PROGRAMABLE AUTONOMOUS ROBOTS

Lucian PESTRIȚU[1], Cosmin GABRIAN[2], Octavian IORGA[2],
Andrei MELNEC[2], Cătălin Gheorghe AMZA[3]

Rezumat. *Această lucrare prezintă avansul tehnologic al micro-controlerelor programabile şi cum aceste circuite electronice pot fi „echipate" cu aplicații software complexe în aşa fel încât să poată acționa independent, devenind un aşa numit robot autonom sau agent autonom. Pentru a ilustra acest lucru, un robot de tip Pololu 3PI a fost folosit, are ca sarcină rezolvarea unui labirint (ieşirea din labirint). Pentru a face acest lucru posibil, a fost implementat un algoritm pentru a ajuta robotul să găsească şi să memoreze ruta corectă de ieşire din labirint.*

Abstract. *This paper aims to present how technology has advanced in terms of programmable microcontrollers and how circuits can be equipped with complex software so they can to act on their own, becoming a so-called autonomous robot or agent. To illustrate this, the 3PI robot is used, which is faced with solving a problem by itself, namely: solving a maze on it's own. To make this possible so we had to implement this robot with a computer algorithm that helps it to remember the route that it had just travelled and then find the shortest and fastest way to the destination point.*

Keywords: autonomous robots, maze-solving robots

1. Introduction

According to Wikipedia.com, an **autonomous robot** is a robot which can perform desired tasks in unstructured environments without continuous human guidance and intervention [1]. There are various degrees of autonomy for various existing robots. A high degree of autonomy is particularly desirable in fields such as space exploration, cleaning floors, mowing lawns, etc. A fully autonomous robot has the ability to [1]:

- Gain information about the environment;
- Work for an extended period without human intervention;
- Move either all or part of itself throughout its operating environment without human assistance;
- Avoid situations that are harmful to people, property, or itself unless those are part of its design specifications.

[1] Student, Computer Science and Automation Faculty, University Politehnica of Bucharest, Bucharest, Romania
[2] Student, IMST Faculty, University Politehnica of Bucharest, Bucharest, Romania (iorga.octavian@yahoo.com)
[3] Reader, PhD., IMST Faculty, University Politehnica of Bucharest, Bucharest, Romania (acata1@camis.pub.ro)

An autonomous robot may also learn or gain new capabilities like adjusting strategies for accomplishing its task(s) or adapting to changing surroundings. Furthermore, an autonomous robot, under certain conditions may become an autonomous agent. An **autonomous agent** is a new way of analysing, designing and implementing complex software systems [2]. An agent is a computer system situated in some environment (the robot in the present case) capable of flexible autonomous action to meet design objectives. It tries to fulfill a complex set of goals. It is *adaptive* by improving its competence at dealing with goals based on experience. *Robust* means that it never breaks down (graceful degradation when components within fail). *Effective* means that the agent is successul at eventually achieving goals. *Flexibility*: responsive-it perceives changes from environment and acts accordingly; pro-active it can take initiatives; social it is able to interact with humans and other agents [2]. A control system for a completely autonomous mobile robot must perform many complex information processing tasks in real time. It operates in an environment where the boundary conditions are changing rapidly. The usual approach to building control systems for such robots is to decompose the problem into a series of functional units as it will be demonstrated below using the Pololu 3PI robot.

2. The robot used in the experiments

The Pololu 3pi robot (Figure 1) is a small, high-performance, autonomous robot designed to excel in line-following and line-maze-solving competitions. Powered by four AAA batteries (not included) and a unique power system that runs the motors at a regulated 9.25 V, 3pi is capable of speeds up to 100 cm/second while making precise turns and spins that don't vary with the battery voltage. This results in highly consistent and repeatable performance of well-tuned code even as the batteries run low [3].

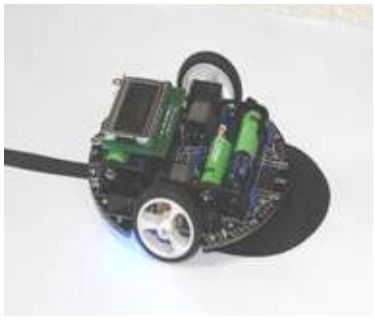

Fig.1. The Pololu 3PI Robot

Fig. 2. Example of maze

The robot is equipped with two micro metal gearmotors, five reflectance sensors, an 8x2 character LCD, a buzzer, three user pushbuttons, all connected to a user-programmable AVR microcontroller. The 3pi measures approximately 3.7 inches (9.5 cm) in diameter and weighs 2.9 oz (83 g) without batteries. The 3pi is based on an Atmel ATmega328 microcontroller, henceforth referred to as the "ATmegaxx8", running at 20 MHz. ATmega328-based 3pi robots feature 32 KB of flash program memory, 2 KB RAM, and 1 KB of persistent EEPROM memory. Free C and C++ development tools are available, and an extensive set of libraries make it easy to interface with all of the integrated hardware [3].

3. The implemented algorithm

The problem imposed for exemplifying how an autonomous robot is implemented to the 3PI robot is solving a maze, which means going through the maze from the start to the end in a fair way, without going through unnecessary lines.
The 3PI robot is able to travel on the lines of the maze with the help of its five reflector sensors. The line is usually black on a white background (preferably duct tape), but it can also be a white line on a black background. The thickness of the line should be approximately 2 centimeters.
The proposed labyrinth consists of several lines that meet the above requirements and have the particularity to have a starting and ending point (see Figure 2). The robot's mission is to get from the starting point to the end on the optimal route. In the present paper a labyrinth that doesn't contain loops has been created.
For this, the robot is implemented with an classic algorithm that will be presented in the following paragraphs. The algorithm used is written in C++ and is implemented with the help of WinAvr development platform.
The algorithm has two separate stages. The first stage consists of the robot exploring the labyrinth in his quest to find a solution (to reach the exit). During this stage, the robot may take all possible routes and spend a lot of time doing this. All moves are memorized by the robot. In the second stage, the robot must be able to recall the solution from its memory and solve the maze without having to go back and explore the maze again.
The implemented algorithm which makes the robot act on its own is named "left hand" [4-7]. It carries this name because the robot always tries to make a left turn at an intersection for the first time he goes through the labyrinth. For the robot to find the optimum course it is necessary for him to go through the whole labyrinth to learn every move it takes to get the end point. The explicit rule is that when the robot gets to an intersection, the first time it tries to make a left turn. If this is not possible it will then try to go straight ahead. In the last instance if neither of this is possible, it will go right. The major condition for this to work is that the labyrinth doesn't contain loops. Also we have to mention that one can also use a variant of

this algorithm the "right hand" rule, the conditions, reversing the terms listed if it preferred to go left.
Figure 3 presents all the cases in which the robot can be in an intersection of the labyrinth.

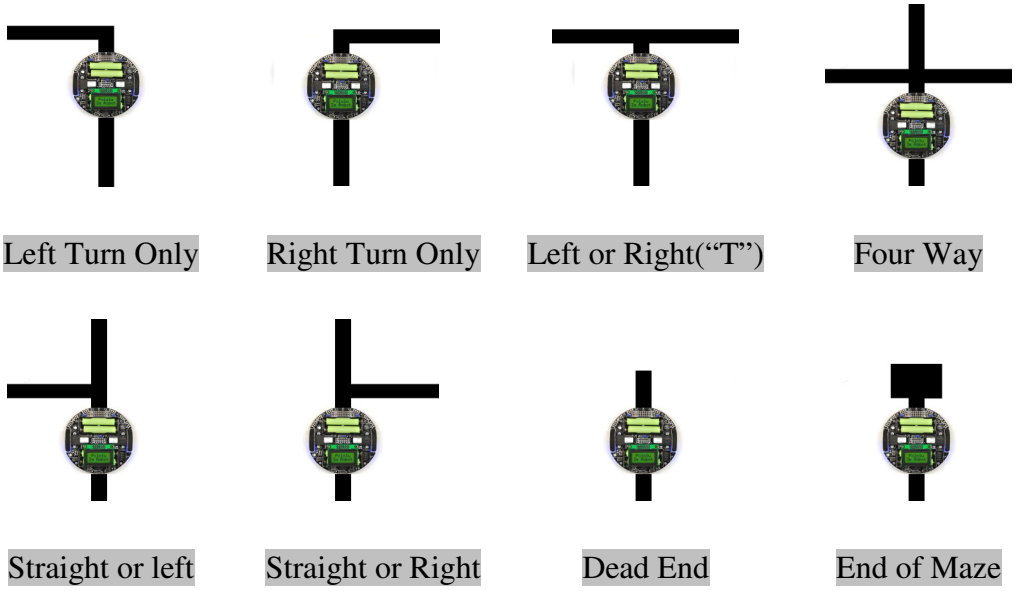

Fig. 3. Possible situations

The robot, most of the time, will be involved in one of the following behaviors:

(i) Following the line, looking for the next intersection;
(ii) At an intersection, deciding what type of intersection it is;
(iii) At an intersection, making a turn.

These steps continue looping over and over until the robot senses the end of the maze.
3PI robot uses sensors in the following way in order to unravel the maze lines: Line sensors shine visible or infrared light down at the floor and then measure the reflection. Using a 1 to mean "Sensor sees black." and 0 to mean "Sensor sees white.", a robot travelling along the black line to the right might produce several patterns:
1 0 0 0 0 = Line off to left
0 1 0 0 0 = Line a little to left
0 0 1 0 0 = Dead center!
0 0 0 1 0 = Line a little to right
0 0 0 0 1 = Line off to right

Following the line is relatively easy. Here is some pseudocode:

```
Select Case Pattern
Case Pattern = %00100 „ Full speed ahead
leftMotor=fast; rightMotor=fast
Case Pattern = %01100 „Go left a little
leftMotor=medium; rightMotor=fast
Case Pattern= %10000 „ Way off!
leftMotor=slow; rightMotor=medium
…and so on
leftMotor=fast; rightMotor=fast
leftMotor=medium; rightMotor=fast
leftMotor=slow; rightMotor=medium
```

Slow, medium and fast are arbitrary speeds that should get the robot turning or moving in the correct direction when straying from the line.
For a better understanding of the method proposed in the algorithm the following maze is proposed in Figure 4.

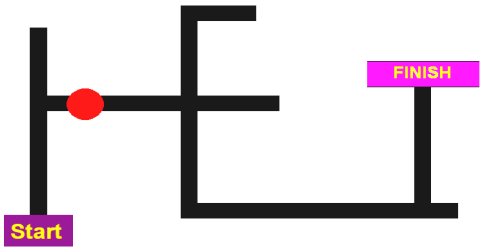

Fig. 4. Algorithm exemplification

The robot will leave from the START. Initially there isn't any information about the course in its memory. First te robot will go along the first line to the first intersection where it will be facing the question "what will I do next, go forward or go right?" and according to our algorithm it will prefer to go forward. At this moment the first information is written in its memory, regarding the first action at the first intersection. The letter S (straight) is what will be shown on its display. After this, 3PI will reaches a dead end, where according to the "left hand" algorithm it will go "backwards", the information in its memory for this action being the letter T (turn). Turning back it will reach the first intersection where he will make a "left turn" memorizing the letter L (left). Now the robot will have in is memory the combination of letters "STL". It can be noticed that ideally the robot should have made the first right instead of the combination "forward-backwards-left", so one can simplify its trail by replacing the combination of letters "STL" with letter R (right). At the second intersection the robot will go left then turn around and at the intersection will make another left. So at this moment in the meroy of 3PI there will be a route defined by the letters "RLTL". It can be

noticed that the directions used by the robot to go straight ahead were useless so it will appear a new optimization rule. So we replace “LTL” (left-turn-left) with “S” so in the robot’s memory will only the letters “RS”.

Again the robot reaches a dead end and he will turn back to the same intersection and making a left turn. Now in his memory he will have the letters “RSTL”. At this junction the robot would have followed the optimum route if he had made a right turn. So instead of going on the directions “straight-backwards-left” he will make a right, the letters now in his memory being “RR”.
At the final junction the robot is going to go left and reach the end of the maze. After going through the whole maze, 3PI will have in his memory the optimum directions he should go: “right-right- left-”(RRL)
From the given example we can simplify the “left hand” algorithm as follows:

- instead of “straight-back-left” the robot will only go “right”;
- instead of “left-back-left`` the robot will only go “straight”.

Conclusions and further work

In conclusion, it is remarkable how a microcontroller like 3PI has the ability of being programmed to manage on his own in a situation like the one we presented. The above problem is only a small part of what one can achieve when programming a robot to be autonomous. The labyrinth can be more complex and it can have obstacles and loops that make the presented algorithm inefficient. As future work we are going to enhance this algorithm with artificial intelligence techniques such as neural networks and fuzzy logic to overcome the problems presented above.

R E F E R E N C E S

[1] Autonomous navigation robots, www.wikipedia.org;

[2] R. Brooks, A robust layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, vol. RA-2, no.1, march 1996;

[3] The Pololu 3PI robot, www.pololu.com;

[4] Richard T. Vannoyii , Design a Line Maze Solving Robot;

[5] David Cook, Robot Building for Beginners;

[6] Myke Predko, Programming Robot Controllers;

[7] Joe Jones, Daniel Roth, Robot Programming.