



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

1	Introduction	3
2	Design problem	3
3	Design solution evaluation	3
3.1	Motion patterns	3
3.1.1	Rotation	3
3.1.2	Forward motion	4
3.2	Obstacle sensor	5
3.3	Power source	6
3.4	I/O setup	6
4	Final design overview	7
5	Conclusion	8
	References	9
	Annexes	10
	Code	10

List of Figures

1	Rotation flowchart	4
2	General behavior flowchart	7
3	OpenCV example	8

List of Tables

1	HC-SR04 power specifications	5
2	Servomotors PWM outputs	6
3	US sensor I/O	6

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 through 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 perfect 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).

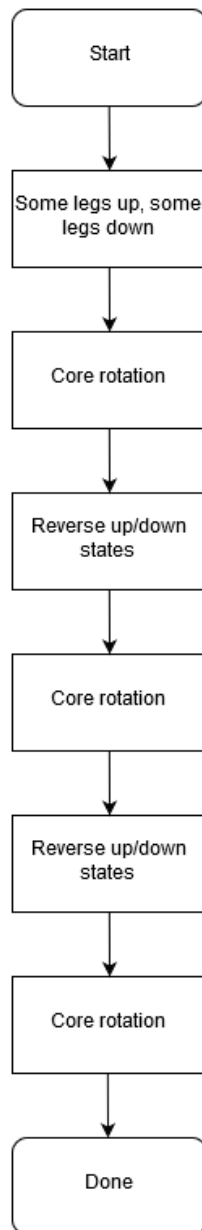


Figure 1: Rotation flowchart

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.

```
1 float getUS() {
2   long duration;
3   float distance;
4   digitalWrite(trigPin, LOW);
5   delayMicroseconds(2);
6   digitalWrite(trigPin, HIGH);
7   delayMicroseconds(10); // trig 10ms HIGH
8   digitalWrite(trigPin, LOW);
9
10  // echo calculation
11  duration = pulseIn(echoPin, HIGH);
12  // distance proportional to the output duration
13  distance = duration*340/(2*10000); // sound speed
14
15  // out of range
16  if ( distance <= 0){
17    Serial.println("Out of range");
18  }
19  else {
20    Serial.print(distance);
21    Serial.print(" cm ");
22    Serial.print(duration);
23    Serial.println(" ms");
24    return(distance);
25  }
26 }
```

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. Output 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

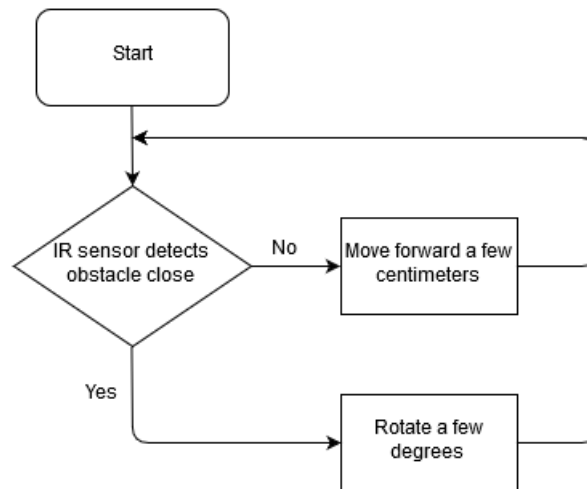


Figure 2: General behavior flowchart

```
1 void loop() {
2   // check distance
3   if (getUS() > 35.0f){
4       // move 'forward' if object not in front
5       // moveFwd() is very quick to execute once so doing
6       // it w/o interrupts or using millis() works properly
7       // few centimeters per moveFwd() call
8       moveFwd();
9       delay(100);
10  }
11  else{
12      // default steady position
13      resetPos();
14      delay(100);
15      // rotate until not facing obstacle
16      // few degrees per rotate() call
17      rotate();
18  }
19 }
```

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 input 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].

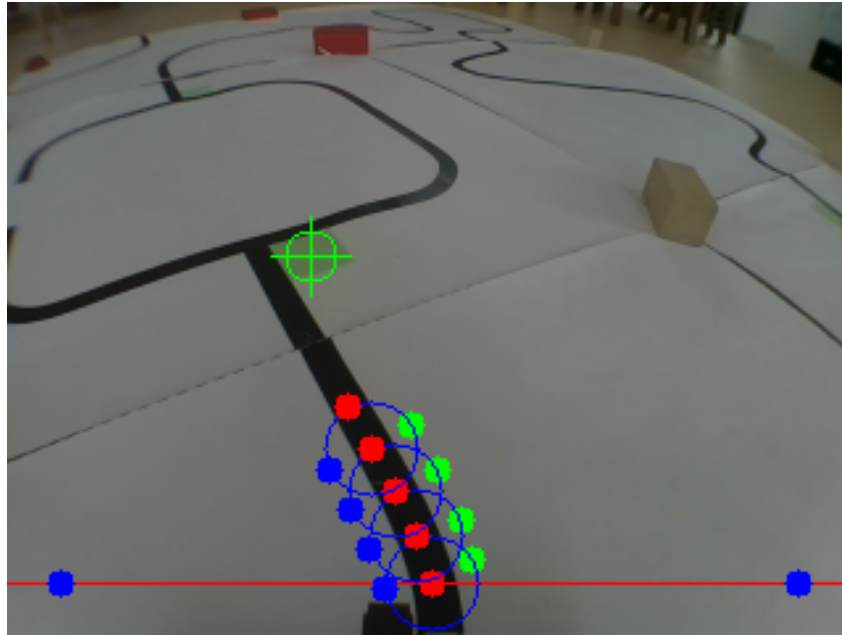


Figure 3: OpenCV example

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](#)

```
1 #include "Servo.h"
2
3 Servo mot[6][3];
4 int offset[6][3];
5 int md; // delay between 'movements'
6
7 // US definition
8 const int trigPin = 15;
9 const int echoPin = 14;
10
11 void setup() {
12     Serial.begin (9600);
13     //US
14     pinMode(trigPin, OUTPUT); // trig is an output
15     pinMode(echoPin, INPUT);  // echo is an input
16
17     md = 200; // delay between 'movements'
18
19     mot[0][0].attach(46);
20     mot[0][1].attach(4);
21     mot[0][2].attach(26);
22
23     offset[0][0] = 0;
24     offset[0][1] = 1;
25     offset[0][2] = 2;
26
27
28     mot[1][0].attach(48);
29     mot[1][1].attach(5);
30     mot[1][2].attach(32);
31
32     offset[1][0] = 0;
33     offset[1][1] = 14;
34     offset[1][2] = 7;
35
36     mot[2][0].attach(50);
37     mot[2][1].attach(6);
38     mot[2][2].attach(34);
39
40     offset[2][0] = 0;
41     offset[2][1] = 0;
42     offset[2][2] = 9;
43
44     mot[3][0].attach(52);
45     mot[3][1].attach(7);
46     mot[3][2].attach(36);
47
48     offset[3][0] = 0;
49     offset[3][1] = 12;
50     offset[3][2] = 3;
51
```

```

52  mot[4][0].attach(42);
53  mot[4][1].attach(2);
54  mot[4][2].attach(22);
55
56  offset[4][0] = 0;
57  offset[4][1] = 10;
58  offset[4][2] = 7;
59
60  mot[5][0].attach(44);
61  mot[5][1].attach(3);
62  mot[5][2].attach(24);
63
64  offset[5][0] = 0;
65  offset[5][1] = 1;
66  offset[5][2] = 0;
67 }
68
69 void loop() {
70   // check distance
71   if (getUS() > 35.0f){
72       // move 'forward' if object not in front
73       // moveFwd() is very quick to execute once so doing
74       // it w/o interrupts or using millis() works properly
75       // few centimeters per moveFwd() call
76       moveFwd();
77       delay(100);
78   }
79   else{
80       // default steady position
81       resetPos();
82       delay(100);
83       // rotate until not facin obstacle
84       // few degrees per Rotate() call
85       rotate();
86   }
87 }
88
89 float getUS() {
90   long duration;
91   float distance;
92   digitalWrite(trigPin, LOW);
93   delayMicroseconds(2);
94   digitalWrite(trigPin, HIGH);
95   delayMicroseconds(10); // trig 10ms HIGH
96   digitalWrite(trigPin, LOW);
97
98   // echo calculation
99   duration = pulseIn(echoPin, HIGH);
100  // distance proportional to the output duration
101  distance = duration*340/(2*10000); // sound speed
102
103  // out of range
104  if ( distance <= 0){
105      Serial.println("Out of range");
106  }
107  else {
108      Serial.print(distance);
109      Serial.print(" cm ");

```

```

110     Serial.print(duration);
111     Serial.println(" ms");
112     return(distance);
113 }
114 }
115
116 // default steady position
117 void resetPos(){
118     for (int i=0; i<6; i+=2)
119     {
120         mot[i][0].write(90);
121         mot[i][1].write(offset[i][1]+34);
122         mot[i][2].write(offset[i][2]+40);
123     }
124     delay(md);
125 }
126 }
127
128 // not used - kept for future additions
129 void nicePose(){
130     for (int i=1; i<6; i+=2)
131     {
132         mot[i][0].write(90);
133         mot[i][1].write(offset[i][1]+100);
134         mot[i][2].write(offset[i][2]+110);
135     }
136 }
137
138 // not used - kept for future additions
139 void fwdPose(){
140     for (int i=1; i<6; i+=2)
141     {
142         mot[i][0].write(90);
143         mot[i][1].write(offset[i][1]+34);
144         mot[i][2].write(offset[i][2]+40);
145     }
146 }
147
148 // rotates on itself
149 // few degrees per Rotate() call
150 void rotate(){
151     Serial.write("Rotate"); // avoids obstacle, rotates on itself
152
153     mot[0][2].write(45);
154     mot[2][2].write(45);
155     mot[4][2].write(45); // leg low
156     mot[0][1].write(150);
157     mot[2][1].write(150); // middle leg
158     mot[4][1].write(150); // 3 first legs up and done
159
160     delay(500);
161     mot[0][0].write(0);
162     mot[2][0].write(0);
163     mot[4][0].write(0); // core rotation and done
164
165     delay(500);
166
167     mot[0][2].write(135);

```

```

168     mot[2][2].write(135);
169     mot[4][2].write(135);
170     mot[0][1].write(30);
171     mot[2][1].write(30);
172     mot[4][1].write(30); // lower 3 first legs
173
174     delay(500);
175     mot[1][2].write(45);
176     mot[4][2].write(45);
177     mot[5][2].write(45);
178     mot[1][1].write(150);
179     mot[3][1].write(150);
180     mot[5][1].write(150); // raise 3 next legs
181
182     delay(500);
183     mot[1][0].write(0);
184     mot[3][0].write(0);
185     mot[4][0].write(0); // 2nd rotation
186
187     delay(500);
188     mot[1][2].write(135);
189     mot[3][2].write(135);
190     mot[5][2].write(135);
191     mot[1][1].write(30);
192     mot[3][1].write(30);
193     mot[5][1].write(30);
194     delay(500); // lower legs
195
196     mot[0][0].write(90);
197     mot[1][0].write(90);
198     mot[2][0].write(90);
199     mot[3][0].write(90);
200     mot[4][0].write(90);
201     mot[5][0].write(90); // final rotation and end
202 }
203
204 // move straight forward
205 void moveFwd(){
206     for (int i=0; i<6; i+=2)
207     {
208         mot[i][0].write(90);
209         mot[i][1].write(offset[i][1]+34);
210         mot[i][2].write(offset[i][2]+40);
211     }
212     delay(md);
213
214     for (int i=1; i<6; i+=2)
215     {
216         mot[i][0].write(90);
217         mot[i][1].write(offset[i][1]+100);
218         mot[i][2].write(offset[i][2]+110);
219     }
220     delay(md);
221     dir1();
222     delay(md);
223
224     for (int i=1; i<6; i+=2)
225     {

```

```

226         mot[i][0].write(90);
227         mot[i][1].write(offset[i][1]+34);
228         mot[i][2].write(offset[i][2]+40);
229     }
230     delay(md);
231     for (int i=1; i<6; i+=2)
232     {
233         mot[i][0].write(90);
234         mot[i][1].write(offset[i][1]+100);
235         mot[i][2].write(offset[i][2]+110);
236     }
237     delay(md);
238     dir2();
239     delay(md);
240     for (int i=1; i<6; i+=2)
241     {
242         mot[i][0].write(90);
243         mot[i][1].write(offset[i][1]+34);
244         mot[i][2].write(offset[i][2]+40);
245     }
246     delay(md);
247
248 }
249
250 // used in 'moveFwd' as pose - legs slight rotation
251 void dir1(){
252     // full
253     for (int i=0; i<6; i+=1)
254     {
255         mot[i][0].write(60);
256     }
257     // 'special'
258     for (int i=0; i<6; i+=2)
259     {
260         mot[i][0].write(120);
261     }
262 }
263
264 // used in 'moveFwd' as pose
265 // legs slight rotation in opposite direction relative to dir1()
266 void dir2(){
267     // full opposite
268     for (int i=0; i<6; i+=1)
269     {
270         mot[i][0].write(120);
271     }
272     // 'special' opposite
273     for (int i=0; i<6; i+=2)
274     {
275         mot[i][0].write(60);
276     }
277 }

```
