MAZE SOLVING AGENT

Team Members:

Mohamed hussien 117969

Ramy Mamdouh 118371

Mostafa Ezz 118110

Contents

Maze Solver Agent

(Intelligent Car)

Abstract:

An agent (car) controlled by Arduino Uno microcontroller is used to solve maze problems. The agent deal with the maze using sensors to figure out its track or lane that he should go through it. The program is represented using Arduino programming language. Our aim is to make an agent that has the ability to reach its goal by perceiving the environment through sensors, and acting upon that environment through its effectors.

Introduction:

The problem is given a maze with one starting position and one ending position and it will have some blocked ways, the robot has to find its way from the starting position to the ending position. The robot supposed to find the way to exit the maze, the robot will start from starting position (start state) using sensors to know if there is an available way in this direction or not and using algorithm to move through the maze until it finds the goal state (exit).There are many techniques have been introduce to develop maze solver agent, but we have been choose the wall follower algorithm to minimize the time need to solve the maze, as in this technique the agent does not has to restart the mission by returning to its start point and try with another branch, as robots the returning step may waste a lot of time.

Objective:

Our objective is to develop an efficient, intelligent, small, autonomous robot to solve an arbitrary maze by finding the shortest fastest path to the destination.

System Architecture:

Agent perceptive: The agent perceive its environment via Infrared sensors proximity, the agent has three infrared sensors (front, right, left) that provide the agent with information about the availability of lanes.

- Infrared sensors proximity: IR Sensors work by using a specific light sensor to detect a select light wavelength in the Infra-Red (IR) spectrum. By using an LED which produces light at the same wavelength as what the sensor is looking for, you can look at the intensity of the received light. When an object is close to the sensor, the light from the LED bounces off the object and into the light sensor. This results in a large jump in the intensity, which we already know can be detected using a threshold. Since the sensor works by looking for reflected light, it is possible to have a sensor that can return the value of the reflected light. This type of sensor can then be used to measure how "bright" the object is. This is useful for tasks like line tracking.

Advantages:

Infrared sensors have several advantages. They can receive infrared light that is irradiated from both living and non-living objects. This is essential for many of the applications that infrared detectors are used for. In fact, infrared detectors can detect infrared light from far distances over a large area, much like the human eye is capable of detecting visible light. Infrared detectors operate in real-time and detect movement, making them ideal for security purposes. Likewise, infrared detectors help humans see what other devices do not allow them to see, such as leaks in underground pipes.

Disadvantages:

Although infrared sensors have many advantages, they also have several disadvantages. They are incapable of distinguishing between objects that irradiate similar thermal energy levels. Infrared detectors are also rather expensive, so they are not as widely used as they could be.

Actuators:

The car has three wheels, two of them is controlled by motors and the other wheel help the car to easily turn left or right. The car can turn left/right by forwarding one motor and backward the other. The motors controlled by L298N Dual H Bridge driver that give instructions to motors.

- L298N Dual H Bridge: This dual bidirectional motor driver, from Cana Kit, is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions. It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. The circuit incorporates 4 direction LEDs (2 per motor), a heat sink, screw-terminals, as well as eight Schottky EMF-protection diodes. Two high-power current sense resistors are also incorporated which allow monitoring of the current drawn on each motor through your microcontroller. An on-board user-accessible 5V regulator is also incorporated which can also be used to supply any additional circuits requiring a regulated 5V DC supply of up to about 1A.The circuit also offers a bridged mode of operation allowing bidirectional control of a single motor of up to about 4A.

  Features of L298N:

- Motor supply: 6 to 35 VDC
- Control Logic: Standard TTL Logic Level
- Output Power: Up to 2 A each
- Current Sense Outputs
- Onboard Power Resistors Provided for Current Limit
- Enable and Direction Control Pins
- External Diode Bridge Provided for Output
- Heat sink for IC
- Power-On LED indicator
- 4 Direction LED indicators

Agent Brain:

Arduino Uno is represented as the brain of the agent which memorize the algorithm and it also receive data from sensors, by process these data and make them understandable information, the agent take actions based on it by sending instructions to the driver.

- Arduino Uno: The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

SOFTWARE

An Intelligent Maze Solving Robot:

The primary purpose of the software is to maintain control over the hardware at all times and determine where to move by solving the maze [see figure 5.1). Controlling the hardware consists of reading the sensors and communicating with any external peripherals. Since alignment Corrections can be made by detecting the walls and respond according to the external environment

In addition to controlling the hardware, software must also keep track of the current position within the maze and deter-mine where to move based on the selected algorithm.

The idea of solution

The principal goal of Maze Solving Robot is to solve the maze and find the end of the maze as quickly as possible. To accomplish this task, the Maze Solving Robot uses a particular searching algorithm.

A vast amount of research on searching techniques already exists and is currently being undertaken.

Mathematicians in the fields of topology and graph theory have been studying maze creation and maze solving algorithms for several centuries. However, the algorithms they have developed are not feasible for these applications due to memory and speed limitations for robots. As a result, robots generally use some variation of the following three searching algorithms:

1. Wall following

2. Depth-First Search

3. Flood-fill

5.2.1 Wall following

Wall following is a simple algorithm in which the mouse chooses a wall, either left or right, then always keeps the chosen wall on its side as it moves through the maze. The algorithm is very simple to implement in code, The wall follower, the best-known rule for traversing mazes, is also known as either the left-hand rule or the right-hand rule. If the maze is simply connected, that is, all its walls are connected together or to the maze's outer boundary, then by keeping one hand in contact with one wall of the maze the player is guaranteed not to get lost and will reach a different exit if there is one; otherwise, he or she will return to the entrance having traversed every corridor in the maze at least once.

5.2.2 Depth-first search

Depth-first search is an intuitive algorithm for searching a maze in which the mouse first starts moving forward and randomly chooses a path when it comes to an intersection. If that path leads to a dead end, the mouse returns to the intersection and choose another path. This algorithm forces the mouse to explore each possible path within the maze, and by exploring every cell, the robot eventually finds the end of the maze. It's called “depth—first” because if the maze is considered a spanning tree, the full depth of one branch is searched before moving onto the next branch. The relative advantage of this method is that the robot always a route. Unfortunately, the major drawback is that the mouse does

not necessarily find the shortest or quickest route, and it wastes too much time exploring the entire maze (due to speed limitations).

5.2.3 Flood-fill algorithm

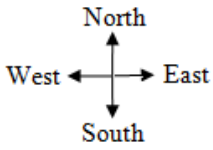
Flood-fill algorithm is a heuristic search technique wherein the mouse learns about its surroundings as it navigates through the maze and constantly updates its memory map to keep track of the walls and the shortest path. The flood-fill Algorithm involves assigning values to each of the cells in the maze where these values represent the distance from any cell on the maze to the destination cell. The destination cell, therefore, is assigned a value of 0. If the mouse is standing in a cell with a value of 1, it is 1 cell away from the goal. If the mouse is standing in a cell with a value of 3, it is 3 cells away from the goal. The fill flood algorithm is derived from the “Bellman Ford Algorithm”. It paved new methods by which complex and difficult mazes can be solved without having any troubles. The algorithm works by assigning value for all cells in the maze, where these values indicate the steps from any cell to the destination cell (Mishra and Bande, 2008).Implementing this algorithm requires to have two arrays. The first array is holding the walls map values, while the other one is storing the distance values (Southall, 2007).

Array Positions

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

Flood Fill

4	3	2	3	4
3	2	1	2	3
2	1	**0**	1	2
3	2	1	2	3
4	3	2	3	4



Bit No.	7	6	5	4	3	2	1	0
Wall					W	S	E	N

Map Array

array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
value	7	3	6	3	6	1	4	9	12	5	5	5	11	2	4	5	5	11	12	5	13	9	10	14	13

Distance Array

array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
value	4	3	2	3	4	3	2	1	2	3	2	1	0	1	2	3	2	1	2	3	4	3	2	3	4

Every time the mouse arrives in a cell it will perform the following steps:

(1) Update the wall map.

(2) Flood the maze with the new distance values (only if necessary).

(3) Decide which neighbouring cell has the lowest distance value.

(4) Move to the neighbouring cell with the lowest distance value.

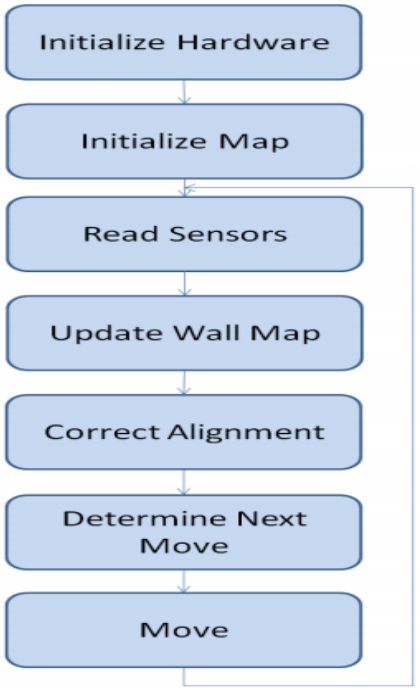
Used algorithm:

Wall follower:

The wall following algorithm is one of the simplest techniques that can be used to solve a maze. Essentially, the robot will take its direction by following either left or right wall. Whenever the robot reaches a junction, it will sense for the opening walls and select it direction giving the priority to the selected wall. By taking the walls as guide, this strategy is capable to make the robot reaches the finish point of the maze without actually solving it. The instructions involved in following wall for both left and right is given in table-1 below. However, this algorithm is no longer considered to be an efficient method in solving the maze. This because the wall follower algorithm will fail to solve some maze construction, such as a maze with a closed loop region

The left wall following routine	The right wall following routine
If there is no wall at left Turn left Else if there is no wall at straight Keep straight Else if there is no wall at right Turn right Else Turn around	If there is no wall at right Turn right Else if there is no wall at straight Keep straight Else if there is no wall at left Turn left Else Turn around

Design



The primary purpose of the software is to maintain control over the harc

At all times and determine where to move by solving the maze controlli

Hardware consists of reading the sensors, setting the motor speed and

Communicating with any external peripherals. Since each of the motor

Are controlled independently alignment correction can be made by incr

Decreasing the speed of a single motor

In addition to controlling the hardware, software must also keep track of the

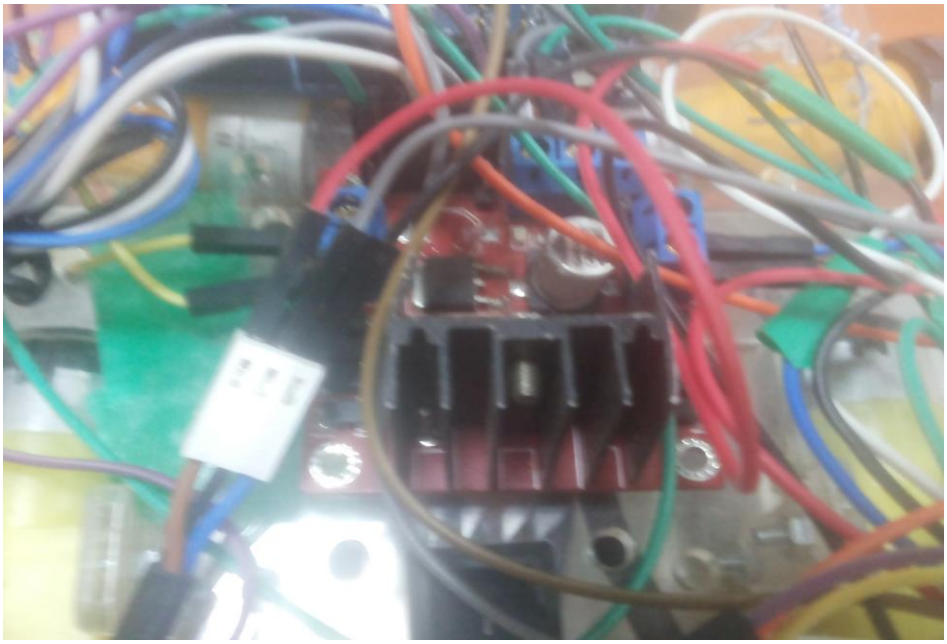
Current position within the maze ad determine where to move based on the

Selected algorithm

Mechanical design:

Car:

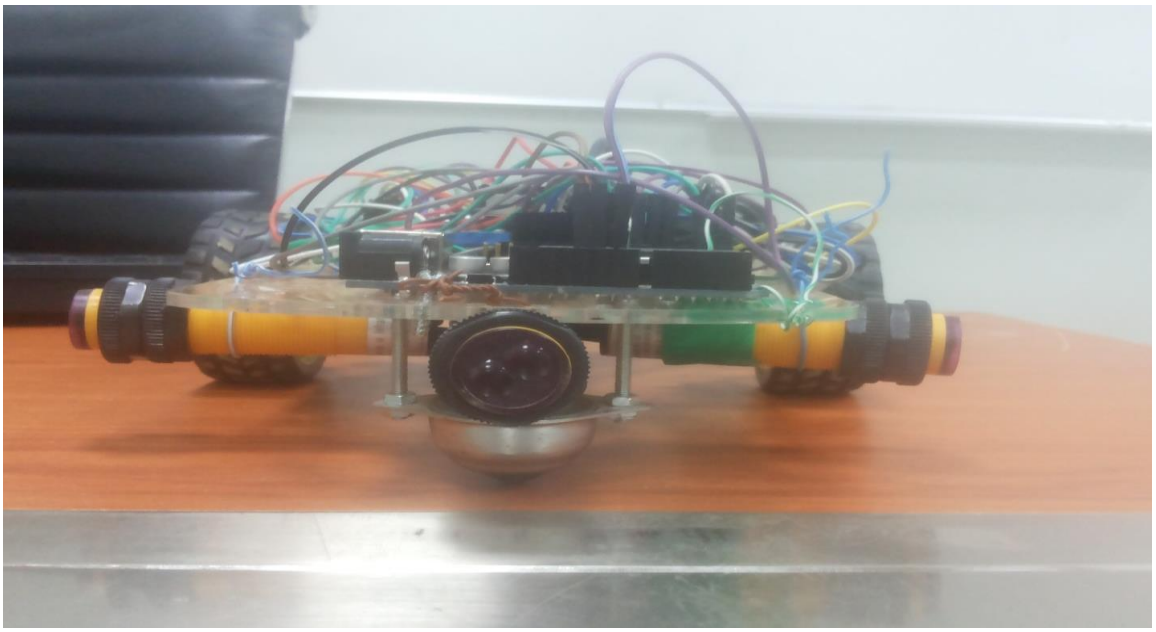
Motor Driver (Dual H-Bridge):



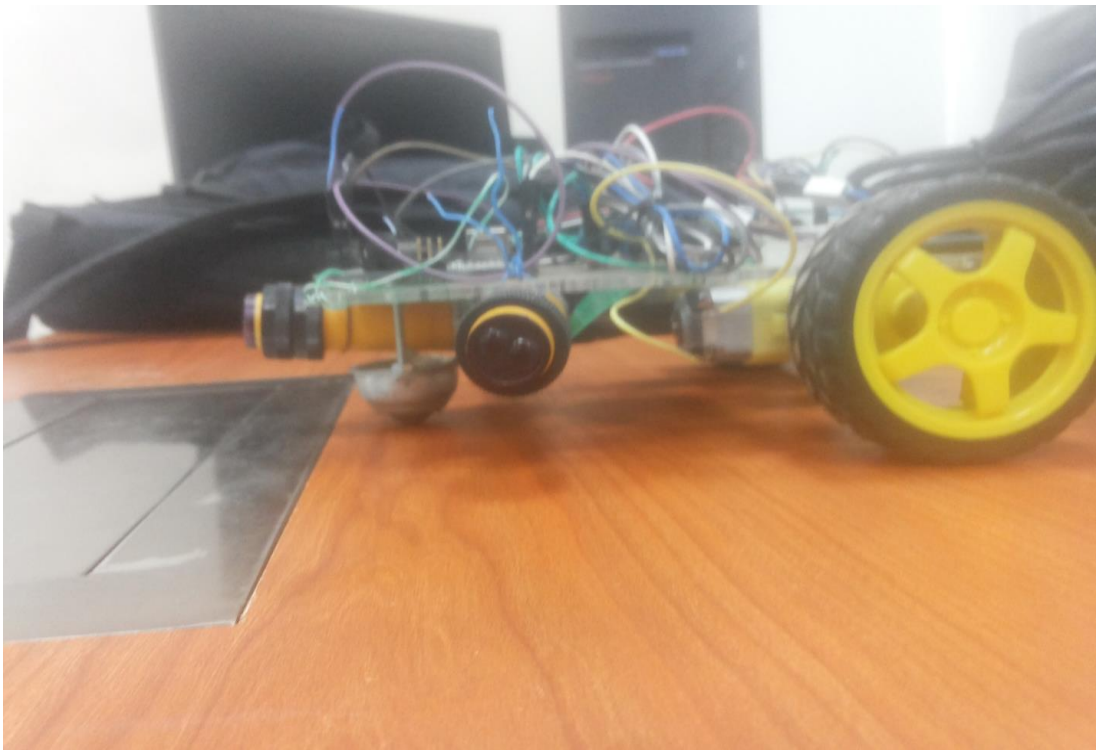
Motor:

Sensors:

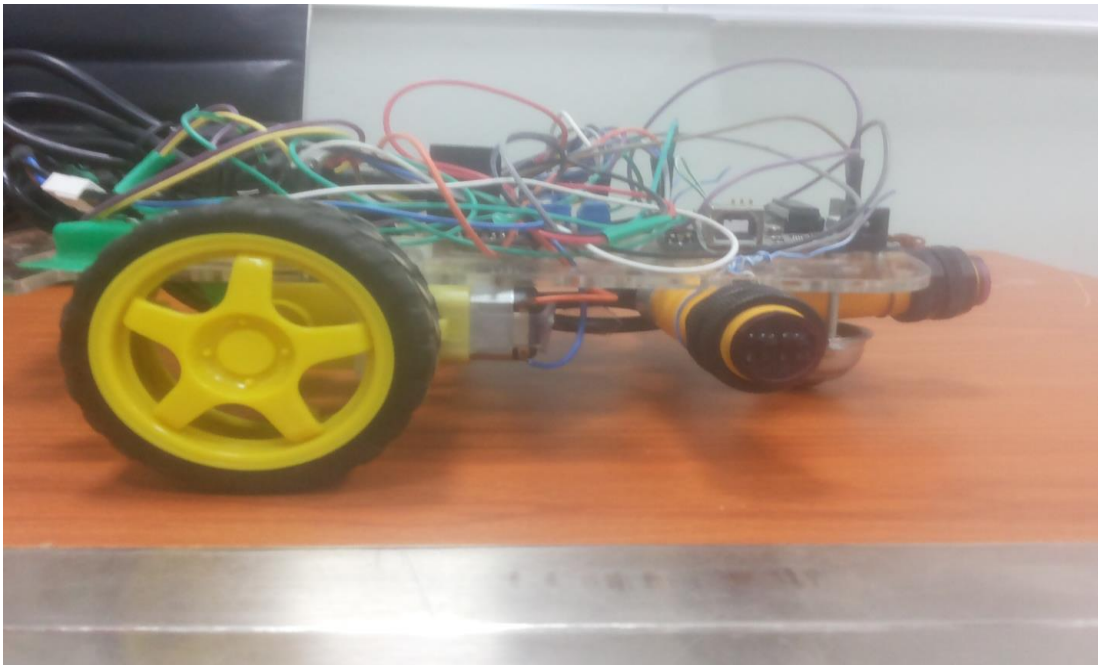
Front Sensor:



Left Sensor:



Right Sensor:



Arduino:

Battery:

Conclusion

The goal for the robot is to be a maze solving robot. I think that we completed this task it can execute the basic functions of being placed in a maze and navigating around it. It can identify when it has reached the end of the maze by using the sensors if there nothing around it then it exit the maze and it will stop.

There are few problems with the execution of the algorithms however. Due to the fact that the turning is calibrated by time, there is some chance that comes into play for turning. It often happens where the robot will turn too far and run into the walls. There is a limited amount of time where the sensors are of no use and he can get lost.

There are also some features that we would have liked to have included that were not in the robot. Due to time constraints we were not able to program the robot to return to the place that it started after it finds the tag.

Problems also arise in in the robot when it has to do loops. There were some problems with the algorithms that we implemented in certain cases. The algorithm rule for mazes only works if the maze is constructed in a particular fashion. It would easily be possible to make a maze where the robot would get stuck and would never be able to get to the end.

Turning can also be a big issue for the robot. As the motors are calibrated in order to turn for a specified amount of time turning is very dependent on the speed of the wheels. This means that changing from carpet to tile will make it probably not work. The batteries running low can also be an issue for it. As the batteries start to run out they turn slower and the time it takes to make a turn becomes greater. If not calibrated correctly it will not work.

The robot could be improved by implementing a better search algorithm that has memory in it. This would include encoders for the wheels to measure distance traveled and storing of turns and distance into the RAM of the microprocessor. A more obvious cue of the robot finishing its task would also be helpful.
