18131611

Title:

Maze solving robot

report

Submission date:

22 October 2021

Contents

[Part 1 3](#_Toc85830543)

[Class Structure 3](#_Toc85830544)

[Testing 8](#_Toc85830545)

[Serial monitor output 10](#_Toc85830546)

[Part 2 11](#_Toc85830547)

[Class structure 11](#_Toc85830548)

[Testing 11](#_Toc85830549)

[Serial monitor output 11](#_Toc85830550)

The maze is a four by four maze with a total number of cells equal to 16. It has a width of 30cm and height of 30cm. The black tape on the maze is useful for marking cells on the maze and directing the car along a path. By using Line tracking Arduino can tell that it has gotten to a new cell once a junction is reached, that is, when all the line tracking modules are activated. The ultra-sonic sensors come in handy when a decision has to made on which direction to take when multiple line tracking modules detect the line. By integrating the two functionalities, the car can efficiently navigate through the paths.

# Part 1

## Class Structure

**Servo**

**+**write()

+attach()

**Movement**

-carSpeed

-turningSpeed

+back()

+forward()

+left()

+right()

-stop()

**Car**

-Movement CarMovement

-UltraDistanceSensor ultraDistanceSensor

-LineTracking lineTracking

-int carState

- x

- y

- goalX

- goalY

- heading

+setup()

+run()

+updatePosition()

**LineTracking**

+detectLine()

**UltraDistanceSensor**

-Servo myServo

+attachServo()

+measureDistance()

+scanDistance()

1. The Servo class is an in-built library which is used to align the ultra-sonic sensors to different directions.

* The write method takes an angular value which in this case is between 0 and 180 meaning from right to left of the car. This program uses 8 (sharp right), 75(straight forward) and 150 (sharp left) angular numbers. It has not used 0,90 and 180 which would be obvious for a sharp right, straight forward and sharp left due to technical issues with the servo.
* The attach method is used for attaching the servo to a digital pin on the Arduino for outputting.

1. The LineTracking class is used for guiding the car along a path. It uses the line tracking module underneath the car to detect a black line against a white surface.

* The detectLine() function reads the LT\_R (for right pin), LT\_M(for middle pin) and LT\_L(for left pin) to guide the car along the path. It returns an integer which represents the input pin that has detected the line from which the direction can be determined. In order for the line tracking to function properly, specific instructions are provided to the Arduino for different scenarios. They are as follows,

If middle module alone picks up the black line, then a forward signal is returned so as to facilitate a forward motion.

Text

Description automatically generated

If the left and middle sensors alone detect the black line , a left signal is returned.

Text

Description automatically generated

If the left module alone detects the line then a signal for a left turn is returned.

Diagram

Description automatically generated with low confidence

If the middle and right module alone detect the line then a signal for a right turn is returned.

Text

Description automatically generated with medium confidence

If right module alone detects the black line then a signal for right turn is returned.

A picture containing text

Description automatically generated

If all the modules detect the line then a signal to imply a junction is returned. This program returns a selected constant value to represent a junction.

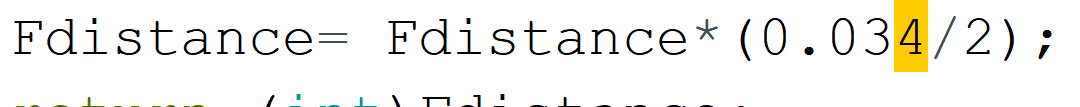
Text

Description automatically generated with medium confidence

1. The UltraDistanceSensor class is used for calculating the distance between the car and the obstacles along the car’s possible paths so as to decide on the best path, which in this case is the longest path.

* The measure Distance method measures and returns the distance between the car and any obstacle along the direction that the sensors are facing as set by the servo. It does this by making use of the ultra-distance sensors, one of which sends waves to the environment and the other waits to receive signals. The two pins attached to the two sensors are ECHO and TRIG pins. The Echo trig pin is an output pin which receives voltage from the Arduino board which then sends the sound waves and the echo pin is the output pin which signals to the Arduino when the sound waves are received back.

From the formula speed equals to distance over time, the distance can be calculated by multiplying the speed of sound on air (which is known) and the time taken for the signal to be received back by the echo pin. The time however is divided by two since it is the total about of time from when the signal was sent and received back.



* The scanDistance method scans the area around the car to determine the longest path and return it. This program scans the right side , middle and left side of the car by setting the servo to different values. It first considers the middle distance which if in case it is more than 20 then there is no need to scan other directions hence a forward signal is returned.

If the middle distance is less than 20 then other directions are scanned too to determine the longest one. The program calculates both the distance from an obstacle on the right side and left side and then stores them. It then compares them to determine the longest one. In case the longest distance is less than 20 meaning there is an obstacle on that direction then the only way to go is back. Constant variables are used to represent either directions. This way helps to avoid ambiguity and confusion.

Text

Description automatically generated

1. The movement class facilitates the motion of the car. It sets the motor pins to different voltages to enhance different motions.
2. The car class is composed of Ultra distance sensor, movement and line tracking sensor classes to facilitate movement and direction.

* The carState variable is use to tell the car if it has reached to the goal or not. If carState is 1 then it has not reached the destination and so it should continue the path finding process and when carState is 0 then it stops since it has reached to the goal.
* The program uses integer variables to keep track of the location of the car. It follows the principle of x and y axis where x is the horizontal trajectory and y the vertical trajectory.
* The program also uses an integer variable to store the angular value of the direction that the car has moved to so as to be used to update the location. If heading is set to 270 then the car has moved forward, if set to 180 then the car has made a left turn, if set to 90 then the car has moved backwards and if set to 0 then the car has taken a right turn.
* The updatePosition method calculates the new horizontal and vertical position of the car according to the direction it took. It does this by using a formula that a value of -1,0 or 1 to the current values in relation to the angular direction taken by the car.

Diagram, text

Description automatically generated with medium confidence

* The run method is the major method that brings together all other functionalities so as to enhance the path navigation of the car.

It does this by first running the line detecting method from LineTracking class which tells it where to go or whether the car has reached a junction which in this case a different approach is used to decide what way to go. If it gets back LT\_M,LT\_R, or LT\_L , it facilitates forward motion, right turn or left turn respectively using the methods in Movement class.

A while loop is used so as to tell the Arduino that as long as the activated module(LT\_R or LT\_L) can still see the line then it should keep turning. The while loop is used only when making a turn(left or right) .

Text

Description automatically generated

When a junction is reached the stops, then run method turns to ultradistanceSensor class to measure the distance along all three directions to determine the most likely one. According to what the UltraDistanceSensor returns, the angular direction is set and then the car moves in the determined direction and then updates the location using the updatePosition method. The update method is called only when the car reaches a junction.

Since the car does not make an accurate left or right turn when all the line tracking modules are active. A technique is used to move the car a bit forward such that only the middle module is active for the car to make a 90 degree turn as shown below.

Text, letter

Description automatically generated

## Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Expectations | Outcome | Evaluation |
| Middle module activated(LT\_M) | The car moves forward | The car moves forward | It works really well especially when the black lines are clearly defined on the surface, if not the left or right module may pick up the line and make the car to turn even if it is meant to keep moving forward. |
| All modules(LT\_M,LT\_L,LT\_R) are activated | The car stops | The car stops | It works really well. Although when the speed is too fast it can get past the junction without stopping. It works even much better when the speed is much low. |
| If the middle distance is more than 20 when the car gets to a junction | Keep moving forward without the need to measure the left or right side of the car. | The car keeps moving forward without measuring the left or right side of the car. | The ultra-sonic distance sensors are very efficient in detecting that the forward distance is enough to keep moving. |
| If the middle distance is less than 20. | Scan the distances from other directions. | Scans the distances from other directions | In order to align the sensors to sharp right , middle and left, the values 8,75 and 150 have been used due to technical issues with the servo. |
| If the right distance is more than the left distance. | Check for an obstacle on the right side of the car | Checks for an obstacle on the right side of the car. | This is done to make sure that although right distance is more than the left, there could still be an obstacle on the right just that it is a bit further than the one on the left. |
| If there is no wall/obstacle on the right side. | Make a right turn. | Makes a right turn. | In order to make an accurate turn a technique is used which moves the car a bit forward when it gets to a junction. By using this technique the car has been able to make accurate turns unlike when it is not used. |
| If the left distance is more than the right. | Check for an obstacle on the left side by measuring the distance between the car and the obstacle. | Checks for an obstacle on the left side. | It checks for an obstacle by checking if the distance is less than 20. |
| If there is no obstacle on the left side of the car. | Make a left turn. | Makes a left turn. |  |
| If there is an obstacle Infront, left and right sides of the car. | Move the car backwards. | The car moves backwards. | The car has nowhere to go but back where is came from. However when the car gets to this point it might keep moving back and for the in a loop because once it moves back that means forward distance increases which forces the car to move forward again. |
| When the car moves to a new cell. | Update the position and check if the goal is reached. | The car’s position is well updated and checks made. | This method of checking whether the car has reached the goal can be tricky especially when it picks up random junctions. For it to work correctly the car should be able to clearly detect set junctions. |
| When the car reaches the goal. | The car stops. | The car does not stop, it keeps looking for the path. | The functionality is clearly defined on the code however there might be some miscalculations. |

## Serial monitor output

Graphical user interface, application

Description automatically generated with medium confidence Graphical user interface, text, application

Description automatically generated

# Part 2

## Class structure

**UltraDistanceSensor**

-Servo myServo

+attachServo()

+measureDistance()

+scanDistance()

**Car**

-Movement CarMovement

-UltraDistanceSensor ultraDistanceSensor

-LineTracking lineTracking

-MapAlgorithm mapAlgorithm

-int carState

- goalX

- goalY

- heading

+setup()

+run()

+updatePosition()

+motionAlgorithm

**MapAlgorithm**

-north

-south

-east

-west

-matrix[4][4]

-newMatrix[4][4]

-westWall

-northWall

-eastWall

-southWall

+x

+y

+rightDist

+leftDist

+middleDist

+heading

+scanCell

+addWall

+floddFill

**Servo**

**+**write()

+attach()

**Movement**

-carSpeed

-turningSpeed

+back()

+forward()

+left()

+right()

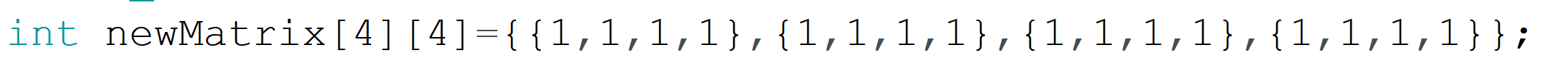
-stop()

**LineTracking**

+detectLine()

This part generates an algorithm that decides what path to take in relation to distances from where the car is to the goal cell. The program for this part is an extension from part 1 therefore the Movement, UltraDistanceSensor and LineTrackingSensor are the same.

A new class is added which enhances the whole process. This class “MapAlgorithm” generates a map detailing the distance from each cell to the goal cell. At first all the cells are valued one to the goal cell.



* The scanCell method is used to scan the cell for any out of bound values on the current cell such as [x-1][y] and remove them from the list of possible route to take by declaring that there is a wall in that direction.
* addWall method determines the directions with the distance value below or equal to 20 and eliminate them from the list of possible path to take from the current cell by declaring a wall in that direction. The distance values are the ones that were generated in the run method of the car class and stored in the MapAlgorithm class.

The program uses an 2D unsigned 8-bit integer array called matrix with zero values initially. The directions that the car can take are defined as north, south, east and west with values 1,2,4,8 respectively. When an a bitwise OR operation is applied between 0 and the directions it produces a value which can later be used to determine whether or not there is a wall in a certain direction within the cell. This is done by performing a bitwise AND operation between the value in the matrix and either directions which in turn tells the Arduino that there is no wall in that specific direction within the cell if the output is a zero, otherwise, there is a wall.

This function comes in use when the final map is generated and the car traverses the shortest distance to the goal.

* floodFill method is used to traverse through each cell and update its value to one more than the least neighboring value. It updates all other cells except the goal cell because it is the one that all other cells are measures from. It keeps updating until there are no more updates to be made
* motionAlgorithm method in MapAlgorithm performs all the necessary calculations and updates once the car enters a new cell(gets to a junction). It does this by implementing individual methods that is, scanCell, addWall, and floodFill methods. Then determines what direction to take(least value neighbour) by setting the heading to the related angular value. The run method which called the motionAlgorithm method then moves the car to the desired position and then update the car’s position.

## Testing

The rest of the functionalities before the car reaches the junction is the same as part one.

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Expectation | Outcome | Evaluation |
| If all modules(LT\_M,LT\_L,LT\_R) are activated. | 1. Scan and store the distances between the car and any obstacles on the right, left and ahead of the car. | Scans and stores the distances from either directions. | Integer variables are used to store the distances measured by the ultraDistanceSensor in the MapAlgorithm class for later use. This approach is efficient because we calculate the distances in advance and use them when needed. |
| 1. Scan the cell to get rid of any out of bounds positions. | Scans the cell to get rid of out of bounds cells. | The approach used however, only check the cells that are in front and left side of the car. This works really well when the car is at the bottom left corner. If in case the car is at any other locations say at the top right, other out of bound positions will be missed. |
|  | 1. Add walls within the current cell that the car is in. | Adds walls if any . | The program uses checks the distances that were stored when it got to the junction and if any of them is less than twenty, then there is a wall. The program uses a bitwise operation “OR” to insert walls in the 2D array which is a very efficient approach. |
| 1. Perform floodFill algorithm to recalculate the distances between all cells and the goal cell. | Performs floodFill algorithm. | The program uses a bool variable to check whether the value that is currently being recalculated changes or not. If it does then the flood fill process continues, if it does not then the flood fill process is terminated. |
| 1. Determine the least value cell and update the angular position of the next location. | Determines the next location with the least distance from the goal. | The program rules out the directions with walls and checks the ones without walls and then picks the shortest one. This is to ensure that the car uses the fastest route to get to its destination. |
| 1. The car moves to the desired(shortest distance from goal) cell. 2. Update position of the car. 3. If goal is reached. 4. If the goal is not reached, repeat the process from line tracking. | The car at first moves to the desired location and then loses its way.  The car’s position is updated.  The check is made successfully.  The process is repeated successfully | This could be due to some miscalculations on the code especially the fact that not all out of bound positions have been removed from the map.  The calculations made to set the new position works well although due to some miscalculations the car sometimes sets the wrong position which makes it difficult to tell whether it has reached the goal or not.  Although sometimes the car may reach the goal but due to some miscalculations, it misses it and keeps looking for path.  This is implemented through the while loop in the main file. |

## Serial monitor output

Text

Description automatically generated