ECAM Strasbourg-Europe — Promotion 2018

Embedded Systems Report

Hexapod Automation

Group:
Nicolas Douard
Alexis Thézé
Promotion 2018

Supervisor:
Mr. Yves Gendrault

Project beginning:

January 9, 2017

Project end:

January 16, 2017

Version 2 – January 16, 2017

Contents

List of Figures

List of Tables

1 Introduction

The goal of this project is to develop a software solution for an existing hexapod robot allowing it to move on its own and avoid obstacles in a simple fashion.

As the robot is already built, the first mandatory step is to understand its mechanical and electronic principles, manage I/O using an Arduino board, power it and develop simple motion patterns.

2 Design problem

The following motion patterns should be implemented:

- Move forward
- Rotate on itself
- Detect and avoid obstacles

Once done, a simple behavior where the robot walks forward only to stop when it gets close to an obstacle in order to avoid it shall be implemented.

This can be decomposed as:

1. The robot starts moving forward
2. As the robot encounters an obstacle, it rotates on itself in order to change its orientation
3. Once the front of the robot is not facing the obstacle anymore, the robot continues to move forward

3 Design solution evaluation

3.1 Motion patterns

3.1.1 Rotation

A functional rotation pattern was implemented as detailed in the next figure.

Rotation is triggered though the rotate() function which makes the robot rotate on itself **by a few degrees**. This means that this function **won't produce a 90° rotation** and must be called repeatedly in order to face a different direction.

This makes prefect sense as the function will be used when the robot faces an obstacle: it simply needs to rotate by a few degrees **repeatedly** until it stops facing said obstacle.

This implementation is satisfactory as rotate() is very fast to execute relative to the motion speed of the robot; it can therefore be used without using interrupts or a millis()-based flow.

rotate() is available in the full code (see code annex).

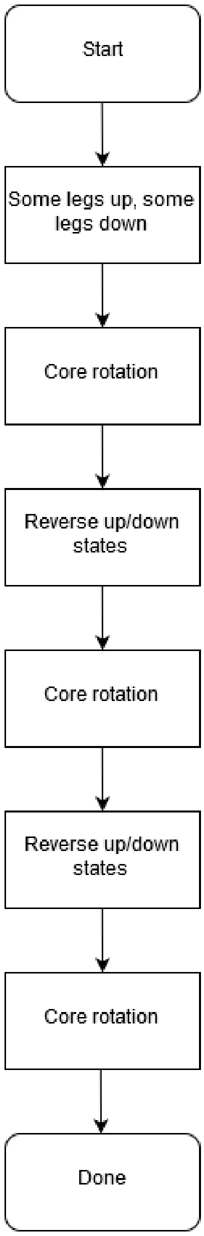

Figure 1: Rotation flowchart

3.1.2 Forward motion

Forward motion pattern was empirically implemented and consists into a modified version of the rotation code using additional log rotations defined in dir1() and dir2() functions.

Just like the rotate() function, the robot will only move of a few centimeters for each moveFwd() call. It can therefore be used in a loop without interrupts or millis().

moveFwd(), dir1() and dir2() are available in the full code (see code annex); code is commented.

3.2 Obstacle sensor

The following sensor will be used:

- Model: HC-SR04
- Type: ultrasonic sensor
- Measurement angle: 15°
- Measurement resolution: 0.3 *cm*
- Distance de mesure: 2 *cm* à 400 *cm*

Parameter	**Min**	**Typ.**	**Max**	**Unit**
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz

Table 1: HC-SR04 power specifications

This sensor can be powered through the Arduino Mega board already installed on the robot. We simply had to solder it, wire it and mount it on top of the robot.

```
float getUS() {
  long duration;
  float distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); // trig 10ms HIGH
  digitalWrite(trigPin, LOW);

  // echo calculation
  duration = pulseIn(echoPin, HIGH);
  // distance proportional to the output duration
  distance = duration*340/(2*10000);  // sound speed

  // out of range
  if ( distance <= 0){
        Serial.println("Out of range");
  }
  else {
        Serial.print(distance);
        Serial.print(" cm ");
        Serial.print(duration);
        Serial.println(" ms");
        return(distance);
  }
}
```

3.3 Power source

The robot requires a power source between 7 V and 25 V as imposed by the Texas Instruments $\mu A7805C$ power regulators. Ouput current for each regulator should not exceed 1.5 A.

We wanted to power the robot using two regulated power supplies in parallel in order to reach higher current (each power supply being limited to 1.0 A). This, however was not sufficient to have the robot handling its own weight. For our tests, we limited the required current by attaching the robot to a bar in order to have it not touching the ground.

One solution would be to use a LiPo battery instead which is capable of providing higher currents.

3.4 I/O setup

Motor	Pin
0.0	46
0.1	4
0.2	26
1.0	48
1.2	5
1.2	32
2.0	50
2.1	6
2.2	9
3.0	52
3.1	7
3.2	36
4.0	42
4.1	2
4.2	22
5.0	44
5.1	3
5.2	24

Table 2: Servomotors PWM outputs

Allocation	Pin	Mode
US trigger	15	Output
US echo	14	Input

Table 3: US sensor I/O

4 Final design overview

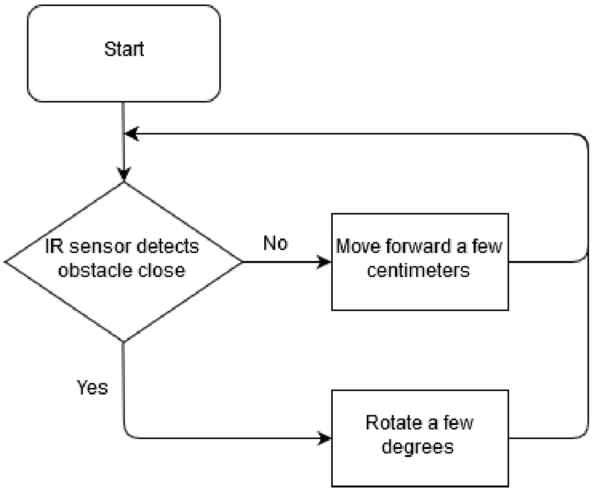

Figure 2: General behavior flowchart

```
void loop() {
  // check distance
  if (getUS() > 35.0f){
        // move 'forward' if object not in front
        // moveFwd() is very quick to execute once so doing
        // it w/o interrupts or using millis() works properly
        // few centimeters per moveFwd() call
        moveFwd();
        delay(100);
  }
  else{
        // default steady position
        resetPos();
        delay(100);
        // rotate until not facing obstacle
        // few degrees per rotate() call
        rotate();
  }
}
```

5 Conclusion

This project led to a functional automated hexapod robot. Compared to a vehicle where it would take only limited amounts of time, motion implementation is critical for such a robots as used patterns are quite complicated and require many tests before reaching satisfactory behavior. This took most of our time.

The current automated behavior is quite simple and only relies on one imput sensor. Improvements could include developing more advanced interactions relying, for example, on computer vision (Raspberry Pi with camera module running OpenCV for example). This could allow the robot to recognize patterns or even people and react according to it [1].

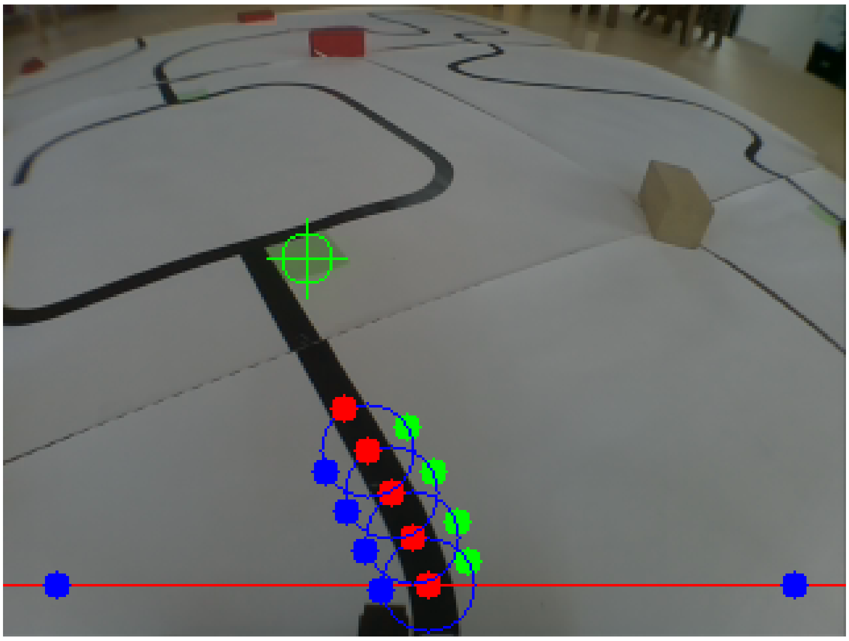

Figure 3: OpenCV example

References

[1] Raspberry Pi. *An image-processing robot for RoboCup Junior*. [Online]. https://www.raspberrypi.org/blog/an-image-processing-robot-for-robocup-junior/.

Annexes

Code

Full code is available here: GitHub

```
#include "Servo.h"

Servo mot[6][3];
int offset[6][3];
int md; // delay between 'movements'

// US definition
const int trigPin = 15;
const int echoPin = 14;

void setup() {
  Serial.begin (9600);
  //US
  pinMode(trigPin, OUTPUT);  // trig is an output
  pinMode(echoPin, INPUT);   // echo is an input

  md = 200; // delay between 'movements'

  mot[0][0].attach(46);
  mot[0][1].attach(4);
  mot[0][2].attach(26);

  offset[0][0] = 0;
  offset[0][1] = 1;
  offset[0][2] = 2;


  mot[1][0].attach(48);
  mot[1][1].attach(5);
  mot[1][2].attach(32);

  offset[1][0] = 0;
  offset[1][1] = 14;
  offset[1][2] = 7;

  mot[2][0].attach(50);
  mot[2][1].attach(6);
  mot[2][2].attach(34);

  offset[2][0] = 0;
  offset[2][1] = 0;
  offset[2][2] = 9;

  mot[3][0].attach(52);
  mot[3][1].attach(7);
  mot[3][2].attach(36);

  offset[3][0] = 0;
  offset[3][1] = 12;
  offset[3][2] = 3;

```

```
  mot[4][0].attach(42);
  mot[4][1].attach(2);
  mot[4][2].attach(22);

  offset[4][0] = 0;
  offset[4][1] = 10;
  offset[4][2] = 7;

  mot[5][0].attach(44);
  mot[5][1].attach(3);
  mot[5][2].attach(24);

  offset[5][0] = 0;
  offset[5][1] = 1;
  offset[5][2] = 0;
}

void loop() {
  // check distance
  if (getUS() > 35.0f){
        // move 'forward' if object not in front
        // moveFwd() is very quick to execute once so doing
        // it w/o interrupts or using millis() works properly
        // few centimeters per moveFwd() call
        moveFwd();
        delay(100);
  }
  else{
        // default steady position
        resetPos();
        delay(100);
        // rotate until not facin obstacle
        // few degrees per Rotate() call
        rotate();
  }
}

float getUS() {
  long duration;
  float distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); // trig 10ms HIGH
  digitalWrite(trigPin, LOW);

  // echo calculation
  duration = pulseIn(echoPin, HIGH);
  // distance proportional to the output duration
  distance = duration*340/(2*10000);  // sound speed

  // out of range
  if ( distance <= 0){
        Serial.println("Out of range");
  }
  else {
        Serial.print(distance);
        Serial.print(" cm ");
```

```
        Serial.print(duration);
        Serial.println(" ms");
        return(distance);
  }
}

// default steady position
void resetPos(){
  for (int i=0; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+34);
        mot[i][2].write(offset[i][2]+40);
  }
  delay(md);

}

// not used - kept for future additions
void nicePose(){
         for (int i=1; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+100);
        mot[i][2].write(offset[i][2]+110);
  }
}

// not used - kept for future additions
void fwdPose(){
  for (int i=1; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+34);
        mot[i][2].write(offset[i][2]+40);
  }
}

// rotates on itself
// few degrees per Rotate() call
void rotate(){
   Serial.write("Rotate"); // avoids obstacle, rotates on itself

   mot[0][2].write(45);
   mot[2][2].write(45);
   mot[4][2].write(45); // leg low
   mot[0][1].write(150);
   mot[2][1].write(150); // middle leg
   mot[4][1].write(150); // 3 first legs up and done

   delay(500);
   mot[0][0].write(0);
   mot[2][0].write(0);
   mot[4][0].write(0);// core rotation and done

   delay(500);

   mot[0][2].write(135);
```

```
  mot[2][2].write(135);
  mot[4][2].write(135);
  mot[0][1].write(30);
  mot[2][1].write(30);
  mot[4][1].write(30);// lower 3 first legs

  delay(500);
  mot[1][2].write(45);
  mot[4][2].write(45);
  mot[5][2].write(45);
  mot[1][1].write(150);
  mot[3][1].write(150);
  mot[5][1].write(150);// raise 3 next legs

  delay(500);
  mot[1][0].write(0);
  mot[3][0].write(0);
  mot[4][0].write(0);// 2nd rotation

  delay(500);
  mot[1][2].write(135);
  mot[3][2].write(135);
  mot[5][2].write(135);
  mot[1][1].write(30);
  mot[3][1].write(30);
  mot[5][1].write(30);
  delay(500); // lower legs

  mot[0][0].write(90);
  mot[1][0].write(90);
  mot[2][0].write(90);
  mot[3][0].write(90);
  mot[4][0].write(90);
  mot[5][0].write(90); // final rotation and end
}

// move straight forward
void moveFwd(){
        for (int i=0; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+34);
        mot[i][2].write(offset[i][2]+40);
  }
  delay(md);

   for (int i=1; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+100);
        mot[i][2].write(offset[i][2]+110);
  }
  delay(md);
  dir1();
  delay(md);

  for (int i=1; i<6; i+=2)
  {
```

```
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+34);
        mot[i][2].write(offset[i][2]+40);
  }
  delay(md);
  for (int i=1; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+100);
        mot[i][2].write(offset[i][2]+110);
  }
  delay(md);
  dir2();
  delay(md);
  for (int i=1; i<6; i+=2)
  {
        mot[i][0].write(90);
        mot[i][1].write(offset[i][1]+34);
        mot[i][2].write(offset[i][2]+40);
  }
  delay(md);

}

// used in 'moveFwd' as pose - legs slight rotation
void dir1(){
  // full
   for (int i=0; i<6; i+=1)
        {
          mot[i][0].write(60);
        }
   // 'special'
   for (int i=0; i<6; i+=2)
        {
          mot[i][0].write(120);
        }
}

// used in 'moveFwd' as pose
// legs slight rotation in opposite direction relative to dir1()
void dir2(){
  // full opposite
   for (int i=0; i<6; i+=1)
        {
          mot[i][0].write(120);
        }
   // 'special' opposite
   for (int i=0; i<6; i+=2)
        {
          mot[i][0].write(60);
        }
}
```