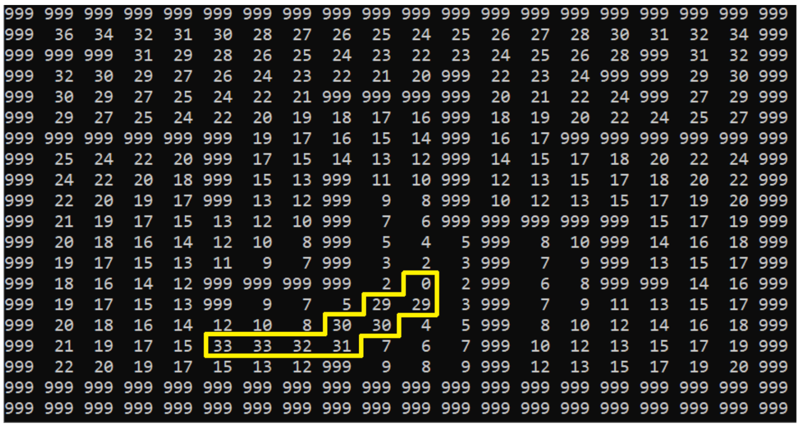
Figure 4.2.2 40x40 representation of the map.

4.3 A Star Algorithm

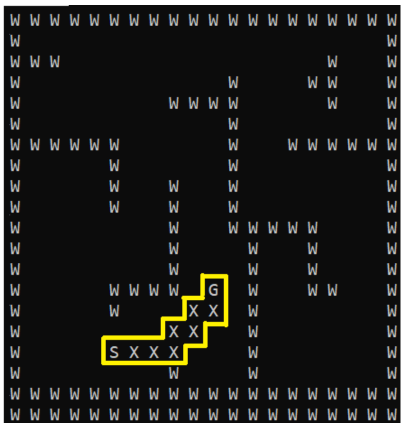
As mentioned in section 2.2, a modified version of A Star Algorithm was developed and tested briefly however the hardware could not handle the amount of data. After running the algorithm with specified nodes, the first few nodes were reached without any problem, and showed promising result. During the 5th and 6th node, the angle at which the robot was instructed to turn was more than what was programmed, and MATLAB was constantly stopping due to corrupted commands. Investigating this problem revealed that some of the variables like angle to turn was getting overwritten in the RAM. Hence A Star Algorithm could not be utilised due to the size of the map.

Figure 4.3.1 shows a simple example of the modified A Star Algorithm implemented. The left image shows characters, where 'S' is the start, 'X' is the path the robot took, 'G' is the goal to be reached, and 'W' shows the walls. The image on the right shows lot of numbers that represent the distance and weight of moving to the goal in each square, where 0 is the goal and the numbers start to increase around it. The number 999 represents the wall as it is such a large number, the robot will never be expected to approach them. One last thing to mention for the image on the right is that the path the robot took left a trail of larger number so that the robot will be less inclined to go back to the path it came from. This map was refreshed for every node.

For small distances that were around the corner, this method worked exceptionally, however, for long distances and goals on the other side of the walls, this method needed additional work.



```
999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
999  36  34  32  31  30  28  27  26  25  24  25  26  27  28  30  31  32  34 999
999 999 999  31  29  28  26  25  24  23  22  23  24  25  26  28 999  31  32 999
999  32  30  29  27  26  24  23  22  21  20 999  22  23  24 999 999  29  30 999
999  30  29  27  25  24  22  21 999 999 999 999  20  21  22  24 999  27  29 999
999  29  27  25  24  22  20  19  18  17  16 999  18  19  20  22  24  25  27 999
999 999 999 999 999 999  19  17  16  15  14 999  16  17 999 999 999 999 999 999
999  25  24  22  20 999  17  15  14  13  12 999  14  15  17  18  20  22  24 999
999  24  22  20  18 999  15  13 999  11  10 999  12  13  15  17  18  20  22 999
999  22  20  19  17 999  13  12 999   9   8 999  10  12  13  15  17  19  20 999
999  21  19  17  15  13  12  10 999   7   6 999 999 999 999 999  15  17  19 999
999  20  18  16  14  12  10   8 999   5   4   5 999   8  10 999  14  16  18 999
999  19  17  15  13  11   9   7 999   3   2   3 999   7   9 999  13  15  17 999
999  18  16  14  12 999 999 999 999   2   0   2 999   6   8 999 999  14  16 999
999  19  17  15  13 999   9   7   5  29  29   3 999   7   9  11  13  15  17 999
999  20  18  16  14  12  10   8  30  30   4   5 999   8  10  12  14  16  18 999
999  21  19  17  15  33  33  32  31   7   6   7 999  10  12  13  15  17  19 999
999  22  20  19  17  15  13  12 999   9   8   9 999  12  13  15  17  19  20 999
999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
```



```
W W W W W W W W W W W W W W W W W W W W
W                                     W
W W W                           W     W
W                     W       W W     W
W               W W W W         W     W
W                     W               W
W W W W W W           W     W W W W W W
W         W           W               W
W         W     W     W               W
W         W     W     W               W
W               W     W W W W W       W
W               W       W     W       W
W               W       W     W       W
W         W W W W   G   W     W W     W
W         W       X X   W             W
W               X X     W             W
W         S X X X       W             W
W               W       W             W
W W W W W W W W W W W W W W W W W W W W
W W W W W W W W W W W W W W W W W W W W
```

Figure 4.3.1 Example of tested A Star Algorithm.

4.4 Path for Navigation

Navigation was designed on the 40x40 map shown in Figure 4.4.1, where 'S' represents the starting point of the robot, it will then go to the node '#0', then to node '#1' and so on until a goal is found. If fire is found before water is reached, the node will be saved into a variable to come back to after finding the water. If the robot finds water first, it will then continue the path to find fire. Each node has a different radius at which to scan the map for the goals, which is defined by the black outlines in figure 4.4.1. Using the precise movement and change in angle as described in section 4.1 each node was reached with great accuracy by giving specific commands to the robot.

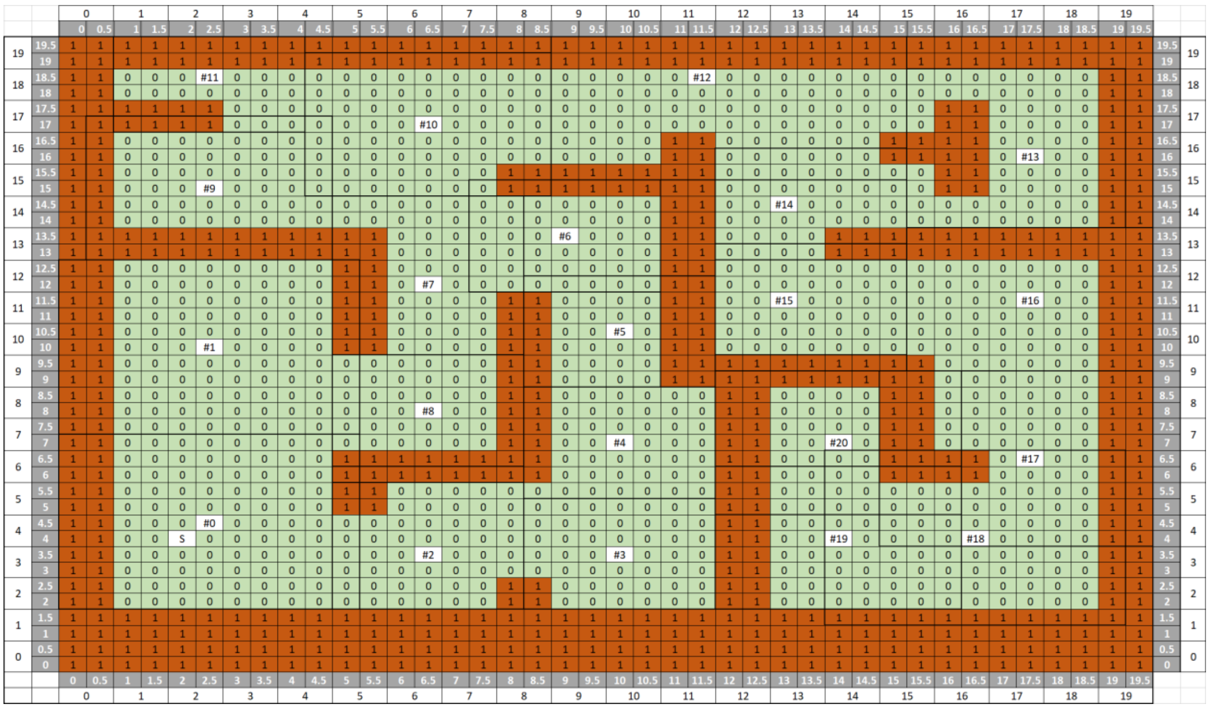


Figure 4.4.1 Path for navigation (not upside-down).

4.5 Goal Finding

In assessment 3, inverse cosine rule was used to calculate the angle required to turn to achieve the goal. Although this method was efficient and time saving, it was also unreliable as it would often give an invalid angle which would stop the MATLAB simulation. Thus, a new method was to be implemented for this project.

Each node visited scans for near by goals with a specified radius, and if it returns a number other than 0, the goal approaching method will be initiated. How it works is, first the robot will move forward by 0.25m, and scan the goal again, but this time saving the value in a different variable to compare if the goal is closer or further away. If the goal is closer, it will continue to move forward, but, if the goal is further away, it will move backwards and scan the goal again until the row or column is aligned, then turn 90 deg and do it again until both the row and column is aligned. If both is aligned, the goal has been achieved and the robot will go back to the node and face the direction as it normally would and either continue the path to find the next goal or return to the node that was tagged.

4.6 Return Home

To return to the start, the program uses the inverse of the navigation. Meaning that the robot trails back the exact path it came from, but with one exception that it looks for the shortest path. For example, if the robot it at node 14 in figure 4.4.1, and all the goals have been achieved, it would skip node 13 and move straight to node 12. But if fire goal is not yet achieved, the robot would go back to the fire node using the shortest distance method. When the robot finally gets all the goals and reaches the starting point, the close command is sent to MATLAB and timer that initially started with the start command will stop.

4.7 Difficulties

Initial difficulty was to get the Arduino to run with the modified A Star Algorithm as outlined section 4.3. This problem was never solved hence why the entire approach to the project was changed to using node points on the map. To implement the node point on the map, several strategies were used. First one was scanning the entire 5m radius, which caused problems when the goal was behind an obstacle. This problem was fixed by creating a function that takes a radius as an input; if the goal was outside the radius, it would return 0 as irrelevant. Another difficulty in this project was to account for finding fire goal before the water goal. This was solved as mention above, by storing the node number into a variable to come back to after finding the water. The last difficulty was when the robot reaches the final node, but does not have all the goals, the robot stops while the counter still ticks. This problem could not be fixed, and hope to achieve all the goal before or at the final node.

5. Appendix

Arduino Complete Code

```
#include <LiquidCrystal.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>

//Author: Jeong Bin Lee
//SID: 12935084
//Date: 30/10/2020
//Status: Finished

/**** C++ ****/
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);


#define START_ROW 4
#define START_COL 2
#define NODE_SIZE 21

/*** ENUM ***/
enum BUTTONS //reads users inputs
{
  NOTHING,
  RIGHT,
  UP,
  DOWN,
  LEFT,
  SELECT
};

/*** GLOBAL VARIABLES ***/
volatile unsigned long int Delay = 0;

uint8_t Button; //enum of user button

uint16_t Adc_Read; //used to read user
button

String goal;

//User Position
float row = START_ROW;
float col = START_COL;
//UP = 0, RIGHT = 90, DOWN = 180, LEFT =
270
int16_t face = 90; //initially 90


uint8_t fire_count = 100; //keeps track on
where the fire node is

static int main_count = 0; //keeps track on
where the current node is

bool water_goal = false; //true if water goal is
achieved, else false
bool fire_goal = false; //true if fire goal is
achieved, else false
bool country_road = false; //true if return
home is enabled, else false
bool timer = false; //true if select is pressed,
else false

//node is the key point the robot visits using
the row, col, face, and radius
//each array shows the important variables
used in this project
float node_row[NODE_SIZE] = {4.5, 10, 3.5,
3.5, 7, 10.5, 13.5, 12, 8, 15, 17, 18.5, 18.5, 16,
14.5, 11.5, 11.5, 6.5, 4, 4, 7};
float node_col[NODE_SIZE] = {2.5, 2.5, 10, 10,
10, 10, 9, 6.5, 6.5, 2.5, 6.5, 2.5, 11.5, 17.5,
13.5, 13, 17.5, 17.5, 16.5, 14, 14};
int node_face[NODE_SIZE] = {0, 0, 90, 90, 0, 0,
270, 0, 0, 270, 0, 90, 90, 180, 270, 180, 90,
180, 270, 270, 0};
float node_radius[NODE_SIZE] = {3.2, 3.7, 2.7,
2.7, 2.7, 2.2, 2.2, 2.7, 2.2, 2.7, 2.7, 2.2, 5.2,
2.4, 2.7, 2.7, 2.7, 3.2, 3.2, 3.2, 3.2};

//used for timing the project
uint32_t totalsec = 0;
uint8_t mins = 0; //for tellimg number of mins
uint8_t secs = 0; //for telling number of secs


//interrupt service routine
ISR(TIMER2_OVF_vect)
{
  static uint32_t counter = 0;
  static uint32_t millisec = 0;
  Delay++; //Global delay counter (1ms)

  //reading buttons
  if(timer == false)
  {
    counter++;
    if(counter >= 100)
    {
      //TEST//
      ReadUserButton();
      counter = 0;
    }
  }
  else if(timer == true)
  {
    millisec++;
    if(millisec >= 980)
    {
      totalsec++;
      secs = (totalsec % 3600) % 60;
      mins = (totalsec % 3600) / 60;
      millisec = 0;
      updateTime();
    }
  }
}

//to initialise delay
void DelayInit() {
  TCCR2A = 0;
  // Prescaler for 64
  TCCR2B &= ~(1<<CS20);
  TCCR2B &= ~(1<<CS21);
  TCCR2B |= (1<<CS22);
  TIMSK2 |= (1<<TOIE2); //Enable Overflow
interrupt
  TIFR2 |= (1<<TOV2);
}

//to get global delay variable
volatile unsigned long int getDelay() {
  return Delay;
}

//my delay fuction
void DelayMilli(uint64_t delayTime) {
  uint64_t count = getDelay();
  while(getDelay() <= (delayTime + count)) {
  }
}

//ADC setup
void ADCSetup(void)
{
  ADCSRA = (1<<ADEN); // Enable ADC
  ADMUX |= (1<<REFS0); // Internal Vcc 5v
}

//read buttons from LCD shield
void ReadUserButton(void)
{
  //read the user input
  if(Adc_Read != ADCsingleREAD(0))
  {
    Adc_Read = ADCsingleREAD(0);
  }
}

//check what the user pressed
void CheckUserButton(void)
{
  if(Adc_Read <= 50) //Right  = 0
  {
    Serial.println("Right");
    Button = RIGHT;
    DelayMilli(200); //debounce
  }
  else if(600 <= Adc_Read && Adc_Read <=
650) //Left   = 626 or 625
  {
    Serial.println("Left");
    Button = LEFT;
    DelayMilli(200); //debounce
  }
  else if(150 <= Adc_Read && Adc_Read <=
350) //Up = 208
  {
    Serial.println("Up");
    Button = UP;
    DelayMilli(200); //debounce
  }
  else if(350 <= Adc_Read && Adc_Read <=
450)//Down = 410 or 409
  {
    Serial.println("Down");
    Button = DOWN;
    DelayMilli(200); //debounce
  }
  else if(750 <= Adc_Read && Adc_Read <=
900)//Select = 824
  {
    Serial.println("Select");
    Button = SELECT;
    DelayMilli(250); //debounce
  }
  else
  {
//    Serial.println("Nothing");
    Button = NOTHING;
//    DelayMilli(150);
  }
}



//read ADC pin once
int ADCsingleREAD(uint8_t pin)
{
  if(pin == 0)
  {
    ADMUX = 0; //Multiplexer  for which pin to
read from
    ADMUX |= (1<<REFS0); // Internal Vcc 5v
  }
  else
  {
    ADMUX = pin;
  }
  ADCSRA |= (1<<ADSC); // start conversion


  // wait for conversion to complete
  while (!(ADCSRA &(1<<ADIF)));

  ADCSRA |= (1<<ADIF);
  return ADC;
}

//send a message from arduino to computer
using Serial
void PrintMessage(String message)
{
 Serial.print(message);
 Serial.write(13); //carriage return character
(ASCII 13, or '\r')
 Serial.write(10); //newline character (ASCII
10, or '\n')
}

//setup before running the main loop
void setup(void)
{
  Serial.begin(9600);
  Serial.setTimeout(350); //set timeout to
300ms
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("12935084");
  lcd.setCursor(0,1);
  lcd.print("00:00");

  //initialise your code here
  DelayInit();
  ADCSetup();

  sei(); // enable interrupts
}

//prints the time on LCD shield
void updateTime(void)
{
  lcd.setCursor(0,1);
  if(mins < 10)
  {
    lcd.print("0");
  }
  lcd.print(mins);

  lcd.print(":");

  if(secs < 10)
  {
    lcd.print("0");
  }
  lcd.print(secs);
}

//the main loop where all the code gets
executed
void loop()
{
  //MAIN LOOP
  DelayMilli(300);


  CheckUserButton();

  //if select button is pressed
  if(Button == SELECT)
  {
    PrintMessage("CMD_START"); // Start the
robot
    DelayMilli(100);
    while(1)
    {
      //moves along the guided path
      if(water_goal == false)
      {
        lcd.setCursor(6,1);
        lcd.print("W"); //finding water
        timer = true; //start timer
        movePath();
      }

      if(goal.toFloat() > 0 && water_goal == false
&& fire_count != main_count - 1) //find water
goal
      {
        findWater(); //move towards the water
goal
        backToNode(); //re-align with the key
point
      }
      else if(fire_count < 100 && water_goal ==
true && fire_goal == false) //back track for
fire goal
      {
        lcd.setCursor(6,1);
        lcd.print("F"); //finding fire
        //move back to the fire node
        takeMeHome();
        if((main_count-1) == fire_count)
        {
          findFire(); //find the fire goal
          backToNode();
        }
      }
      else if(water_goal == true && fire_goal ==
false) //keep going for fire goal
      {
        lcd.setCursor(6,1);
        lcd.print("F"); //finding fire
        //move forward until another goal
appears
        movePath();
        if(goal.toFloat() > 0)
        {
          findFire(); //find the fire goal
          backToNode();
        }
      }
      else if(water_goal == true && fire_goal ==
true) //both goals achieved
      {
        lcd.setCursor(6,1);
        lcd.print("H");
        country_road = true; //return home is
true
        takeMeHome();
        DelayMilli(500);
        if(main_count == 0 || (row <= 5.5 && col
<= 3.5))
        {
          lcd.setCursor(6,1);
          lcd.print("C");
          PrintMessage("CMD_CLOSE"); //close
the bot
          timer = false;
          PrintMessage("CMD_CLOSE"); //close
the bot again (in case)
          timer = false;
          while(1);
        }
      }
    else if((water_goal == false || fire_goal ==
false) && main_count > 20 && fire_goal ==
100)
    {
      while(1) //abort mission
      {
        lcd.setCursor(6,1);
        lcd.print("H");
        country_road = true;
        takeMeHome();
        DelayMilli(500);
        if(main_count == 0 || (row <= 5.5 && col
<= 3.5))
        {
          lcd.setCursor(6,1);
          lcd.print("C");
          PrintMessage("CMD_CLOSE");
          timer = false;
          PrintMessage("CMD_CLOSE");
          timer = false;
          while(1);
        }
      }
    }
   }
  }
}


//move back with extreme accuracy
//takes parameter of float for the distance to
move
void moveUp(float dist)
{
  float tmpDist = dist;

  //experimented for maximum accuracy
movement
  if(dist == 0.25)
  {
    dist = (dist/1.04);
  }
  else if(dist == 0.5)
  {
    dist = (dist/1.04) + 0.001;
  }
  else if(dist == 1)
  {
    dist = (dist/1.04) + 0.002;
  }
  else if(dist == 1.5)
  {
    dist = (dist/1.04) + 0.002;
  }
  else if(dist == 2)
  {
    dist = (dist/1.04) + 0.003;
  }
  else if(dist == 2.5)
  {
    dist = (dist/1.04) + (0.05/1.04);
  }
  else if(dist == 3)
  {
    dist = (dist/1.04) + (0.062/1.04);
  }
  else if(dist == 3.5)
  {
    dist = (dist/1.04) + (0.063/1.04);
  }
  else if(dist == 4)
  {
    dist = (dist/1.04) + (0.073/1.04);
  }
  else if(dist == 4.5)
  {
    dist = (dist/1.04) + (0.083/1.04);
  }
  else if(dist == 5)
  {
    dist = (dist/1.04) + (0.094/1.04);
  }
  else
  {
    return;
  }
  PrintMessage("CMD_ACT_LAT_1_" +
(String)(dist));

  if(face == 0) //UP
  {
    row += tmpDist;
  }
  else if(face == 90) //RIGHT
  {
    col += tmpDist;
  }
  else if(face == 180) //DOWN
  {
    row -= tmpDist;
  }
  else if(face == 270) //LEFT
  {
    col -= tmpDist;
  }

  DelayMilli(400); //delay for the time it take
to move
}

//move back with extreme accuracy
//takes parameter of float for the distance to
move
void moveDown(float dist)
{
  float tmpDist = dist;

  //experimented for maximum accuracy
movement
  if(dist == 0.25)
  {
    dist = (dist/1.04);
  }
  else if(dist == 0.5)
  {
    dist = (dist/1.04) + 0.001;
  }
  else if(dist == 1)
  {
    dist = (dist/1.04) + 0.002;
  }
  else if(dist == 1.5)
  {
    dist = (dist/1.04) + 0.002;
  }
  else if(dist == 2)
  {
    dist = (dist/1.04) + 0.003;
  }
  else if(dist == 2.5)
  {
    dist = (dist/1.04) + (0.05/1.04);
  }
  else if(dist == 3)
  {
    dist = (dist/1.04) + (0.062/1.04);
  }
  else if(dist == 3.5)
  {
    dist = (dist/1.04) + (0.063/1.04);
  }
  else if(dist == 4)
  {
    dist = (dist/1.04) + (0.073/1.04);
  }
  else if(dist == 4.5)
  {
    dist = (dist/1.04) + (0.083/1.04);
  }
  else if(dist == 5)
  {
    dist = (dist/1.04) + (0.094/1.04);
  }
  else
  {
    return;
  }

  PrintMessage("CMD_ACT_LAT_0_" +
(String)(dist));

  if(face == 0) //UP
  {
    row -= tmpDist;
  }
  else if(face == 90) //RIGHT
  {
    col -= tmpDist;
  }
  else if(face == 180) //DOWN
  {
    row += tmpDist;
  }
  else if(face == 270) //LEFT
```

```
 {
  col += tmpDist;
 }

 DelayMilli(400); //delay for the time it take
to move
}

//turn with extreme accuracy CCW
//takes parameter angle for the angle to turn
void turnCCW(int angle)
{
 for(int i=0; i<(angle/0.5); i++)
 {
  PrintMessage("CMD_ACT_ROT_0_0.5");
 }

 if(angle == 90)
 {
  DelayMilli(1000);
 }
 else if(angle == 180)
 {
  DelayMilli(3000);
 }
 else if(angle > 180)
 {
  DelayMilli(4000);
 }

 face -= angle;

 if(face < 0)
 {
  face = 270;
 }
}

//turn with extreme accuracy CW
//takes parameter angle for the angle to turn
void turnCW(int angle)
{
 for(int i=0; i<(angle/0.5); i++)
 {
  PrintMessage("CMD_ACT_ROT_1_0.5");
 }
 if(angle == 90)
 {
  DelayMilli(1000);
 }
 else if(angle == 180)
 {
  DelayMilli(3000);
 }
 else if(angle > 180)
 {
  DelayMilli(4000);
 }

 face += angle;

 if(face > 270)
 {
  face = 0;
 }

}

//scans for any goal near by, ignores the goals
outside the radius
//takes parameter float for the radius to scan
//return 0 if outside the radius, or returns the
goal scanned within the radius
String scanGoal(float radius)
{
 String g;
 PrintMessage("CMD_SEN_PING");
 DelayMilli(200);
 g = Serial.readString();
 DelayMilli(100);
 if(g.toFloat() <= radius)
 {
  return g;
 }
 return "0";
}

//scans for any goal near by
//scans the maximum range (5m)
String scanGoal(void)
{
 String g;
```

```
  PrintMessage("CMD_SEN_PING");
  DelayMilli(100);
  g = Serial.readString();
  DelayMilli(100);

  return g;
}

//approach the water goal until achieved
void findWater(void)
{
  String g1, g2;
  String goalID;

  PrintMessage("CMD_SEN_GOAL");
  DelayMilli(500);
  goalID = Serial.readString();
  DelayMilli(500);
  if(goalID.toFloat() == 1)
  {
    water_goal = true;
    return;
  }

  do
  {
    g1 = scanGoal();
    moveUp(0.25);
    DelayMilli(500);
    g2 = scanGoal();

    if(g1.toFloat() == 0)
    {
      g1 = "100";
    }
    if(g2.toFloat() == 0)
    {
      g2 = "100";
    }

    if(g1.toFloat() > g2.toFloat())
    {
      while(g1.toFloat() > g2.toFloat())
      {
        g1 = g2;
        moveUp(0.25);
        DelayMilli(500);
        g2 = scanGoal();

        //check if goal is achieved
        PrintMessage("CMD_SEN_GOAL");
        DelayMilli(500);
        goalID = Serial.readString();
        DelayMilli(500);
        if(goalID.toFloat() == 1)
        {
          water_goal = true;
          return;
        }

        //check if the goal can be ID
        if(g2.toFloat() < 2.0)
        {
          PrintMessage("CMD_SEN_ID");
          DelayMilli(100);
          String ID = Serial.readString();
          if(ID.toFloat() == 2)
          {
            lcd.print("E1");
            fire_count = main_count - 1; //store
the case in main loop
            return;
          }
        }
        else if(g2.toFloat() == 0)
        {
          break;
        }
      }
      moveDown(0.25);
      DelayMilli(500);
    }
    else if (g2.toFloat() > g1.toFloat())
    {
      moveDown(0.25);
      DelayMilli(500);
      while(g2.toFloat() > g1.toFloat())
      {
        g2 = scanGoal();
        moveDown(0.25);
        DelayMilli(500);
        g1 = scanGoal();
      //check if goal is achieved
      PrintMessage("CMD_SEN_GOAL");
      DelayMilli(500);
      goalID = Serial.readString();
      DelayMilli(500);
      if(goalID.toFloat() == 1)
      {
        water_goal = true;
        return;
      }

      //check if the goal can be ID
      if(g1.toFloat() < 2.0)
      {
        PrintMessage("CMD_SEN_ID");
        String ID = Serial.readString();
        if(ID.toFloat() == 2)
        {
          fire_count = main_count - 1; //store
the case in main loop
          return;
        }
      }
      else if(g1.toFloat() == 0)
      {
        break;
      }
    }
    moveUp(0.25);
    DelayMilli(500);
  }

  //check if goal is achieved
  PrintMessage("CMD_SEN_GOAL");
  DelayMilli(500);
  goalID = Serial.readString();
  DelayMilli(500);
  if(goalID.toFloat() == 1)
  {
    water_goal = true;
    return;
  }

  turnCW(90);
  DelayMilli(2000);
```

```
  g1 = scanGoal();
  moveUp(0.25);
  DelayMilli(500);
  g2 = scanGoal();

  if(g1.toFloat() > g2.toFloat())
  {
    while(g1.toFloat() > g2.toFloat())
    {
      g1 = g2;
      moveUp(0.25);
      DelayMilli(500);
      g2 = scanGoal();

      //check if goal is achieved
      PrintMessage("CMD_SEN_GOAL");
      DelayMilli(500);
      goalID = Serial.readString();
      DelayMilli(500);
      if(goalID.toFloat() == 1)
      {
        water_goal = true;
        return;
      }

      //check if the goal can be ID
      if(g2.toFloat() < 2.0)
      {
        PrintMessage("CMD_SEN_ID");
        String ID = Serial.readString();
        if(ID.toFloat() == 2)
        {
          fire_count = main_count - 1; //store
the case in main loop
          return;
        }
      }
    }
    moveDown(0.25);
    DelayMilli(500);
  }
  else if (g2.toFloat() > g1.toFloat())
  {
    moveDown(0.25);
    DelayMilli(500);
```

```
      while(g2.toFloat() > g1.toFloat())
      {
        g2 = scanGoal();
        moveDown(0.25);
        DelayMilli(500);
        g1 = scanGoal();

        //check if goal is achieved
        PrintMessage("CMD_SEN_GOAL");
        DelayMilli(500);
        goalID = Serial.readString();
        DelayMilli(500);
        if(goalID.toFloat() == 1)
        {
          water_goal = true;
          return;
        }

        //check if the goal can be ID
        if(g1.toFloat() < 2.0)
        {
          PrintMessage("CMD_SEN_ID");
          String ID = Serial.readString();
          if(ID.toFloat() == 2)
          {
            fire_count = main_count - 1; //store
the case in main loop
            return;
          }
        }
      }
      moveUp(0.25);
    }

    //check if goal is achieved
    PrintMessage("CMD_SEN_GOAL");
    DelayMilli(500);
    goalID = Serial.readString();
    DelayMilli(500);
    if(goalID.toFloat() == 1)
    {
      water_goal = true;
      return;
    }
  } while(goalID.toFloat() != 1);
}
```

```
//takes the robot back to the current node
void backToNode(void)
{
  float minDist = 100, dist;
  float r, c;

  //go back to the node
  if(main_count <= 0)
  {
    r = node_row[0];
    c = node_col[0];
  }
  else
  {
    r = node_row[main_count - 1];
    c = node_col[main_count - 1];
  }

  //face UP
//  turnCCW(face);
  while(r != row || c != col)
  {
    if(face == 0)
    {
      while(r != row)
      {
        if(r > row)
        {
          moveUp(0.25);
        }
        else if(r < row)
        {
          moveDown(0.25);
        }
        DelayMilli(200);
      }
    }
    else if(face == 180)
    {
      while(r != row)
      {
        if(r > row)
        {
          moveDown(0.25);
      }
      else if(r < row)
      {
        moveUp(0.25);
      }
      DelayMilli(200);
    }
  }
  //face RIGHT
//  turnCW(abs(90 - face));
  else if(face == 90)
  {
    while(c != col)
    {
      if(c > col)
      {
        moveUp(0.25);
      }
      else if(c < col)
      {
        moveDown(0.25);
      }
      DelayMilli(200);
    }
  }
  else if(face == 270)
  {
    while(c != col)
    {
      if(c > col)
      {
        moveDown(0.25);
      }
      else if(c < col)
      {
        moveUp(0.25);
      }
      DelayMilli(200);
    }
  }
  //turn 90 and do it again if row or col
doesnt match
  if(r != row || c != col)
  {
    turnCW(90);
  }
```

```
  }

  //face the correct direction
  if((node_face[main_count - 1] - face) > 0)
  {
    turnCW(abs(node_face[main_count - 1] -
face));
    DelayMilli(2000);
  }
  else if((node_face[main_count - 1] - face) <
0)
  {
    turnCCW(abs(node_face[main_count - 1] -
face));
    DelayMilli(2000);
  }

  //find the distance to each node
  //find the smallest distance for the nodes
  //find the difference in row && move 0.5m
closer until row is reached
  //find the difference in col && move 0.5m
closer until col is reached
}


//approach the fire goal until achieved
void findFire(void)
{
  //if the current node is equal to fire node
  String g1, g2;
  String goalID;

  //check if goal is achieved
  PrintMessage("CMD_SEN_GOAL");
  DelayMilli(500);
  goalID = Serial.readString();
  DelayMilli(500);
  if(goalID.toFloat() == 2)
  {
    fire_goal = true;
    return;
  }

  do
  {
```

```
g1 = scanGoal();
moveUp(0.25);
DelayMilli(500);
g2 = scanGoal();

if(g1.toFloat() == 0)
{
  g1 = "100";
}
if(g2.toFloat() == 0)
{
  g2 = "100";
}

if(g1.toFloat() > g2.toFloat())
{
  while(g1.toFloat() > g2.toFloat())
  {
    g1 = g2;
    moveUp(0.25);
    DelayMilli(500);
    g2 = scanGoal();
    //check if goal is achieved
    PrintMessage("CMD_SEN_GOAL");
    DelayMilli(500);
    goalID = Serial.readString();
    DelayMilli(500);
    if(goalID.toFloat() == 2)
    {
      fire_goal = true;
      return;
    }
  }
  moveDown(0.25);
}
else if (g2.toFloat() > g1.toFloat())
{
  moveDown(0.25);
  while(g2.toFloat() > g1.toFloat())
  {
    g2 = scanGoal();
    moveDown(0.25);
    DelayMilli(500);
    g1 = scanGoal();
    //check if goal is achieved
    PrintMessage("CMD_SEN_GOAL");
    DelayMilli(500);
    goalID = Serial.readString();
    DelayMilli(500);
    if(goalID.toFloat() == 2)
    {
      fire_goal = true;
      return;
    }
  }
  moveUp(0.25);
  DelayMilli(500);
}

//check if goal is achieved
PrintMessage("CMD_SEN_GOAL");
DelayMilli(500);
goalID = Serial.readString();
DelayMilli(500);
if(goalID.toFloat() == 2)
{
  fire_goal = true;
  return;
}

turnCW(90);
DelayMilli(2000);

g1 = scanGoal();
moveUp(0.25);
DelayMilli(500);
g2 = scanGoal();

if(g1.toFloat() > g2.toFloat())
{
  while(g1.toFloat() > g2.toFloat())
  {
    g1 = g2;
    moveUp(0.25);
    DelayMilli(500);
    g2 = scanGoal();
    //check if goal is achieved
    PrintMessage("CMD_SEN_GOAL");
    DelayMilli(500);
    goalID = Serial.readString();
    DelayMilli(500);
    if(goalID.toFloat() == 2)
```

```
      {
        fire_goal = true;
        return;
      }
    }
    moveDown(0.25);
    DelayMilli(500);
  }
  else if (g2.toFloat() > g1.toFloat())
  {
    moveDown(0.25);
    DelayMilli(500);
    while(g2.toFloat() > g1.toFloat())
    {
      g2 = scanGoal();
      moveDown(0.25);
      DelayMilli(500);
      g1 = scanGoal();
      //check if goal is achieved
      PrintMessage("CMD_SEN_GOAL");
      DelayMilli(500);
      goalID = Serial.readString();
      DelayMilli(500);
      if(goalID.toFloat() == 2)
      {
        fire_goal = true;
        return;
      }
    }
    moveUp(0.25);
    DelayMilli(500);
  }

  //check if goal is achieved
  PrintMessage("CMD_SEN_GOAL");
  DelayMilli(500);
  goalID = Serial.readString();
  DelayMilli(500);
  if(goalID.toFloat() == 2)
  {
    fire_goal = true;
    return;
  }
 } while(goalID.toFloat() != 2);
}
```

```
//move through the main path and scan for goals
void movePath(void)
{
 switch(main_count)
 {
   DelayMilli(500);
   case 0:
     moveUp(0.5); //1
     turnCCW(90); //2
     moveUp(0.5); //3
     //4
     break;
   case 1:
     moveUp(1.5); //5
     moveUp(4); //6
     //7
     break;
   case 2:
     moveDown(1.5); //8
     moveDown(5); //9
     turnCW(90); //10
     moveUp(4); //11
     //12
     break;
   case 3:
     moveUp(3.5); //13
     DelayMilli(500);
     //14
     break;
   case 4:
     turnCCW(90); //15
     moveUp(3.5); //16
     //17
     break;
   case 5:
     moveUp(3.5); //18
     //19
     break;
   case 6:
     moveUp(3); //20
     turnCCW(90); //21
     moveUp(1); //22
     //23
     break;
case 7:
  moveUp(2.5); //24
  turnCW(90); //25
  moveDown(1.5); //26
  //27
  break;
case 8:
  moveDown(4); //28
  //29
  break;
case 9:
  moveUp(5); //30
  moveUp(2); //31
  turnCCW(90); //32
  moveUp(4); //33
  //34
  break;
case 10:
  moveDown(4); //35
  turnCW(90); //36
  moveUp(2); //37
  //38
  break;
case 11:
  moveUp(1.5); //39
  turnCW(90); //40
  moveDown(4); //41
  //42
  break;
case 12:
  moveUp(4.5); //43
  moveUp(4.5); //44
  //45
  break;
case 13:
  moveUp(5); //46
  moveUp(1); //47
  turnCW(90); //48
  moveUp(2.5); //49
  //50
  break;
case 14:
  moveUp(1.5); //51
  turnCW(90); //52
  moveUp(4.5); //53
  //54
  break;
 case 15:
  turnCCW(90); //55
  moveUp(3); //56
  //57
  break;
 case 16:
  turnCCW(90); //58
  moveUp(4.5); //59
  //60
  break;
 case 17:
  turnCW(90); //61
  moveUp(5); //62
  //63
  break;
 case 18:
  moveUp(2.5); //64
  turnCW(90); //65
  moveUp(1); //66
  //67
  break;
 case 19:
  moveUp(2.5); //68
  //69
  break;
 case 20:
  turnCW(90); //70
  moveUp(3); //71
  //72
  break;
}

if(main_count > 20)
{
 main_count = 20;
}

DelayMilli(1000);
goal = scanGoal(node_radius[main_count]);
if(goal.toFloat() == 0)
{
 DelayMilli(250);
 goal = scanGoal(node_radius[main_count]);
 DelayMilli(250);
}
```

```
  main_count++;

  //check if goal is achieved
  PrintMessage("CMD_SEN_ID");
  DelayMilli(500);
  String goalID = Serial.readString();
  DelayMilli(500);
  if(goalID.toFloat() == 2)
  {
    fire_count = main_count - 1;
  }
}


//back track to starting position by going back
its original path
void takeMeHome(void)
{
  if(main_count > 21)
  {
    main_count = 21;
  }
  main_count--;
  switch(main_count)
  {
    case 20:
      moveDown(3);
      turnCCW(90);
      break;
    case 19:
      moveDown(2.5);
      break;
    case 18:
      moveDown(1);
      turnCCW(90);
      moveDown(2.5);
      break;
    case 17:
      moveDown(5);
      turnCCW(90);
      break;
    case 16:
      moveDown(4.5);
      turnCW(90);
      break;
    case 15:
      moveDown(3);
      if(country_road == false)
      {
        turnCW(90);
      }
      break;
    case 14:
      if(country_road == true || (fire_count !=
13))
      {
        moveDown(4);
        turnCCW(90);
        moveDown(1.5);
        main_count--;
      }
      else
      {
        moveDown(4.5);
        turnCCW(90);
        moveDown(1.5);
      }
      break;
    case 13:
      moveDown(2.5);
      turnCCW(90);
      moveDown(1);
      moveDown(5);
      break;
    case 12:
      if(country_road == true || (fire_count !=
11))
      {
        moveDown(5);
        turnCCW(90);
        moveDown(1.5);
        main_count--;
      }
      else
      {
        moveDown(4.5);
        moveDown(4.5);
      }
      break;
    case 11:
```

```
    moveUp(4);
    turnCCW(90);
    moveDown(1.5);
    break;
  case 10:
    if(country_road == true || (fire_count !=
9))
    {
      moveDown(5);
      moveDown(4);
      main_count--;
    }
    else if(fire_count <= 6)
    {
      moveDown(3.5);
      turnCCW(90);
      moveDown(2.5);
      main_count = 7;
    }
    else
    {
      moveDown(2);
      turnCCW(90);
      moveUp(4);
    }
    break;
  case 9:
    moveDown(4);
    turnCW(90);
    moveDown(2);
    moveDown(5);
    break;
  case 8:
    if(country_road == true || (fire_count < 1
&& fire_count > 8))
    {
      turnCW(90);
      moveDown(4);
      turnCCW(90);
      moveDown(3.5);
      main_count = 1;
    }
    else
    {
      moveUp(4);
    }
    break;
  case 7:
    moveUp(1.5);
    turnCCW(90);
    moveDown(2.5);
    break;
  case 6:
    moveDown(1);
    turnCW(90);
    moveDown(3);
    break;
  case 5:
    moveDown(3.5);
    break;
  case 4:
    moveDown(3.5);
    turnCW(90);
    break;
  case 3:
    moveDown(3.5);
    break;
  case 2:
    if(country_road == true )
    {
      moveDown(4);
      main_count = 1;
    }
    else if(fire_count != 1)
    {
      moveDown(4);
      turnCCW(90);
      moveUp(1);
      main_count = 1;
    }
    else
    {
      moveDown(4);
      turnCCW(90);
      moveUp(5);
      moveUp(1.5);
    }
    break;
  case 1:
    moveDown(4);
    moveDown(1.5);
    break;
    case 0:
      moveDown(0.5);
      turnCW(90);
      moveDown(0.5);
      break;
  }
  DelayMilli(2000);
}
```