Journal of Electrical Engineering, Electronics, Control and Computer Science – JEEECCS, Volume 5, Issue 15, pages 7-16, 2019

Software Implementation of an Autonomous Robot Capable of Detecting and Extinguishing a Flame

Gabriel-Petruţ Bădicioiu, Alexandru Săvulescu
Department of Automatics, Computers and Electronics
Petroleum Gas University of Ploieşti
Ploieşti, Romania
badicioiugabriel@yahoo.com, asavulescu@upg-ploiesti.ro

***Abstract* –In this paper, it is presented, mainly from the point of view of designing the control program, the making of a mobile robot capable of detecting and extinguishing a flame. The program was structured using two main functions: the *avoid*() function that makes the robot to move around, avoiding the obstacles encountered in his path, and the *firefighter*() function that detects and extinguishes the flame. In terms of flame detection, the disturbance (sunlight variation) on the flame sensors has been canceled. There are presented the organization chart of the implemented program, the role of the auxiliary and main functions designed, as well as the main implemented code structures. After uploading the program on the Arduino Microcontroller, it appeared that the robot worked well and accomplished the intended task.**

Keywords-autonomous robot, mobile robot, Arduino programming, robot navigation, ultrasonic sensor, servomotor

I. INTRODUCTION

Autonomous mobile robots nowadays have a strong development with application in various fields: industry, military, environmental protection, research etc. An important role for many robots of this type is to perform certain tasks in extremely dangerous environments for the human operator. The application described in this paper in terms of implemented software has the objective of operating a mobile robot capable of detecting and extinguishing a flame, which means exactly to fulfill a task that can become extremely dangerous for humans.

The software design of autonomous mobile robots is considering the desired degree of autonomy and has as main objectives [1]:

- the robot's knowledge of the operating environment and the dynamic update of any changes;

- determining an optimal trajectory of the robot and interacting with the operating area to move the robot in order to complete its task;

- the permanent localization of the robot;

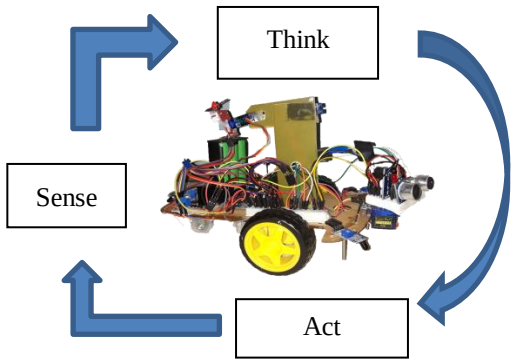

Figure 1. The operating principle of an autonomous robot

- the functional coordination of the robot with other systems in order to fulfill the desired task.

Unlike other finite programs, robotic programs run in an infinite loop, repeating indefinitely. The principle of an autonomous mobile robot is the continuous realization of the **Sense - Think - Act** cycle, as suggestively presented in Fig. 1 for the built robot. This cycle also represents a paradigm of classical artificial intelligence. Inspired by control theory, the **Sense - Think - Act** cycle has as its main objective the continuously attempt to minimize the error between the actual state of the system and its desired state and in the software it is introduced by running in an infinite loop [2].

Fig. 2 shows the interaction principle of an autonomous mobile robot with the environment . It can be observed that the robot:

-**senses,** meaning that it permanently gets information from its sensors;

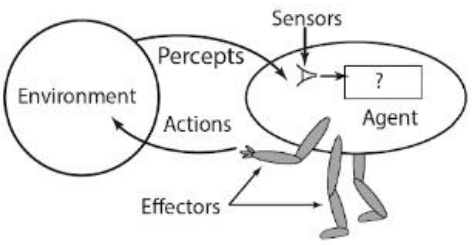

Figure 2. The interaction of the autonomous robot with the environment

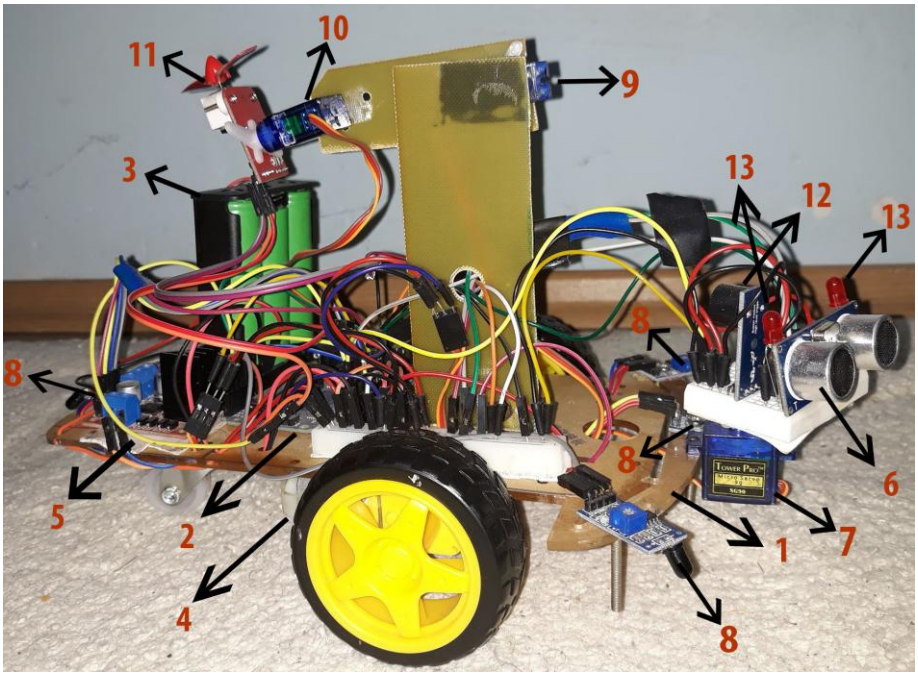

Figure 3. The physically made mobile robot

- **thinks,** meaning that it processes the sensory information received, understands the conditions of the environment where it is located and makes the best decisions on the way it must act, also taking into account its internal intentions;
- **acts** on the devices it is equipped with to achieve its mission. After the actions, changes are made in the environment, the robot notices (senses) the effects of these changes, and the cycle goes on theoretically forever [3].

II. The Hardware Description of the Physically Made Autonomous Robot

In order to achieve its tasks accurately, any autonomous robot must be equipped with [4]:

- a versatile and reliable mechanical system;

- a high-performance actuating system;

- a sensory system capable of sensing different parameters;

- high-performance algorithms and driving systems.

Considering these elements, it was made an autonomous mobile robot capable of detecting and extinguishing a flame, whose image is shown in Fig. 3.

The electrical diagram of how the components were connected and a detailed description of how the robot works are presented in [5] and its operational diagram is shown in Fig. 4.

The robot has two main tasks: *to move* around avoiding all the obstacles encountered in his path, searching for flames in the area intended for protection and *to detect and extinguish these flames.*

The components required for the robot to move, to sense the obstacles and to avoid them are as follows (according to the numbering in Fig. 3):

1 - three wheel robot chassis;

2 - Arduino UNO R3 microcontroller [6];

3 - power supply (12V);

4 - two DC geared motors to actuate the wheels;

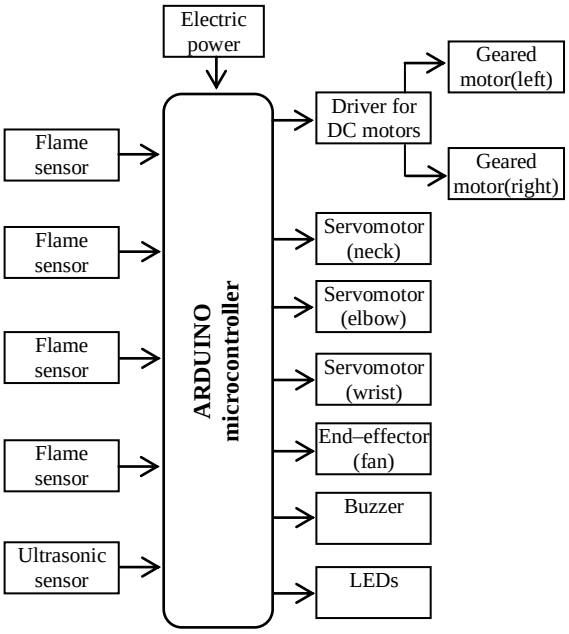

Figure 4. The operational diagram of the robot

5 - L298N driver to control the DC geared motors;

6 - ultrasonic sensor to sense the obstacles;

7 - SG90 servo to rotate the ultrasonic sensor.

For the flame detection, the audible and luminous warning of the human operators and the flame extinguisher, besides the power supply and the Arduino microcontroller, the following components are required (according to the numbering in Fig. 3):

8 - four flame sensors, one in the front, two in the sides and one in the back so it can sense the flame from any angle;

9 - servomotor for the elbow;

10 - servomotor for the wrist;

11 - a fan(the end-effector of the robotic arm);

12 - a buzzer for the audible warning of flame detection;

13 - two LEDs for the luminous warning of flame detection.

III. THE ROBOT'S DRIVING PROGRAM ORGANIZATION CHART

To control the experimental robot, the organization chart shown in Fig. 5 was created.

There are 4 hierarchical levels of an autonomous robot driving system, each level assuming a data processing by a computing system. Based on the

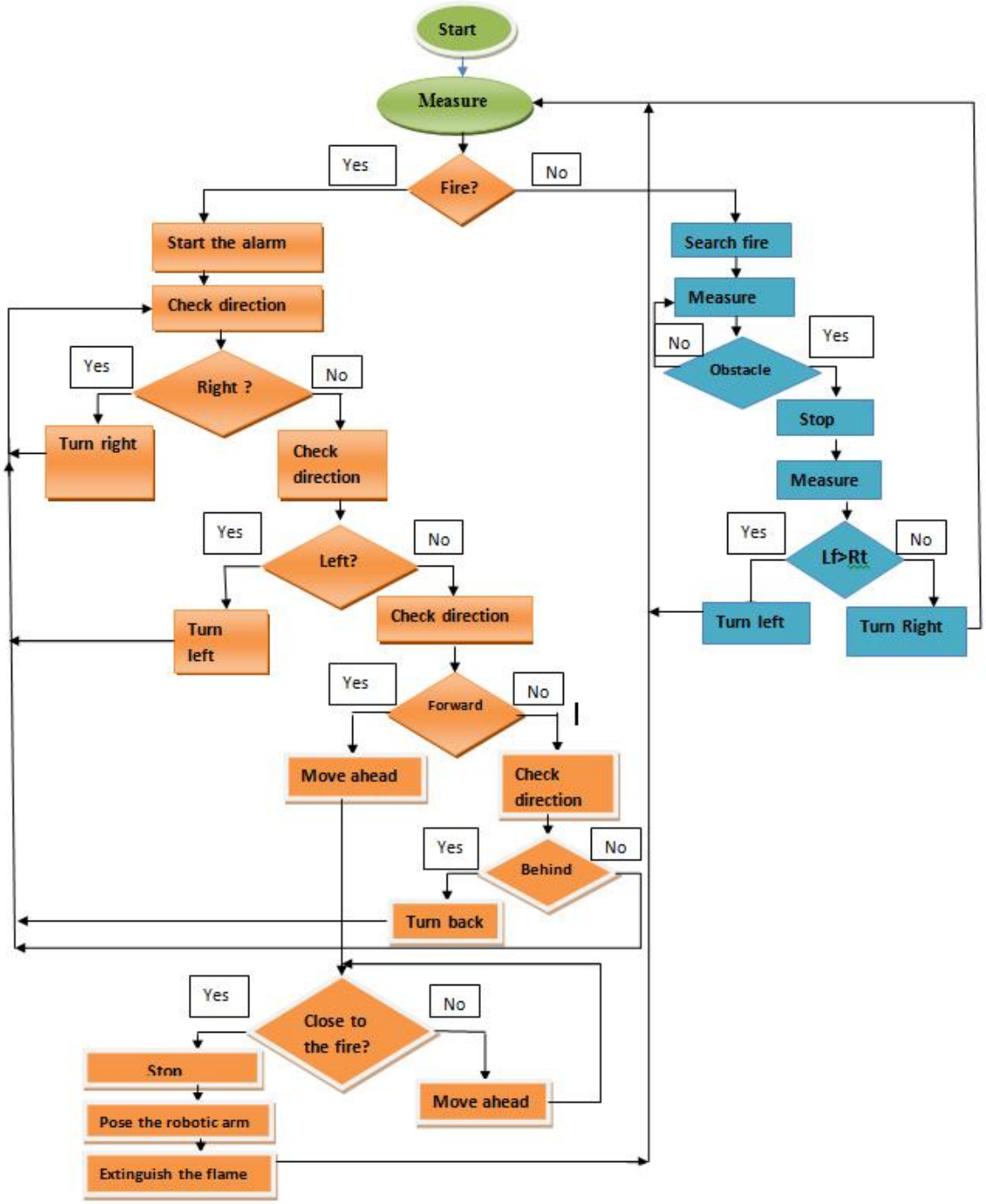

Figure 5. Robot's driving program organization chart

results obtained, the computer (in this case the Arduino microcontroller) generates the command in order to achieve the proposed task [7, 8]. The four levels of the driving system are:

- **the robot control:** this level ensures physical control of the sensors and actuators available on the robot;
- **sensory interpretation:** at this level, the sensors data acquisition and interpretation by the sensor module are realized;
- **the driving level :** at this level, the sensory information is processed and commands are generated in accordance with the task to be performed;
- **the executive level :** it is ensured that each movement is accomplished so that the robot performs its task.

IV. AVOID() FUNCTION PRESENTATION

The program for the robot control is composed of two basic functions called **avoid()** and **firefighter()** working within an infinite loop called **void loop().** The **avoid()** function is represented in the blue part of the organization chart shown in Fig. 5, and the **firefighter()** function is represented by the orange part.

The **avoid()** consists of nine auxiliary functions, as shown:

-five functions which assure the **forward() / backward() / turnRight() / turnLeft() / stop()** movements;

-the **checkDistance()** function measures the distance up to the obstacles;

-the **orientation()** function by which the robot receives the appropriate information in order to choose the right direction to avoid the obstacles;

- the **angle()** function increases the visibility angle of the robot.

A. The movement functions of the robot

The control of the DC motors acting on the side wheels of the robot platform shall be performed via the L298N driver shown in Fig 6. The *in*1 and *in*2 input pins of the driver are connected to the

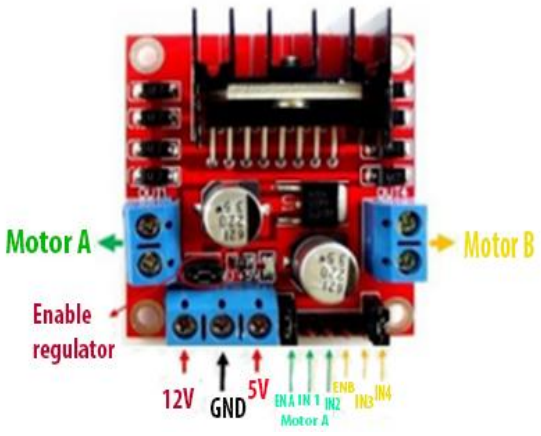

Figure 6. L298N driver to control the DC motors

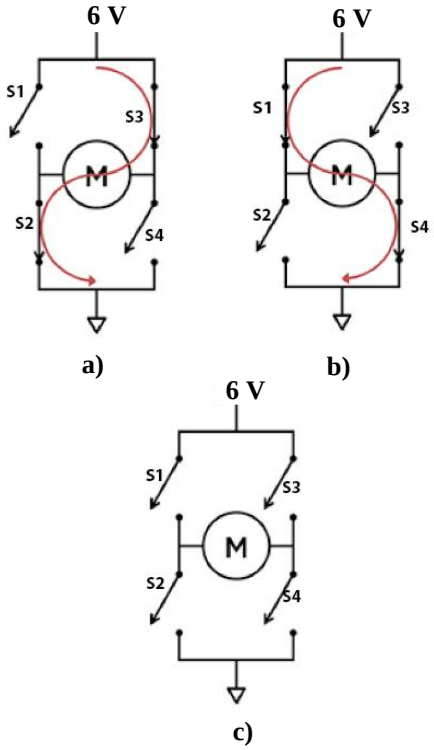

Figure 7. The rotation of the motor depending on the voltage applied a) clockwise rotation b) counterclockwise rotation c) break

microcontroller and correspond to the *out*1 and *out*2 output pins corresponding to the motor A. Similarly, the *in*3 and *in*4 input pins of the driver correspond to the *out*3 and *out*4 output pins connected to the motor B.

The *enA* and *enB* pins enable and disable the corresponding motor. They are normally used to control the motor speed via a PWM signal. The command of the two motors using the L298N driver (known in the literature as a dual H-bridge) is based on the H-bridge principle. An H-bridge is an electronic circuit used to reverse the polarity of a voltage applied to a load (a DC motor in this case), as shown in Fig. 7, which leads to three motor operating stages: forward, backward and breaking. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards [9].

According to Fig. 7a, if the S3 and S2 switches are closed and the S1 and S4 switches remain open, a positive voltage is applied to the motor M and it will rotate clockwise. If the S3 and S2 switches are opened and the S1 and S4 switches are closed, as shown in Fig. 7b, the voltage is reversed and the motor will turn counterclockwise. To break the motors, all the switches must be opened so the motor is no longer powered (Fig. 7c).

The **forward()** function provides the clockwise rotation of both DC motors so that the robot can move forward. The right motor is controlled on the *in*1 and *in*2 pins of the driver and the left motor is controlled on the *in*3 and *in*4 pins. To turn the motors clockwise, a HIGH signal is sent via the *digitalWrite* command to the *in*1 and *in*3 pins and similarly, the *in*2 and *in*4 pins are put on LOW.

```
void forward()
{
//right motor
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
//left motor
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
}
```

The **backword()** function works the same as the **forward()** function, but this time *in*1 and *in*3 pins will be set on LOW and *in*2 and *in*4 will be set on HIGH (the polarity is reversed and the motor rotates in the opposite direction).

In order to make the robot turn right, it is necessary to rotate the right motor backwards while the left motor is running forward. For the left turn, it is necessary to rotate the right motor forward and the left motor backwards. For this purpose, the **turnRight()** and **turnLeft()** functions have been created on the same principle as the one previously described.

Based on the same principle, the **stop()** function has been created in order to stop the robot. Within this function**,** all the pins are set either LOW or HIGH.

*B. The **checkDistance( )** function*

The measurement of the distance to the obstacle is done by the **checkDistance()** function which controls the ultrasonic sensor (HC-SR04) operation.

The ultrasonic sensor has 4 pins: *vcc*, *echo*, *trig* and *gnd*. When a voltage signal is applied to the *trig* pin, the electroacoustic transducer is powered, emitting in space a cycle of 8 ultrasonic bursts according to Fig. 8 and the internal timer of the transducer start to measure. At the *echo* pin, a HIGH signal is emitted that returns the sound propagation time from the sensor to the obstacle and back [10].

In Fig. 9, it is shown a principle drawing for the ultrasonic sensor operation. The time variable *t* measured from the moment of signal transmission

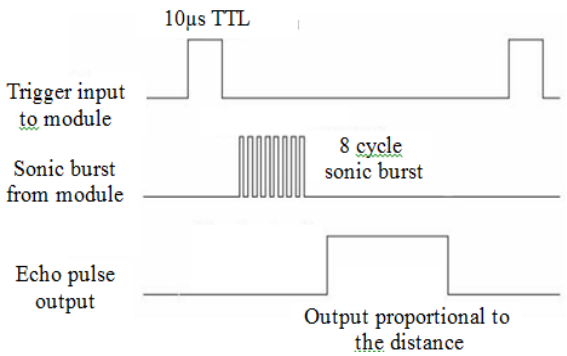

Figure 8. Ultrasonic sensor signals

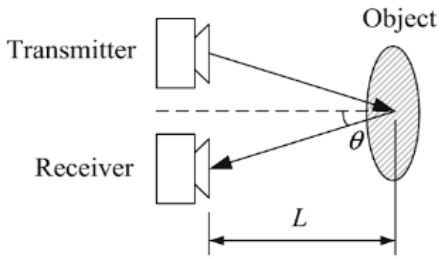

Figure 9. The principle of the ultrasonic sensor

until the signal return is used. The distance *L* to a possible obstacle is calculated with the following relation:

$$L = v \cdot t \cdot (\cos \theta) / 2 \qquad (1)$$

where *v* represents the velocity of the ultrasonic pulses and θ is the angle of reflection. If the transmitter is close to the receiver, as it is for the sensor used (HC-SR04), then *cos* θ = 1 (Fig. 9).

In order to emit the ultrasonic pulses, a voltage pulse with an amplitude of 5V and a duration of 10µs must be applied to the trigger pin. It will start with the *trig* pin on LOW, wait for the signal to settle, then the pin will be set on HIGH, wait 10 µs and then the pin will be set again on LOW, as seen in the following code sequence:

```
digitalWrite(trig, LOW);
delayMicroseconds(2000);
digitalWrite(trig, HIGH);
delayMicroseconds(10);
digitalWrite(trig, LOW);
```

After the 8 ultrasonic pulses have been created, the *pulseIn* command is used to search a HIGH signal at the *echo* pin, a signal that returns in µs the propagation time of the sound from the sensor to the obstacle and back. This value is stored in the *time* variable and is converted from microseconds in hours and the distance is converted from km to cm.

```
time=pulseIn(echo, HIGH);
time=(time/1000000.)/3600.;
distance=((speedOfSound*time)/2) * 100000;
```

*C. The **orientation( )** function*

If it detects an obstacle in front of it, the robot stops and orients itself using the **orientation()** auxiliary function. Once it stops, the robot measures the distance to the right and to the left, compares the two measured values and then turns in the direction where it has more space. For the servo that turns the ultrasonic sensor, the *neck* object was created.

The *angle* values (in degrees) with witch the servo rotates in each direction will be stored in the *lookRight* and *lookLeft* variables. In the *checkRight* and *checkLeft* variables is stored the measured distance for each direction.

***Orientation*() function code**

```
Void orientation()
{ neck.write(lookRight); //ultrasonic sensor oriented to the right
checkDistance(); //measure distance to the right
delay(checkDelay); // time required to measure
checkRight1=distance; //store the distance in the checkRight1 variable
neck.write(lookRight+30); // rotate the servo to the right by 30 degrees
checkDistance(); //call the checkDistance function to measure the distance
delay(checkDelay); //time required to measure
checkRight2=distance; //store the distance in the checkRight2 variable
//compare checkRight1 to checkRight2 and checkRight gets the lowest value
if(checkRight1>checkRight2)
    checkRight=checkRight1;
    else
    checkRight=checkRight2;
//the code is repeated for the left side, but instead of neck.write(checkRight+30), use neck.write(checkLeft-30)
//compare checkLeft1 to checkLeft2 and checkLeft gets the lowest value
}
// void orientation
```

*D. The **angle( )** function*

The **angle()** function has been created to increase the robot's visibility angle. The robot constantly rotates the ultrasonic sensor in every direction by 30 degrees. It knows in which direction to look by checking the value of the *move* variable that changes its value after each rotation. For example, if *move*=1, the robot knows he has to look to the right, measure, reset the timer after 250 ms and assing to the *move* variable the value of 2. This way it knows it has to look ahead and so on. The **angle()** function is called as long as there is no obstacle, so only if the distance is greater that 20 cm [11].

The robot switches his aim depending on the value of the *move* variable.

```
Void angle()
{ static int move=1; //look to the right
  if(timer>250 && distance>20 && move==1)
{neck.write(lookRight+60) //look right
move=2; // assign the value of 2 to the move variable
timer=0; // reset timer
return; //return is used so that it doesn't go to the next condition until the next cycle
} // if( timer>250 && distance>20 && move==1)
if( timer>250 && distance>20 && move==2)
{
neck.write(lookAhead);
move==3;
timer=0;
return;
} // if( timer>250 && distance>20 && move==2)
if( timer>250 && distance>20 && move==3)
{
neck.write(lookLeft-60);
move=4;
timer=0;
return;
} // if( timer>250 && distance>20 && move==3)
if( timer>250 && distance>20 && move==4)
{
neck.write(lookAhead);
move=1;
timer=0;
return;
} // if( timer>250 && distance>20 && move==4)
}
//void angle
```

*E. Description of the main function **avoid()***

Using these auxiliary functions, the main **avoid()** function was created. This function first calls the **angle()** function to increase the visibility angle of the ultrasonic sensor. Then the **checkDistance()** function is called to check the distance. Imposing the condition that if the distance is less than 20 cm, the robot stops, measures the distance in each direction, compares the two measured values and turns in the direction where it has more space.

```
Void avoid()
{
angle(); //call the angle function to increase the visibility angle
checkDistance(); //measure the distance
```

```
if(distance<20) // if the distance is less than 20cm the robot stops and turns
{  backward();
delay(350);
stop();
orientation();
//compare the two measured values and turn in the direction where it has more space
if(checkLeft<checkRight)
{
neck.write(lookAhead);
 turnRight();
delay(turnDelay);
}
// if(checkLeft<checkRight)
      else
{gat.write(lookAhead);
turnLeft();
delay(turnDelay);
} //else
} //if(distance<20)
else
forward();
timer++;
 }
//void avoid()
```

V. DESCRIPTION OF **FIREFIGHTER()** FUNCTION

The main function *firefighter()* has four auxiliary functions:

-**detectFire()** function senses the presence of the fire;

-**fanOn()** and **fanOff()** functions to start respectively stop the fan;

- **adapt()** function that cancels the influence of the sunlight on the sensors.

A. ***detectFire( )*** *function*

Using the **detectFire()** function, the values of the four flame sensors are read using the *analogRead* command and stored in a variable.

```
Void detectFire()
{
right=analogRead(rightSensor); //reads the value of the right flame sensor and store it in the right variable
front=analogRead(frontSensor);
left=analogRead(leftSensor);
back=analogRead(backSensor);
}
// void detectFire()
```

B. ***fanOn( )*** *and* ***fanOff()*** *functions*

The **fanOn()** function is called when the robot reaches a reduced, predetermined distance from the flame (it knows it has come close to the flame by comparing the sensor values with a reference value from the memory). Then, the *elbow* servo is actuated using the command *elbow.write()* (*elbow* is the object created for this servo), directing the end-effector (the fan) towards the flame; the fan is turned on by sending a HIGH signal to the *ina* pin and a LOW signal to the *inb* pin. The end-effector performs a curved motion with the help of the servo named *wrist*. To achieve this motion, two *for* loops were used, where *i* represents the current position of the servo, it compares this variable to 180 (a servo rotates to 180 degrees) and increments by one after each iteration while *i*<=180. To get back at 0 degrees, the variable *i* decreases by one as long as *i*>=0. All this is executed in a repetitive statement *do*{} *while*(). The *while* condition is that the front flame sensor value drops below the reference value and has the lowest value compared to the other flame sensors (if, for example, the value of the right sensor is lower than the front sensor, although the value of the front sensor is less than the reference value, it will turn right until the front sensor has the lower value, which means the robot is facing the flame).

fanOn() function code

```
void fanOn()
{elbow.write(145); //direct the end-effector to the flame
delay(500);  // wait until the elbow servo rotates
//turn on the fan by sending a HIGH signal to ina and a LOW signal to inb
digitalWrite(ina, HIGH);
digitalWrite(inb, LOW);
// perform the curved motion and start the alarm
do
{  for(int i=45;i<=180;i++)
{ wrist.write(i);
delay(17);
if(i==45)
digitalWrite(buzzer, HIGH);
if(i==90);
   digitalWrite(buzzer, LOW);
} // for(int i=45;i<=180;i++)
```

```
for(int i=180; i>=0; i--)
{wrist.write(i);
delay(17);
if(i==180)
digitalWrite(buzzer, HIGH);
if(i==90)
      digitalWrite(buzzer, LOW);
if(i==45)
      digitalWrite(buzzer, HIGH);
} // for(int i=180; i>=0; i--)
detectFire();
} //do
while(reference>=600 && reference<=950 && front<350 && front<right && front<left || reference<600 && reference>350 && front<250 && front<right && front<left || reference<350 && front<150 && front<right && front<left || reference>950 && reference< 500 && front<right && front<left);
}
//void fanOn()
```

After the flame has been extinguished (the robot knows that the flame is extinguished when the sensors values rise above the reference), the **fanOff** function turns off the fan by sending a LOW signal to both ina and inb pins and poses the elbow and the wrist in the initial position.

fanOff() function code

```
void fanOff()
{
//turn off the fan
digitalWrite(ina, LOW);
digitalWrite(inb, LOW);
//put the arm in the initial position and turn off the alarm
for(int i=45;i<=90;i++)
{ wrist.write(i);
delay(17); }
// for(int i=45;i<=90;i++)
digitalWrite(buzzer, LOW); }
//void fanOff()
```

The *reference* variable value is calculated using the **setRef()** function. This function first checks the flame sensors values by calling the **detectFire()** function, then compares the values and takes the lowest value as reference.

setRef() function code

```
void setRef()
{
detectFire();
if(right<front && right<left)
{     reference=right-120;
if(right<100)
      reference=right-10;
}  // if(right<front && right<left)
if(front<right && front<left)
{     reference=front-120;
if(front<100)
      reference=front-10;
}  // if(front<right && front<left)
if(left<right && left<front)
{     reference=left-120;
if(left<100)
      reference=left-10;
}  //  if(left<right && left<front)
refBack=reference-180;
}
//void setRef()
```

C. The ***adapt()*** *function*

Because the sunlight disturbs the flame sensors, the robot needs a *reference* value that adapts to that light. For this, the **adapt()** function was created. As long as there is no fire (the sensors values are over the reference), the reference is updated in real time. For example, if the robot moves into an area where the sun is brighter, the reference will be updated. Updating the reference is permanently made as long as the values of all sensors are above the reference, when the values of all the sensors are suddenly decreasing (the robot has passed into a bright area) and when all the values are suddenly rising (the robot has gone to a shaded area).

The adapt() function code

```
void adapt()
{
if(left>reference && front>reference && right>reference) //if the values are above the reference update the reference in real time
{  if(right<front && right<left)
reference=right-120;
if(left<front && left<right)
reference=left-120;
if(front<right && front<left)
reference=front-120;
refBack=reference-170;
```

```
} // if(left>reference && front>reference && right>reference)

if(right<reference && left<reference && front<reference) //if all the values are dropping below the reference

{ if(right>front && front>left) //if right has the greater value

{if(right-front<=200 && right-left<=200) //and the difference between right and the other sensors is less than 200

setRef(); //update the reference

} // if(right>front && front>left)

if(front>right && front>left) //if front has the greatest value

{ if(front-right<=200 && front-left<=200) //and the difference between front and the other sensors is less than 200

setRef(); //update the reference

} // if(front>right && front>left

if(left>front && left>right) //if left has the greater value

{ if(left-front<=200 && left-right<=200) //and the difference between left and the other sensors is less than 200

setRef(); //update the reference

} // if(left>front && left>right)

if(reference-front<=100 || reference-right<=100 || reference-left<=100) //if the difference between reference and the sensor that drops below the reference is less than 100

setRef(); //update reference

} // if(left>reference && front>reference && right>reference)

if(right>reference+200 && front>reference+200 && left>reference+200) //if all the values are rising (the robot got in the shaded area)

setRef(); //update the reference

if(reference<100)

setRef(); //if the reference is lower than 100 update

if(front<100 && left<100 || front<100 && right<100 || front<100 && left<100 && right<100)

setRef(); }

//void adapt()
```

D. The ***firefighter()*** function

The robot is being guided to the flame depending on the flame sensors values. For example, if the right sensor drops below the reference value and it has the lowest value, the robot triggers the alarm and turns right until he is facing the fire (while turning, he calls the **detectFire()** function and compares the right sensor value with the front sensor value)

```
if(right<reference && right<front && right<left && right<back)

{

turnRight();

digitalWrite(buzzer, HIGH);

delay(250);

digitalWrite(buzzer, LOW);

delay(250);

detectFire();

}
```

When the value of the front sensor drops below the reference value and has the lowest value, the **forward()** function is called so the robot moves toward the flame. As he approaches the fire, the value of the front sensor decreases and when it reaches a value of less than 100, it stops, calls the **fanOn()** function to extinguish the flame and then it stops the fan by calling the **fanOff()** function.

```
if(front<reference && front<100 && front<right && front<left)

{

stop();

fanOn();

fanOff();
}
```

VI. The Main Function Void Loop ()

The main function, **void loop()**, is an infinite loop. This is where the values of the sensors are constantly checked by calling the **detectFire()** function and where the robot adapts to the sunlight using the **adapt()** function [12].

If the sensor values are above the reference (there is no fire), the **avoid()** function is called (the robot is looking for fire avoiding obstacles), otherwise the **firefighter()** function is called.

```
void loop()

{

detectFire(); //check the flame sensors values

adapt(); //adapt to the sunlight

firefighter(); //call the firefighter function

if(right>reference && front>reference && left>reference && back>reference) //if the values are above the reference (there is no fire), call the avoid() function (search fire while avoiding obstacles)

avoid();

} //void loop()
```

CONCLUSIONS

One of the key benefits of using robots is that they can be used to perform tasks in extremely dangerous environments for human operators. In this category of robots, it is also included the autonomous mobile robot capable of detecting and extinguishing a flame described in this paper, especially in terms of the execution of the control program.

Because it is an experimental robot, pretty simple and inexpensive devices have been used for the hardware implementation, wanting actually to achieve an experimental model to test the development of the most comprehensive software to ensure the accomplishment of the proposed tasks. This structured presentation of the software can help the reader understand the operation of such robots, as well as develop the ability to perform similar functions as part of the robot control programs.

The program is structured using two main functions: the **avoid()** function that moves the robot around, avoiding the obstacles encountered in his path, and the **firefighter()** function to detect and extinguish the flame. An important improvement of the **firefighter()** function is cancelling the sunlight variation effect on the flame sensors by permanently updating the value of the *reference* variable, which is used to determine whether there is a fire or not.

After the implementation of the program, it appeared that the robot achieves his mission of detecting and extinguishing a flame, even if the source of the fire is moving during the robot's movement or if the outer light or shade conditions are extremely variable. The flame detection and extinguishing system can be further improved by using smoke detectors, by increasing the mobility of the robotic arm or by using a water pump instead of the fan.

REFERENCES

[1] W. Yang, "Autonomous robots research advances", Nova Science Publishers Inc., New York, 2008.

[2] H. Hexmoor, "Essential principles for autonomous robotics", Morgan & Claypool Publishers, 2013

[3] S. Tzafestas, "Introduction to mobile robot control", Elsevier, 2013.

[4] H. Asama, T. Fukuda, T. Arai, I. Endo, "Distributed autonomous robotic systems 2", Springer Science, 1996.

[5] G.P. Bădicioiu, A. Săvulescu, "Hardware implementation of a robot capable of detecting and extinguishing a flame", unpublished

[6] S. Barret, "Arduino microcontroller processing for everyone", Third Edition, Morgan & Claypool Publishers, 2013

[7] D. Brugali, "Software engineering for experimental robotics", Springer – Verlag Berlin Heidelberg, 2007

[8] M. Margolis, "Make an Arduino-controlled robot", O'Reilly Media Inc., 2013

[9] I. Noda, N. Ando, D. Brugali, J. Kuffner, "Simulation, modeling and programming for autonomous robots", Springer – Verlag Berlin Heidelberg, 2012

[10] A. Whitbrook, "Programming mobile robots with aria and player. A guide to C++ object oriented control", Springer – Verlag London Limited, 2010

[11] L. Joseph, "Learning Robotics Using Python", Packt Publisher, Birmingham, 2015

[12] J. Blum, "Exploring Arduino: tools and techniques for engineering wizardry", John Wiley & Sons Inc., Indianopolis, 2013