# The algorithm for going through a labyrinth by an autonomous

*Prof.PhD.Eng.Eugen Raduca, Eng.Păun Adrian, Prof.Assist.PhD.Eng.Mihaela Raduca, PhD. Eng.Silviu Drăghici, PhD Eng. Cornelia Anghel Drugarin, PhD. Cristian Rudolf*

Electrical Engineering and Informatics Department
Eftimie Murgu University of Resita
Resita, Romania
e.raduca@uem.ro, padrian88@gmail.com
m.raduca@uem.ro, silviudraghici75@yahoo.com,
c.anghel@uem.ro, c.rudolf@uem.ro,

*Abstract*— **The paper presents an algorithm for going through a path type labyrinth by an autonomous vehicle. The detection of the path and the maintaining of the motion direction have been addressed as well as going through the labyrinth on road segments and the categories of crossings in the said labyrinth. The algorithm has been implemented in C++ and validated in an experimental model that has totally confirmed its correctness.**

***Keywords—algorithm, vehicle, autonomous, labyrinth, C++***

## I. Introduction

The auto field is one of the most productive important and dynamic sectors of the present. The branch of this field that deals with implementing IT technologies has known spectacular progresses and achievements in the last period, one of the main research directions being targeted towards the realizing of autonomous vehicles that present elements of artificial intelligence. The importance of this sector from the field is best indicated by statements of auto producers, who express the fact that the IT sector from the auto domain has not been practically affected by the recent global crisis. The work tackles just the theme of interest from the IT sphere associated with autonomous vehicles.

## II. PRESENTATION METHOD

In this present paper there is analyzed and solved the going through of an autonomous vehicle through a labyrinth type runway made up of right edged segments. The labyrinth has a single way out, and the algorithm of command for the vehicle is thus conceived that the vehicle, at any point in the interior of the labyrinth, may follow a path to find the way out. The algorithm, written in C++ [1] has been implemented on an experimental model. It is useful to state that the algorithm may be used as basis, without major modifications to the source code, for going through a vehicle on a road that also present curved runways.

The labyrinth on which basis the algorithm is presented is indicated in fig.1. In the center of the labyrinth one can notice the model of vehicle on which the experiments have been realized. For defining the whole runway in the labyrinth on which the vehicle may evolve, the said one, totally, is marked by a strip with a high absorption index. Concretely, for the presented case, one uses a black strip, on the basis of the labyrinth associated to the possible runways of the vehicle, that are, by contrast, white.

The principle after which the vehicle follows a runway path is highlighted in fig.2 [2].

It can be seen that an electronic device emit radiation for the photometer convenient frequency band which is strongly reflected in the route. A photo detector receives the reflected wave band, this being the information based on which the microcontroller UC sets the direction of the vehicle on which the vehicle is traveling below.

The UC used for the soft implementing of the algorithm is of Arduino type [3].

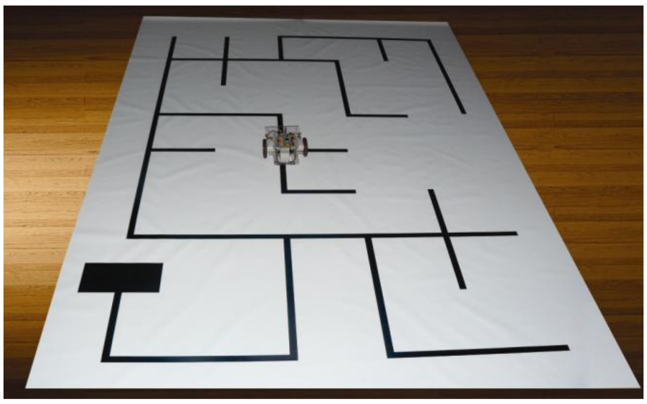

Fig.1 The labyrinth and the vehicle model with which the experiments have been carried out

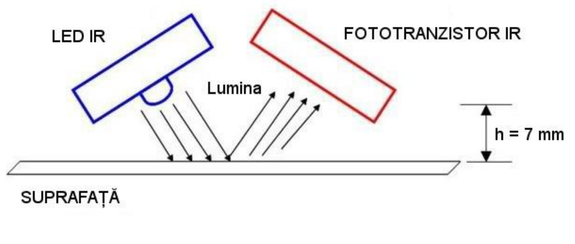

Fig.2 The work principle of the IR (infrared) sensors for detecting the path

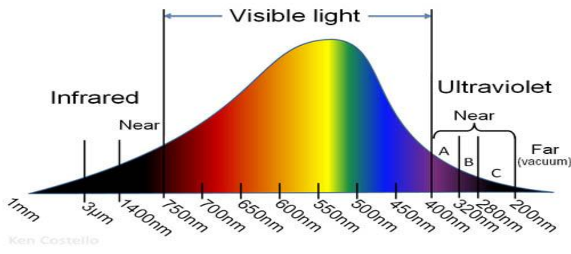

Fig.3 The specter of solar light

For the construction and implementation of the algorithm required several pairs of such sensors. Since a moving vehicle rule in natural solar emission can seriously disrupt the implementation of the algorithm. Therefore based on the energy distribution of the solar spectrum [4] (Figure 3) was used strap emission sensors - infrared reception [5]. In the algorithm, positional sensors are classified into two categories: central (2 pieces) and terminal (4 pieces, 2 of right terminal and two terminals left).

## III. The Algorithm for covering the path

### *A. Going through the labyrinth*

The vehicle has the task of completing the maze achieved between two endpoints known as "go" and "exit"; other endpoints of the maze are considered "turning points" and are those points where appropriate tape path ends abruptly without pay further options route endpoints that are landlocked. In other words "end points" except the two mentioned, there are points where the only option to continue the route for the vehicle when it came to such a point is a turning maneuver the vehicle and continue maze on another branch.

The starting point can be considered any terminal point of the maze during which output endpoint was distinctly marked by a black rectangle (Fig. 1). Marking separate exit point has been done to facilitate the detection of its unique avoid confusion with other markers of the route, such as intersections or absence of markings such as turning points. Between the starting point and the vehicle moves out of the maze segments of fixed lengths that can be crossed, for this case, only the angles of 90 °.

All segments, intersections, turning points and the presence of two terminal points of departure and exit from the labyrinth itself, the autonomous vehicle will need to go through. The components of the maze and the way in which they are positioned are summarized in fig.4.

.

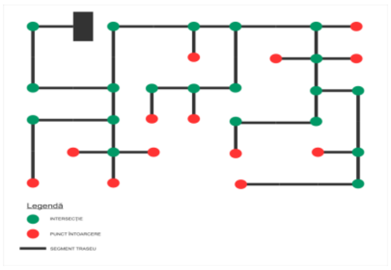

Fig.4 The structure of the labyrinth

Actual movement of the vehicle on the road is made by independent operation of two electric motors, each coupled to a wheel. Role writing program based on the algorithm is to determine at any time the control logic of each motor vehicle so as to move between points "out" and "exit".

### *B. The detection of the path and maintain the motion direction*

The detection path and to maintain the direction of movement is an iterative process involving a number of separate functions and ensures the continuous detection of the tape path by means of optical sensors in order to be able to determine the position of the vehicle relative to the track. Its deviation is processed and matched by generating two PWM signals [6].

The two signals will order 2 differential drivers of the vehicle by setting the duty cycle ensuring appropriate differential speed between the 2 engines, compensating the error gradually until it was eliminated.

In order to detect lane was implemented sensors _ *reading function ()* that is to make a sequential numerical analog conversion values received from the six sensors and determine the maximum and minimum values obtained :

```
Void reading _ sensors ()
{
 Minim = 0;
 Maxim = 0;
 Analog Read (A3); delay (1); s1 = analog Read (A0); maxim=s1; minim=s1; detected = 1;
 Analog Read (A3); delay (1); s2 = analog Read (A1);
 If (s2>=maxim) {maxim = s2; detected = 2 ;}
 If (s2<=minim) {minim = s2 ;}
 Analog Read (A3); delay (1); s3 = analog Read (A2);
 If (s3>=maxim) {maxim = s3; detected = 3 ;}
 If (s3<=minim) {minim = s3 ;}
 Analog Read (A3); delay (1); s4 = analog Read (A6);
 If (s4>=maxim) {maxim = s4; detected = 4 ;}
 If (s4<=minim) {minim = s4 ;}
 Analog Read (A3); delay (1); s5 = analog Read (A4);
 If (s5>=maxim) {maxim = s5; detected = 5 ;}
 If (s5<=minim) {minim = s5 ;}
 Analog Read (A3); delay (1); s6 = analog Read (A5);
 If (s6>=maxim) {maxim = s6; detected = 6 ;}
 If (s6<=minim) {minim = s6 ;}

}
```

The function converts the numeric analog is called *Read (pin)*, this function returns an integer numeric value in the range (0-1023) by the voltage applied to pin point the argument. The result of the conversion is assigned to the variables *s1*, *s2*, *s3*, *s4*, *s5*, *s6* which is the number from left to right in the bar sensor sensing. The *analog function Read ()* is part of the library "Wire.h" which is automatically included by the compiler Arduino IDE.

The variables used according to *maximum and minimum* are designed to continuously compare the current value with the previous sensor. In this way out of office, after all sensors have been traveled and compare, maximum and minimum values remain set, being declared global and can be used in the current cycle.

A variable has a *maximum* of great importance because; with the variable which is detected may be determined position of the vehicle from the lane which is black in the tread.

The vehicle is considered the center of the band center when the two sensors *S3 and S4* are detected at the same time, to a certain extent, the tape. Position straps and even number of sensors confirms that the absolute center of the vehicle is just halfway between *S3 and S4*. Since *sensors _ reading function ()* establishes a maximum even between these two central sensors, detect variable value will become one of them, the one with the highest value, i.e., one that sees the best band. Experiments have confirmed that when the middle of the vehicle lies between the two sensors, a correction can be made smoother, which is favorable for the movement of the segments crossing in the right angles of 90°.

Next there was implemented a general function driving called *SRA ()* that is designed to keep the vehicle on the road constantly compensating for deviations by generating two fixed frequency PWM signal *duty cycle* parameter variable in steps according to deviation detected from the position of the vehicle considered ideal (the middle of the vehicle to find the center of the black belt).

The motor control signal PWM1 PWM2 signal of the right and left of the vehicle engine can be obtained various rotational speeds of the two motors, which makes the wheels of the vehicle rotate differently. By consequence if the right wheel rotates faster than the left, the vehicle will swerve left and if left wheel rotates faster than the right vehicle to swerve to the right, that vehicle will be able to print the desired direction . Of course if the vehicle has to travel before PWM1 and PWM2 signals will be identical so that the two wheels will have the same peripheral speed.

In the *SRA function ()* parameter setting *duty1*, *duty2* representing engine duty cycle values right or left, is 8 bits (in the range 0-255). Function that implements PWM signal generation is *analog Write (pin, duty _ cycle)*. Calling them is after being called sensors _ reading function () variable detected after passing through a *switch control structure type (int.)*, which aims to set *duty1* and *duty2* by misconduct by selecting a single case of possible. For each case the variables duty2 duty1 and get a set of predefined values that ensure swift compensation misconduct. Structure selection *switch (detected)* is shown below:

```
Switch (detected)
 {
  Case 4:
  duty2 = 255;
  If ((s4-s3)>= 300) {duty1 = 160 ;} else {duty1 = 255 ;}
  Break;
  Case 3:
  If ((s3-s4)>= 300) {duty2 = 160 ;} else {duty2 = 255 ;}
  duty1 = 255;
  Break;
  Case 1:
  duty1 = 255;
  duty2 = c2;
  Break;
  Case 2:
  duty1 = 255;
  duty2 = c1;
  Break;
  Case 5:
  duty1 = c1;
  duty2 = 255;
  Break;
  Case 6:
  duty1 = c2;
  duty2 = 255;
  Break;
 }
Analog Write (pwm1, duty1);
Analog Write (pwm2, duty2);
```

The command principle of the vehicle after deviation, based on the information provided by the optic sensors, is suggested graphic in fig.5.

## C. *The detection and mapping the intersection*

The crossings or intersections represent the elements that transform the path in a Labyrinth, the intersections together with the turning points require a specific handling.

The four categories of intersections that the vehicle may encounter and that are found in the studied labyrinth are presented in fig.6.

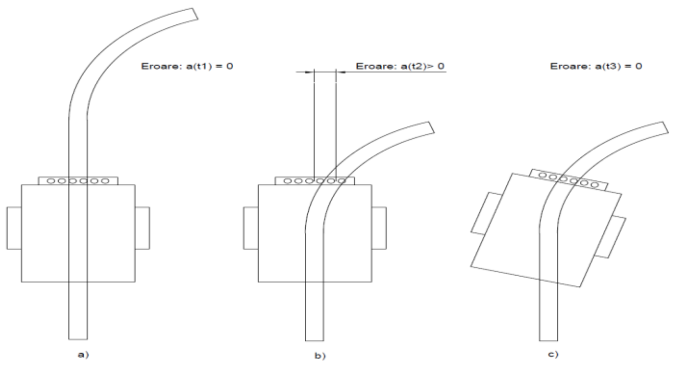

Fig.5 The command principle of the vehicle after deviation

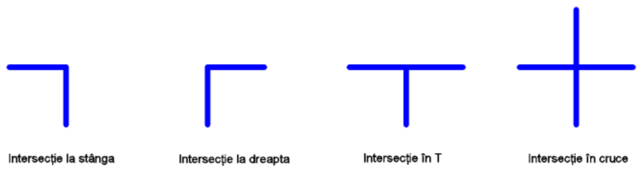

Fig. 6 Types of intersections

The detection of the crossings takes place in the main regulation function *SRA ()* being positioned at their end, according to the instructions of compensating a deviation.

The vehicle detects an intersection, when at least two terminal sensors recorded simultaneously values indicating the presence of black tape under them. The logic of this approach concludes that in this case the vehicle has encountered a route segment transverse to the direction of travel and is therefore an intersection.

If only 2 sensors detected junction terminal available for junctions that allow one option away intersections they are either left or right. If the intersection with left turn is conditional, the terminal of the 2 sensors will be on the left, *s1* and *s2* respectively. If the intersection is turning right, the conditioning is done on the right side terminal sensors, *s5* and *s6*. The conditions mentioned were implemented by two lines of code that sets the IF type two Boolean flags: *left and right* detection as true if the left or right. This approach has the advantage of setting flags that allow control decisions by setting them *true or false*. Flags can be compared and put themselves IF instruction for command decisions and thus help in the implementation of a general algorithm scroll available.

```
If ((s1>=400) && (s2>=400)) {left = true ;}
If ((s6>=450) && (s5>=400) && (s1<=200)) {right = true ;}
```

The T- intersections or cross intersecting segments 3 and 4 track segments and enables the vehicle to traverse them in 2 or 3 directions complicating the maze solving and finding the exit. These intersections require different treatment given the number of options that the vehicle can take. It was necessary to implement an algorithm of scroll in order to ensure that it takes up so that the vehicle does not pass 2 times through the same intersection, and in the same direction and be able to find out from labyrinth into a finite number of passes through each intersection.

Analyzing the conditions for the validity of a scroll efficient algorithm can deduce that it must allow a maximum of 4 passes through an intersection type cross and three passes through an intersection type T. This condition ensures that the vehicle is will return to already visited points and will continue to look for other ways to find exits.

The algorithm based on the "left hand rule" provides such a solution by setting strict scroll. This algorithm requires compliance with four simple rules, valid for any type of intersection encountered:

-If the vehicle can make only Left, it will turn left

-If the vehicle can do and left and right, it will turn left

-If the vehicle can do right or to go forward, it will go forward

-If the vehicle can do strict right, it will turn right

Implementation of these rules was made by making flags and settings, will be considered as finally set flags: right, left, back and depending on them to be on call to scroll individual intersections. At the above program is found as follows:

```
If (left && right) {left (); left=false; right=false ;}
If (left) {left (); left=false; left=false ;}
If (left) {right (); left=false; left=false ;}
If (back) {halt (); left=false; left=false ;}
```

According to the algorithm described, the vehicle will go through the crossing encountered by choosing one of the three directions of movement: forward, left of right. In the moment in which one crossing has been detected, that one is counted by incrementing the *nodes* variable. According to the set flags by the runway algorithm, a decision is taken on the direction on which the progress will continue. The implementing of the decision taken, at the program level, is reflected in a function call that temporarily takes over the control of the vehicle throughout the covering of the crossing, in other words, until the vehicle is centered again on the desired path.

The 2 generic functions named: *left ()* and *right ()* contain code lines that ensure the turning of the vehicle in the desired direction, upon exiting these functions, the vehicle must find itself centered again on the segment of runway towards which the turn has been made, the normal cycle of the program being able to continue through calling the *SRA ()* in the *loop ()* hole, the process being repeated upon detecting another intersection.

## IV. CONCLUSIONS AND CONTRIBUTIONS

The proposed algorithm is general and can be applied to any configuration labyrinth. To be known and possibly easy to use and optimized algorithm was implemented in a program widely used C + +. The algorithm can be applied to curved paths containing junctions with any number of branches. The implementation of the algorithm is simple, it provides only two commands output PWM correlated. The PWM commands are recognized and accepted as input by a large number of programmable logic being used in applications. The validity of the algorithm was tested in an experimental model. The principle of identification of the route by the experimental model avoids recognition of complex images, but uses optical information which is associated binary.

## V. REFERENCES

[1] Jamsa, L. Klander - "Totul despre C si C++". Manualul fundamental de programare in C si C++. Ed.Teora 1999

[2] http://www.w9xt.com/page_microdesign_pt11_opto_inputs.html

[3] http://arduino.cc/en/Tutorial/HomePage

[4] http://en.wikipedia.org/wiki/Electromagnetic_spectrum

[5] http://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf

[6] http://arduino.cc/en/Tutorial/SecretsOfArduinoPWM