Maze Solving Algorithm for Autonomous Navigation Robot

RAJA KIRUTHIKA[†1] TIPPORN LAOHAKANGVALVIT[†1]
PEERAYA SRIPIAN[†1] MIDORI SUGAYA[†1]

Abstract: Autonomous navigation robots have the capability of moving from a point and finding the end point without any human assistance. In this study, we designed and developed an autonomous line maze solving robot using the LSRB algorithm and a PID controller. The designed robot uses the LSRB algorithm to learn and solve the maze by taking the shortest path after path optimization. A PID controller is later added as feedback control to increase the accuracy of the design. As a result, our designed robot is accurately able to learn and solve the maze by taking the shortest path.

Keywords: Maze solving robot, Algorithm, PID control, Navigation

1. Introduction

A maze is a network of paths, that generally has an entrance as well as an exit. Ever since the concept of Maze originated, many mathematicians have considered various techniques and algorithms to solve the maze [1]. Maze-solving problems and algorithms are considered an important field of robotics as it assists the rapidly growing field of Autonomous Technology.

Automation technology has been growing rapidly in recent years with the role of autonomous robots increasing in our day-to-day life. These robots can be used to transport items quickly from one location to another. An example of a day-to-day application of having an autonomous maze-solving robot is in the field of traffic navigation. Developing algorithms that allow one to reach from one point to another in the shortest time enables useful applications such as in emergencies for an ambulance [2]. Autonomous driving vehicles can also integrate the algorithms and techniques discussed in maze solving problems.

Currently, there are various algorithms used to design robots for maze solving problems. Wall following, flood fill algorithms, and pledge are popular algorithms employed while designing a maze-solving robot [3]. In this study, we started by designing a simple Line-Follower Maze solving robot consisting of IR sensors and motors using the LSRB (Left Straight Right Back) algorithm. PID control was then used to increase the accuracy of line tracing to solve various maze paths. A line maze solving algorithm can typically follow one of the two methods: LSRB algorithm or RSLB (Right Straight Left Back) algorithm [3]. A robot that follows the LSRB algorithm prioritizes taking a “left” turn when met with an intersection whereas a robot following the RSLB algorithm prioritizes a “right” turn. We used a feedback technique called PID control to increase the accuracy and the line following speed in addition to the LSRB algorithm.

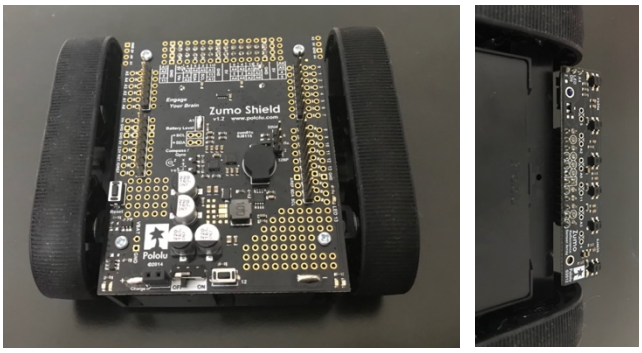
There are two goals for a typical autonomous maze-solving robot: (1) to learn and map out the maze and (2) to optimize the path by taking the shortest path to the desired endpoint. The algorithm and design used to reach these goals are discussed in the following sections of this paper.

2. Robot Control

2.1 Following the Maze

For the robot to follow the maze line, we used Infrared (IR) sensors to keep track of the path, turns, and directions. IR sensors contain a pair of the light-emitting diode and a receiver. The infrared light from the LED reflects off the surface and is detected by the receiver. They are often used to detect white and black surfaces. The use of digital IR sensors provides us with two outputs: 0 if it detects white and 1 if it detects black.

In our project, we use “Zumo Shield for Arduino” as shown in Fig. 1, which is equipped with 6 IR sensors. The IR sensor readings are used to determine when the robot must turn and its sense of direction. Fig. 2 shows sensor readings obtained from the Zumo robot.



(a) (b)

Fig. 1 (a) Zumo Shield for Arduino (b) Bottom view of IR sensors located on Zumo robot

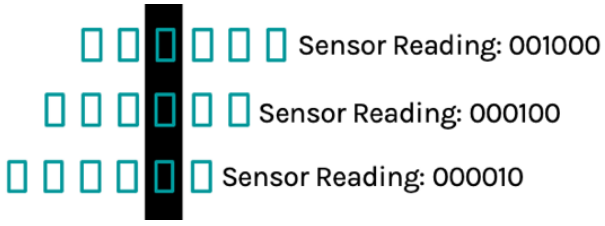


Fig. 2. Example of IR sensor reading in ZUMO robot

The robot identifies the maze and the turns using the IR sensor

†1 Shibaura Institute of Technology

readings as seen in Fig 2. If the maze consists of a left turn the IR sensor reads: 000001 or 000010. Depending on this the robot changes its direction as seen in Table 1.

Table 1. Sensor readings and corresponding direction

Sensor Reading	Direction
000100 or 001000	Go Straight
000001 or 000010	Right Turn
100000 or 010000	Left Turn
000000	Back Turn

2.2 Feedback Control

The following are the motor movements for each direction:

- To move straight/forward, both motors are set at the maximum speed and rotate forward simultaneously.
- To turn left, the right motor is set at the maximum speed and the left motor speed is set to 0.
- To turn right, the left motor is set to the maximum speed and the right motor is set to 0.
- To turn back, the robot makes a 180 degree turn in the left direction thereby setting the left motor to maximum speed and the right motor to 0.

The different steering conditions, directions, and their sensor values are illustrated in Table 1 and Fig. 3.

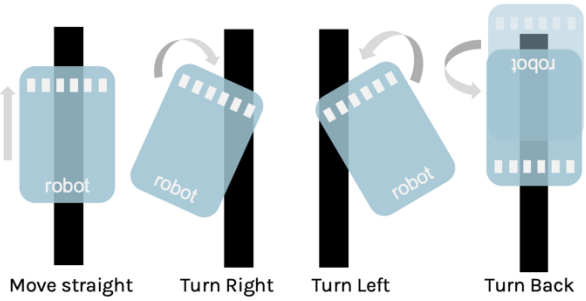


Fig. 3. Steering movements of Zumo robot

2.3 Feedback Control

A PID controller is used for feedback control. More than 95% of the control loops are of PID or PI type these days and can be found in all areas where control is needed [4]. In line mazes it is important for the robot to follow the line accurately despite steep turns and curves. Using a PID controller allows our robot to do this efficiently by monitoring the robot's speed and increasing its smoothness while making turns.

PID controller uses the combination of the following 3 basic values [5,6]:

- Proportional (P)
- Integral (I)
- Derivative (D)

Proportional value gives us the position of our robot with respect to the line. In this case, we control the speed of the right and left motors of the robot, so that it follows the center of a black line. So, when the robot is exactly on the maze-line we will get a proportional number of 0. When it is to the left of the line the value is positive and towards the right the value is negative.

Eq. 1 is the Proportional (P) controller in PID, where e is error, y is the input (position obtained from IR sensor) and r is the reference value (center value = 2500).

$$e = y - r \tag{1}$$

Eq. 2 is a P-controller that can be used as feedback control. The constant Kp is multiplied with e where u is the output value.

$$u = K_p \cdot e \tag{2}$$

This at times can lead to a never-ending overshoot of the robot so we introduce the integral term (PI-controller) as follows:

The integral value keeps track of the robot's motion. In other words, it is the sum of all the values of the proportional term (Eq. 3).

$$u = K_p \cdot e + K_i \cdot \int_0^T e(\tau)d\tau \tag{3}$$

The derivative value is the rate of change of the proportional. A derivative term is added to manipulate the output value after considering how fast the input values are changing (Eq. 4).

$$u = K_p \cdot e + K_i \cdot \int_0^T e(\tau)d\tau + K_d \cdot \frac{de(t)}{d(t)} \tag{4}$$

The proportional, integral, and derivative part together forms a PID controller [5, 6].

3. Learning the Maze

The first objective for an autonomous maze-solving robot is to learn the maze. Here the robot must start and reach the endpoint and learn all the possibilities it can take before reaching the end goal.

3.1 Maze Possibilities

Fig. 4. Illustrates all the possibilities the robot encounters while navigating the maze to learn it.

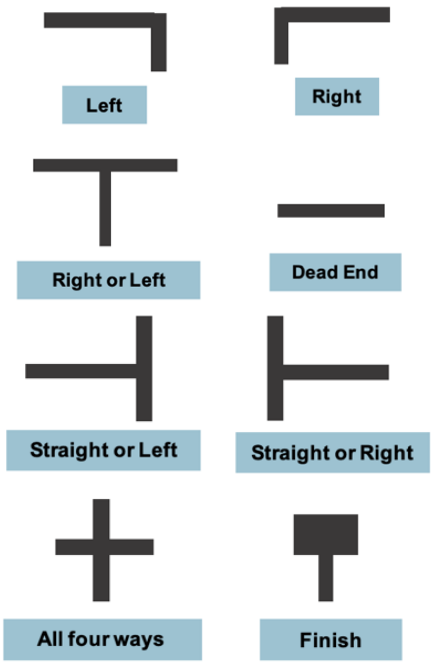


Fig. 4. All maze path possibilities

3.2 Algorithm

The robot operation for each possibility is decided based on the LSRB algorithm mentioned before. The algorithm resorts to taking a left turn when met with the different maze possibilities (Fig. 3). Each operation/turn made by the robot is stored in memory as a letter. This is later used by the robot to optimize the path the robot has taken. Using this algorithm, the robot reaches the end of the maze. Fig. 5 illustrates the flow chart for the LSRB algorithm.

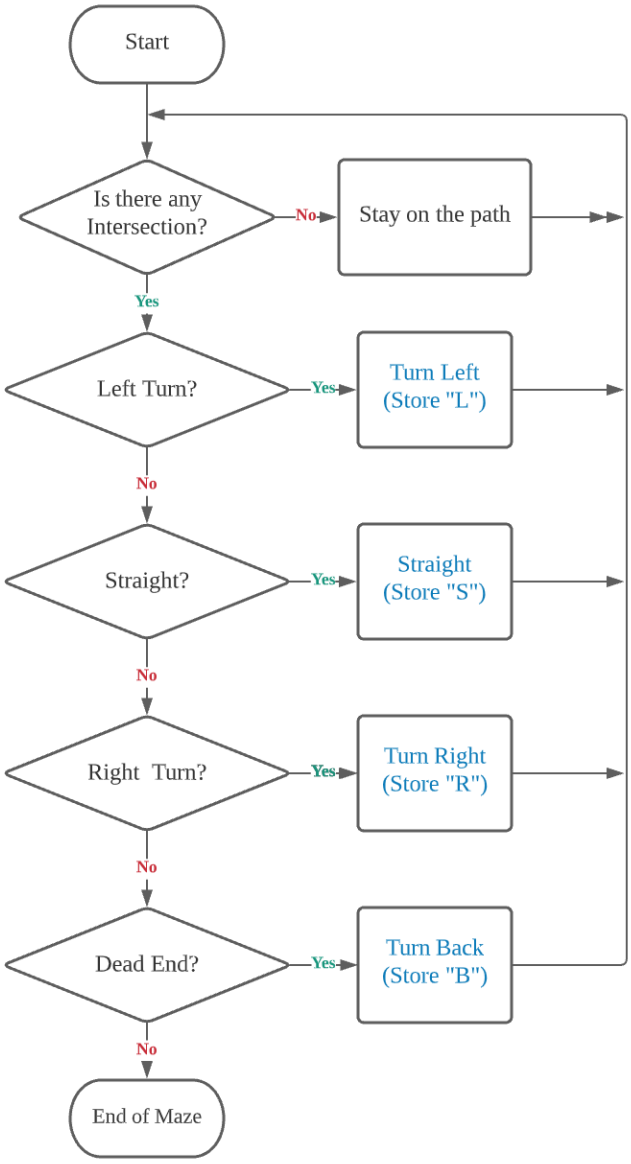


Fig. 5. Flowchart for LSRB algorithm

The following shows the robot operation for each of the maze possibilities discussed in Section 3.1.

- Possibility = Right: Operation = Right (R)
- Possibility = Right or left: Operation = Left (L)
- Possibility = Dead End: Operation = Back (B)
- Possibility = Straight or Left: Operation = Left (L)
- Possibility = Straight or Right: Operation = Straight (S)
- Possibility = All four ways: Operation = Left (L)
- Possibility = Finish/ End of Maze: Operation = Stop

4. Solving the Maze

Once the maze has been learned by the robot, the path needs to be shortened to avoid dead-end and solve the maze by taking the shortest route. All the turns taken by the robot are stored in its memory by using letters: “L” “S” “R” or “B”. The path optimization algorithm is explained in the following.

An example maze path is shown in Fig. 6, where the blue rectangular figure indicates the position of the Zumo robot. The steps of the robot solving the maze is as follows:

1. The robot comes to a four-way point and takes a left turn. It then reaches a dead end and turns back (Fig. 7).
2. The robot takes a left turn on its way back (Fig. 8).
3. The robot comes to a Straight or Right situation and continues to go straight till it reaches a dead-end (Fig. 9).
4. The robot takes a back turn and finally takes a left turn to reach its destination (Fig. 10).

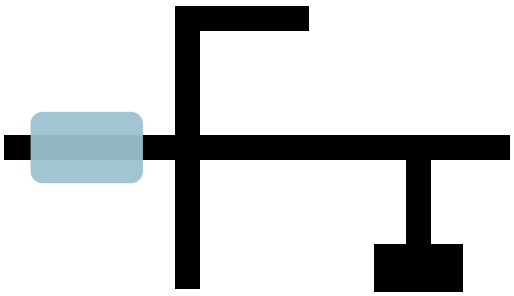


Fig. 6. Example maze path

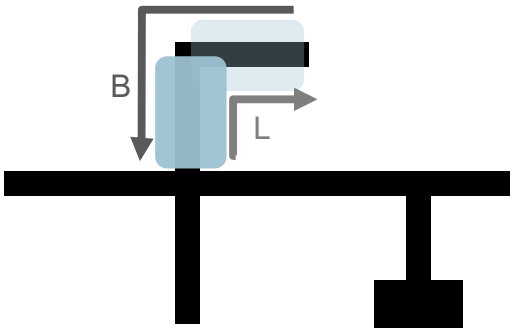


Fig. 7. Path stored in memory = [“L” “B”]

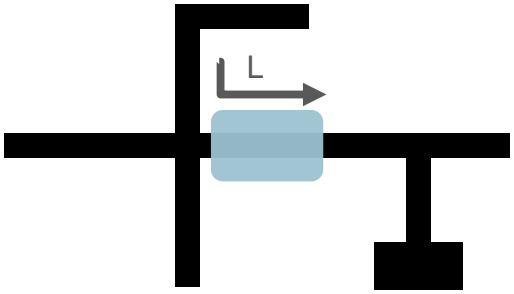


Fig. 8. Path stored in memory = [“L” “B” “L”]

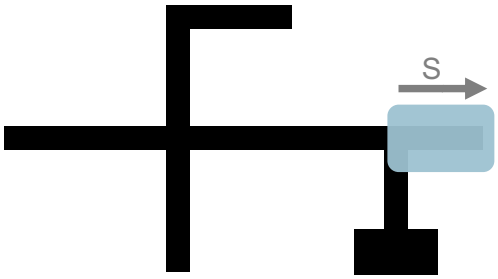


Fig. 9. Path stored in memory = [“L” “B” “L” “S”]

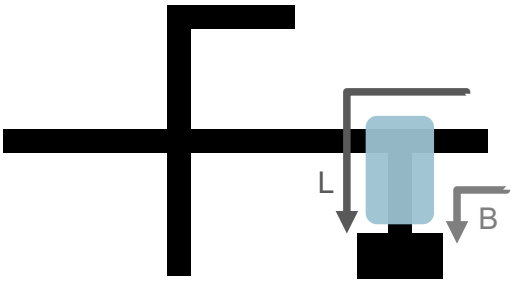


Fig. 10. Path stored in memory = [“L” “B” “L” “S” “B” “L”]

The final stored path when it reaches the end is observed in Fig. 10 is [“L” “B” “L” “S” “B” “L”]. We can see that this is not the shortest path for the robot to reach the end goal. Instead, we need to shorten the stored path by reducing the stored memory whenever it takes a back turn.

By observing the path taken we can tell that the path [“L” “B” “L”] is the same thing as if the robot were to store just “S”. Similarly, the path [“S” “B” “L”] is simply just “R”. Thus, the optimized path for the example maze in Fig. 6 is [“S” “R”].

Considering all the possibilities we obtained the optimized path stored in the memory as shown in Table 2.

Table 2. Optimization for path stored in memory

Memory Path	Optimized Path
[“L” “B” “R”]	B
[“R” “B” “L”]	B
[“S” “B” “L”]	R
[“L” “B” “L”]	S
[“R” “B” “R”]	S
[“S” “B” “R”]	L
[“L” “B” “S”]	R
[“R” “B” “S”]	L
[“S” “B” “S”]	B

After learning the maze given in Fig. 6, we have [“L” “B” “L” “S” “B” “L”] and by path optimization (see table 2 for reference) we get the optimized path as [“S” “R”]. The robot after learning will take this optimized path thereby solving the maze by taking the shortest distance.

If the second letter stored from the end in maze path is “B” when maze length stored in memory is 3 or more then we optimize the path stored in the memory. The following pseudocode is used for the Path Optimization Algorithm:

Algorithm 1 *Path Optimization Algorithm*

Input: An array p*ath* containing the path stored in memory, as well as the array length *mazeLength*.
Output: Optimized Maze path

```
if (mazeLength > 2 and path [mazeLength – 2] = ‘B’)
    totalAngle ← 0
    for i = 1 → 3, do
        if (path [mazeLength – i]) = “R” then
            totalAngle += 90
        else if (path [mazeLength – i]) = “B” then
            totalAngle += 180
        else if (path [mazeLength – i]) = “L” then
            totalAngle += 270
        else
            totalAngle += 0
        end
    end
    totalAngle ← totalAngle % 360
    if totalAngle = 0 then
        path [mazeLength – 3]) = “S” then
    else if totalAngle = 90 then
        path [mazeLength – 3]) = “R” then
    else if totalAngle = 180 then
        path [mazeLength – 3]) = “B” then
    else if totalAngle = 270 then
        path [mazeLength – 3]) = “L” then
    else
        do nothing;
    end
end
```

Fig. 11 Algorithm for path optimization

The algorithm (Fig. 11) was used to obtain path optimization results as shown in Table 2. For example, [LBR] we would have a total angle of 540, and after finding the remainder we have 180 thus optimized to ‘S’ in memory. The following algorithm is followed until the end of the maze path.

5. Results and Discussion

Experiments were conducted to confirm the effectiveness of our developed line-tracing robot with path optimization. At first, we used a simple line maze (Fig. 11) without PID control.

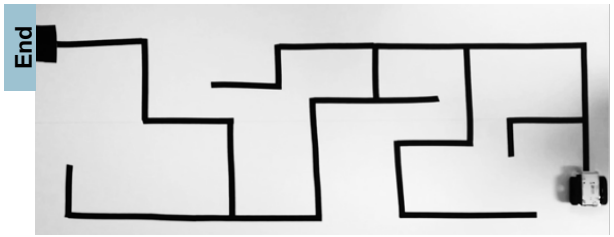


Fig. 12. Simple line maze

Initially while learning the maze the robot made 28 turns but by path optimization, we reduced this to 10 turns.

Table 3. Comparison of the number of turns and time taken between with and without path optimization

Criteria	Without Path Optimization	With Path Optimization
No. of turns	28	10
Time taken	31.7 seconds	10.8 seconds

When experimenting with only the LSRB algorithm and path optimization there were times in the experiment where the robot could not follow the line accurately while making sharp turns when it comes across intersections. To increase accuracy of robot while following the line we performed another experiment using PID control along with the algorithm. After using PID control the time decreased from 10.8 seconds to 10 seconds, thereby increasing the speed of our robot. From our observation of the robot behavior, this PID control also increased the capability of the robot to trace the line smoothly in an S-shaped line track, circular and straight motion. Systematic performance evaluation will be performed in our future work.

6. Conclusion and Future Work

In this study, we developed an autonomous navigation robot with maze solving algorithm. We implemented a method for solving and learning maze using the LSRB algorithm and path optimization method. Then we used PID for feedback control to improve our design by increasing speed of solving maze along with its ability to trace line. Experiment results show that the robot can effectively solve maze in less time by path optimization with higher accuracy in line tracing by PID control.

The PID control was tested in smaller mazes which involved the robot to take circular motion, steep turns and move on S-shaped tracks etc. The robot's behavior can be improved even more by adding higher performance hardware and enabling it to travel at higher speeds. For our future work, we will continue measuring robot’s performance on various tracks as well as employing other sensors such as replacing infrared sensors with ultrasonic sensors to remove the need for a line. The robots can they be compared based on various algorithms such as Wall follower, Tremaux’s algorithm and Recursive algorithm etc. The designed robot can then be used in real world applications that requires one to reach from the start to end point in the shortest distance available such as in traffic navigation. Factory or industrial robots can use it to transport objects. It can learn the maze path over time and take the shortest path to transport goods to and from warehouses. The robots can also be used in home automation for purposes such as in lawn mowers and cleaning robots.

References

[1] M. Rahman, “Autonomous Maze Solving Robot”, 10.13140/RG.2.2.34525.82403, 2017.

[2] M. Ben-Ari and F. Mondada, “Robots and Their Applications”, In: Elements of Robotics, Springer, Cham, pp. 1–20, 2018.

[3] B. Gupta and S. Sehgal, "Survey on techniques used in Autonomous Maze Solving Robot,” In: 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), pp. 323–328, 2014.

[4] K. J. Astrom, R. M. Murray, L. Desborough and R. Miller, “Feedback Systems: An Introduction for Scientists and Engineers”, Princeton University Press, 2002.

[5] M. Abdul Kader, M. Z. Islam, J. Al Rafi, M. Rasedul Islam and F. Sharif Hossain, “Line Following Autonomous Office Assistant Robot with PID Algorithm,” In: International Conference on Innovations in Science, Engineering and Technology (ICISET), pp. 109–114, 2018.

[6] V. Balaji, M. Balaji, M. Chandrasekaran, M.K.A. Ahamed khan, I. Elamvazuthi, “Optimization of PID Control for High Speed Line Tracking Robots”, Procedia Computer Science, Vol. 76, pp. 147–154, 2015.

Acknowledgments We would like to thank Dr. Adilin Anuardi for his lectures on IoT robotics and guidance on robot development in this research.