Autonomous Maze Solving Robot *

Conference Paper · May 2020

DOI: 10.1109/AQTR49680.2020.9129943

3 authors, including:

Gabriel Harja
Universitatea Tehnica Cluj-Napoca
14 PUBLICATIONS **92** CITATIONS
SEE PROFILE

Ioan Nascu
Universitatea Tehnica Cluj-Napoca
91 PUBLICATIONS **683** CITATIONS
SEE PROFILE

Autonomous Maze Solving Robot

*

Rares Covaci
Technical University of Cluj Napoca
Department of Automation
Cluj Napoca, Romania
rarescovaci95@gmail.com

Gabriel Harja
Technical University of Cluj Napoca
Department of Automation
Cluj Napoca, Romania
Gabriel.Harja@aut.utcluj.ro

Ioan Nascu
Technical University of Cluj Napoca
Department of Automation
Cluj Napoca, Romania
Ioan.Nascu@aut.utcluj.ro

***Abstract*—In this article, the construction and programming of a line-follower maze solving robot will be presented. In the beginning a general overview is offered, followed by the main technical aspects and implementation details. The main objective of the robot is exploration and terrain mapping. However, after finishing the exploration process, it is often desired that the robot must be able to find a specific object or return to a given point. The proposed terrain is a maze consisting of lines, turns and crossroads, which the robot detects using infrared sensors. An algorithm for exploring, learning and solving the maze has been implemented. Therefore, the robot manages to optimally find the exit of the maze during the second run. A Bluetooth module was used in order to broadcast real-time data and plot it in Matlab during the robot's operation.**

***Index Terms*—autonomous robot, maze solving, control, PI**

I. Introduction

Mobile robots are an increasingly growing technology and have a major importance in scientific and industrial areas, as their abilities of solving complex tasks have evolved continuously. There is a wide range of applications in which mobile robots are used: exploration of difficult terrain, surveillance and security [1], household activities (room cleaning), military applications (land mine detection), transportation (nuclear waste manipulation), rescue operations and even space exploration. However, being able to achieve these tasks comes hand in hand with a series of requirements: the mobile robot must run safely, both for its own safety and the surrounding environment [2] (not inflicting damage on surroundings or endangering living creatures). Moreover, considering limited moving space, mobility comes as an important factor, as executing the necessary moves within the given space boundaries often proves essential. Finally, being able to operate on its own is a major requirement for autonomous robots, as they must be able to find directions and reach certain targets based only on their sensors and software implemented algorithms.

Precision is a key concept for mobile robots. The expected precision can only be achieved by using special control algorithms that keep the robot movements in the prescribed limitations. For decades the PID controller has been the most common control algorithm because of its evident advantages: having a simple and clear structure and also being adequate for most control loops. K.J. Astrom states in one of his articles [3] that PID controllers are used in more than 90% of the control loops in the industrial applications.

The paper is organized as follows: the description of the autonomous robot is presented in the next section. Section III describes the control methodologies that have been used in this work. The optimization of the path, closed loop control simulation results and the comparison between two control algorithms are included in Section IV while last section summarizes the conclusions.

II. Robot construction

Regarding the robot construction, some aspects have been considered. First of all, the maze and the robot must be appropriately dimensioned. Furthermore, the robot must be able to run through the maze, turn around and take turns, if needed. A ball caster was chosen as front wheel in order to provide more mobility. Placing an extra platform above the chassis also helps reducing the size of the robot. The battery pack was placed on the lower platform for increased stability. The final version of the robot is shown in Fig. 1.

The following hardware components were used:

- STM32F303RE development board;

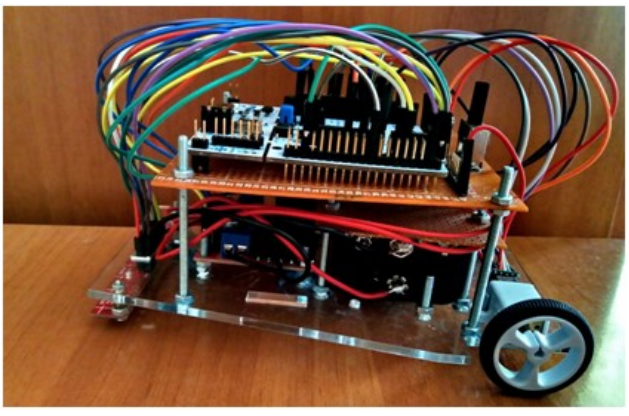

Fig. 1. Design overview of the robot

978-1-7281-7166-1/20/$31.00 ©2020 IEEE

- two DC 6V brushed motors, with 75:1 and Hall effect quadrature sensors with 900 pulses/rotation;
- one 6V DC power supply which consists of four 1.5V AAA batteries;
- L298N - electric motor driver;
- 8 analog infrared sensors for line detection;
- ball caster front wheel, for improved mobility;
- two 32 mm diameter rear wheels;
- other components (wires, plastic platforms).

The control algorithms were implemented on the microcontroller using C programming language. The code was developed using the Keil uVision5 environment, while the microcontroller peripherals were configured using the ST CubeMX tool.

III. Control Methodologies

This section describes the control strategy that was used in this work. For controlling the motor's speed, the PI algorithm was used while the navigation in the maze was done using the left hand on the wall algorithm presented in [4]. The PID controller is well known and is used on large scale in control systems and has three major components as proportional, integral and derivative. Though, PI controller is sufficient for the purpose of this paper. The algorithm is described in time domain by the equation that follows:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(t)\,dt\right) \tag{1}$$

The ideal PI algorithm in Laplace domain is represented as a transfer function in the following equation:

$$H_{PI}(s) = P + \frac{I}{s} \tag{2}$$

A. Motor control

In order to have the robot running at the desired speed, controlling the motor rotational speed is necessary. Rotational speed was measured in pulses per second, but the robot's speed can be easily calculated using the number of pulses per rotation given by sensors and the rear wheel diameter. Because the motors behavior can differ, even if both are the same model, it is required that model identification and controller implementation is done for each motor individually. Considering that the system nonlinearities are negligible, PI controllers are sufficient to meet these requirements. First of all, system identification was performed for each electric motor. The motors are equipped with Hall sensor encoders. The rotational speed was obtained by measuring the time elapsed between two successive pulses from encoder. The desired speed of the robot is 10 cm/s, so the duty cycle for identification was given accordingly: positive and negative steps between 40% and 50% percent duty cycle. The speed of each motor for the given input is presented in Fig. 2. The data acquisition was made using a microcontroller connected to PC via Bluetooth. The PC was running Matlab where all the system identification and control design processes were carried out.

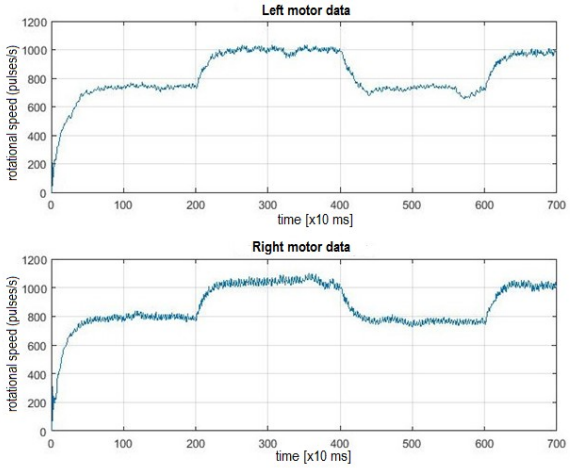

Fig. 2. Identification data

The linear model of the motor was obtained as a first order transfer function through step response identification. The resulted transfer functions for each motor are represented in the following equations:

$$H_{LM}(s) = \frac{2.57}{0.25s + 1} \tag{3}$$

$$H_{RM}(s) = \frac{2.50}{0.25s + 1} \tag{4}$$

For the implementation of the PI controllers we impose the closed-loop transfer function based on the desired behavior: zero steady state error and a response time of 0.2 seconds, which translates to a time constant of 0.05 seconds. The imposed closed-loop transfer function is:

$$H_0(s) = \frac{1}{0.05s + 1} \tag{5}$$

The PI controllers can be computed using the identified and the imposed closed-loop transfer functions:

$$H_C(s) = \frac{1}{H_{motor}(s)} \cdot \frac{H_0(s)}{1 - H_0(s)} \tag{6}$$

After applying the above equation for the two motors, the resulted controllers are:

$$H_{CLM}(s) = 0.9727 + \frac{7.782}{s} \tag{7}$$

$$H_{CRM}(s) = 0.9977 + \frac{7.978}{s} \tag{8}$$

For the microcontroller implementation, the controller discretization is required. Using the Tustin method and a sampling time of $T_s = 0.01sec.$. The sampling period was chosen to be at least two times smaller than the motor's time constant. To ensure that the sample time is met, a timer was configured to generate an interrupt every 10 miliseconds. The discrete controller transfer functions are:

$$H_{CLM}(z) = \frac{1.01165z - 0.93383}{z - 1} \quad (9)$$

$$H_{CRM}(z) = \frac{1,03710z - 0.95732}{z - 1} \quad (10)$$

We can then calculate the value for the control signal $C(k)$ at time step k based on its previous value and on the error signal ε at time steps k and $k-1$:

$$C_L(k) = C_L(k-1) + 1.01165\varepsilon_L(k) - 0.93383\varepsilon_L(k-1) \quad (11)$$

$$C_R(k) = C_R(k-1) + 1.03710\varepsilon_R(k) - 0.95732\varepsilon_R(k-1) \quad (12)$$

The results of the controller implementation are found in Fig. 3. The set point speed was given as follows: $600\ pulses/s\ for\ t \in [0,2), 900\ pulses/s\ for\ t \in [2,4), 800\ pulses/s\ for\ t \in [4,6), 600\ pulses/s\ for\ t \in [6,7]$.

B. Direction control

For identifying which sensor detects the line, numbers were assigned to each sensor as shown in Fig. 4. The infrared sensors are read every 100 ms. This sample time results by considering the robot's desired speed $(10cm/s)$ and the maze line width $(1.5cm)$. To ensure that the robot does not miss any crossroad or turn, the minimum required sample time is:

$$T_{S_{min}} = \frac{w_{line}}{v_{robot}} = \frac{1.5cm}{10cm/s} = 0.15s = 150ms \quad (13)$$

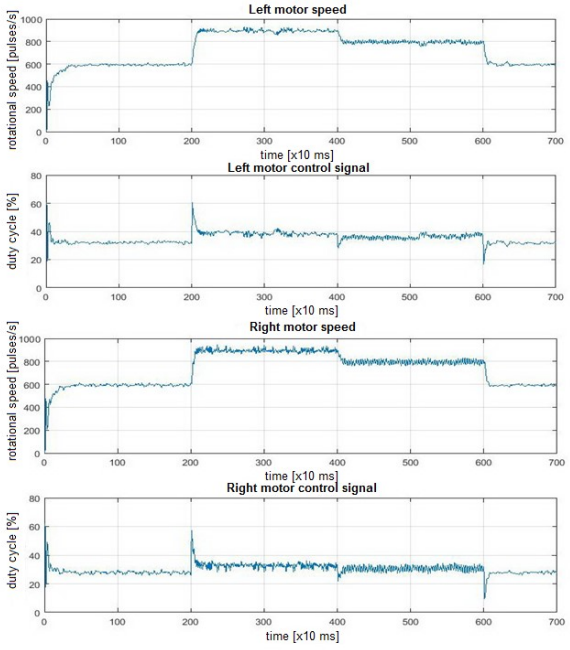

Fig. 3. Closed-loop motor control

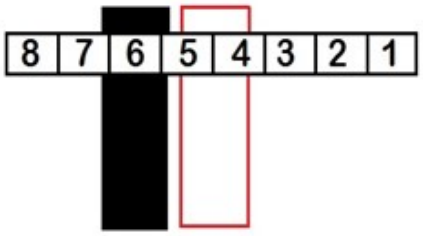

Fig. 4. Line following

Similarly to the motor control implementation, another timer was configured to generate an interrupt every 100 ms. Direction control was implemented by adjusting the motor speeds based on the relative position of the detected line to the central sensors, as described in Algorithm 1. Note that vl_{sp} and vr_{sp} are the speed setpoint values for the left and right motors.

Algorithm 1 Direction control algorithm

if $sensor7 > threshold$ **then**
  $vl_{sp} = 200; vr_{sp} = 1100$
else if $sensor2 > threshold$ **then**
  $vl_{sp} = 1100; vr_{sp} = 200$
else if $sensor6 > threshold$ **then**
  $vl_{sp} = 500; vr_{sp} = 700$
else if $sensor3 > threshold$ **then**
  $vl_{sp} = 700; vr_{sp} = 500$
else if $(sensor4 > threshold)$ and $(sensor5 > threshold)$ **then**
  $vl_{sp} = 600; vr_{sp} = 600$
else if $sensor4 > threshold$ **then**
  $vl_{sp} = 600; vr_{sp} = 500$
else if $sensor5 > threshold$ **then**
  $vl_{sp} = 500; vr_{sp} = 600$
end if

IV. Solving the Maze

It is important to mention that, at the start of its first run, the robot has no prior knowledge about the maze. For solving the maze, the robot uses the left hand rule algorithm, also known as left hand on the wall algorithm. This algorithm assumes that the maze contains no loops. The robot uses the infrared line sensors to detect a crossroad, and then chooses the left-most possible direction: if it is possible, go left, otherwise, go straight. If the road ends, the robot must turn around. Turning around is done by reversing the direction of the left motor while keeping the right motor running. The robot stores each decision inside an array in the microcontroller's memory, as a character (8-bit char) corresponding to the chosen direction (L for left, S for straight, R for right and B for turnarounds) such as in Fig. 5.

In order to increase the algorithm efficiency and reliability, the left hand rule algorithm presented in [4] was modified

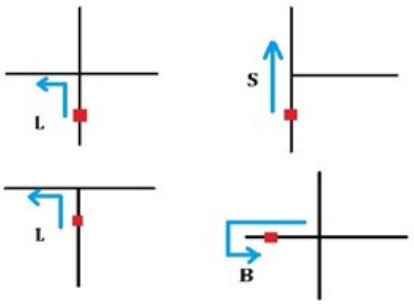

Fig. 5. Maze possible decisions

as follows: the turns inside the maze are not considered crossroads and they are ignored by the robot. The reason is that, if there is only one available direction to follow, the robot has no other option and no decision is required. Furthermore, simple turns are not relevant for path optimization, for the same reason described above. Another reason for ignoring the maze turns is the limited amount of memory available on the microcontroller.

The robot will keep running according to this algorithm until it finds the maze exit, which is marked by a black square. The robot knows that it has found the maze exit when it detects the black square with all its eight line sensors. It will then blink a green LED on the development board and wait for the user to send it for a second run.

A. Path optimization

By the time the robot has reached the maze exit, its memory contains a string with all the decisions taken at crossroads. Before starting the second run, the robot optimizes his path based on the following formulas, where each letter corresponds to a direction as described before:

- LBR = B;
- RBL = B;
- SBS = B:
- LBL = S;
- SBL = R;
- LBS = R.

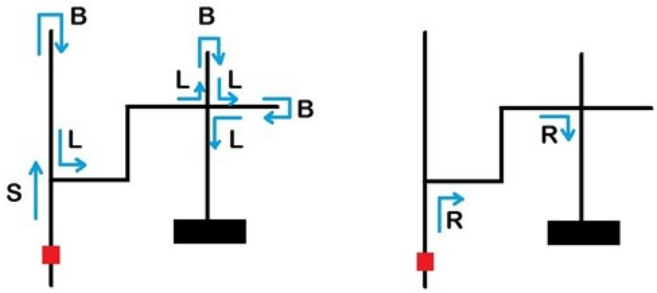

Fig. 6. Path optimization

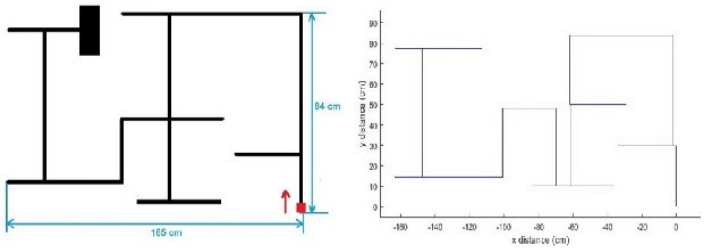

Fig. 7. Maze mapping

B. Real-time mapping

After the path optimization, the number of characters in the string will be the same as the number of crossroads in the maze, and the robot will know what decisions to make so it can find the shortest path to the maze exit. It is necessary that, for the second run, the robot departs from the place where it started its first run. One path optimization example is depicted in Fig. 6. After the first run, the resulted decision string is $SBLLBLBL$ and becomes RR after optimization process.

The real-time maze mapping was done by integrating the rotational speeds of the two motors and reading the robot's decision at each crossroad. Although the mapping is affected by the wheels sliding on the maze surface, the results are still satisfactory. The speed data was integrated on the computer after reading it from the receiver and plotted in Matlab. An example is shown in Fig. 7.

V. Conclusions and Future Work

A method for solving and learning non-looping maze was successfully implemented by using the left hand on the wall algorithm and some optimization techniques. However, the robot's behavior can be improved by adding higher performance hardware and enabling it to travel at higher speeds. Infrared sensors can also be replaced with ultrasonic sensors, removing the need for following a line. The maze solving algorithm can be improved by implementing optimization algorithms for a looping maze. Another point where improvements can be made is storing the decisions made by the robot. For more efficient memory usage, the path optimization can be done while the robot runs through the maze, so that the decision array will occupy as little memory as possible. Sliding has also been a problem on certain surfaces, affecting the maze mapping, but this can be solved by using ruber wheels with a higher grip coefficient.

References

[1] Nehmzow, U. "Mobile Robotics: Research, Applications and Challenges", University of Manchester, United Kingdom
[2] D.Floreano et al, "Design, Control and Applications of Mobile Robots", Swiss Federal Institute of Technology
[3] K. J. Astrom and T. Hagglund, "The future of PID control", Control Engineering Practice, Volume 9, Issue 11, pp. 1163-1175, 2001
[4] Ahmad Zarkasi et al, "Implementation of RAM Based Neural Networks on Maze Mapping Algorithms for Wall Follower Robot", 2019 J. Phys.: Conf. Ser. 1196 012043