UNIT 11: RC-SERVOMOTOR CONTROL

AIMS

The Arduino controller, like the majority of modern controllers, is capable of generating width modulated digital signals. They're called "PWM" ("Pulse Width Modulation") or "PDM" (Pulse-duration Modulation) signals.

This unit looks at what they are, how they're created and what they're used for. I can tell you now that they're closely connected to voltage regulation and control.

Although it may not have much to do with the topic of this unit, we're also going to look at some other functions of the Arduino programming language. You might be able to use them in some of the exercises on power regulation.

THEORY SECTION

- Servomotors
- The "servo" library and its functions
  - The #include function
  - The attach() function
  - The writeMicroseconds() function
  - The write() function
  - The read() function
  - The attached() function
  - The detach() function

PRACTICE SECTION

- EXAMPLE 1: The servo
- EXAMPLE 2: The servo V2
- EXAMPLE 3: The servo V3
- EXAMPLE 4: Controlling a rudder

PRACTICE MATERIALS

-Lap top or desk top computer
-Arduino IDE work environment; this should include the supplementary material already installed and configured.
-Arduino UNO controller board
-A USB cable

TABLE OF CONTENTS

THEORY SECTION 3

1. SERVOMOTORES 3
2. THE "SERVO" LIBRARY AND ITS FUNCTIONS 5

PRACTICE SECTION 10

3. EXAMPLE 1: THE SERVO 10
4. EXAMPLE 2: THE SERVO V2 12
5. EXAMPLE 3: THE SERVO V3 12
6. EXAMPLE 4: CONTROLLING A RUDDER 13

REFERENCES 14

THEORY SECTION

1. SERVOMOTORES

Have you heard of them? Maybe not but you've almost certainly seen them in action. Think of the following remote controlled toys (which sometimes seem like more than just toys): a car, a boat and a plane.

1. What controls the steerage of the car wheels?
2. What makes the rudder change the direction of the boat?
3. And the flaps and the rudder on the plane?

The answer is a "servomotor" or just plain "servo". These mechanisms are similar to a conventional motor but they are able to make turns or controlled movements in any direction and in any position within their operational range. For instance, we can direct it to make a 34º clockwise turn. It'll stay in this position until we direct it to make another movement. As the figure on the left shows, there are a number of accessories that make it possible to connect the shaft of the motor to the part it moves: the car steerage, the rudder, the flaps etc…

But don't think that servos are just used for toys. They're also used in robotics and manufacturing: to move and turn a robot's arm, to open and close a valve, to move an implement or tool, to position an object and many other applications.

Regardless of a servo's size, shape or usage, its internal composition is usually much the same and comprises three basic elements: a motor, a gear train and an electronic control board. Have a look at the figure on the left to get an idea of what the servo you're going to use looks like on the inside.

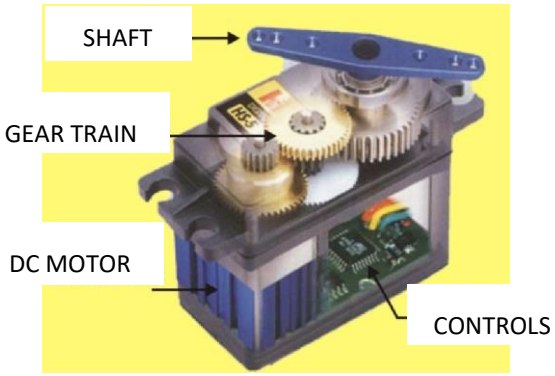

Figure 1: RC-SERVOMOTOR

We need only three leads to connect up our servo. We connect the black lead to the GND port or the 0 V power port and the red one to the +5 V power port. The PWM control signal goes through the white lead. Our Arduino will generate this signal of course.

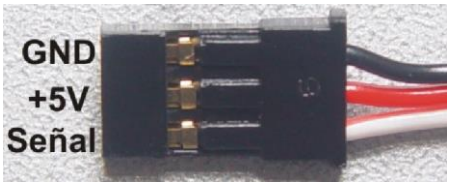

Figure 2: RC-SERVOMOTOR CONECTIONS

You'll find lots of manufacturers, types and models of servos. There are different sizes, strengths or torques, speeds, supply voltages, types of shafts and travels. There are servos with shafts that can rotate freely and others with shafts that can only rotate a certain number of degrees. The one you're going to use in these exercises can rotate 180º.

I've already said that the servo is controlled by sending a PWM signal through the appropriate pin. The duty cycle of the signal determines the position the shaft moves to or its rotation. It's essential to follow the guidelines supplied by each particular model's manufacturer but here's an example in the figure below that may be of use.

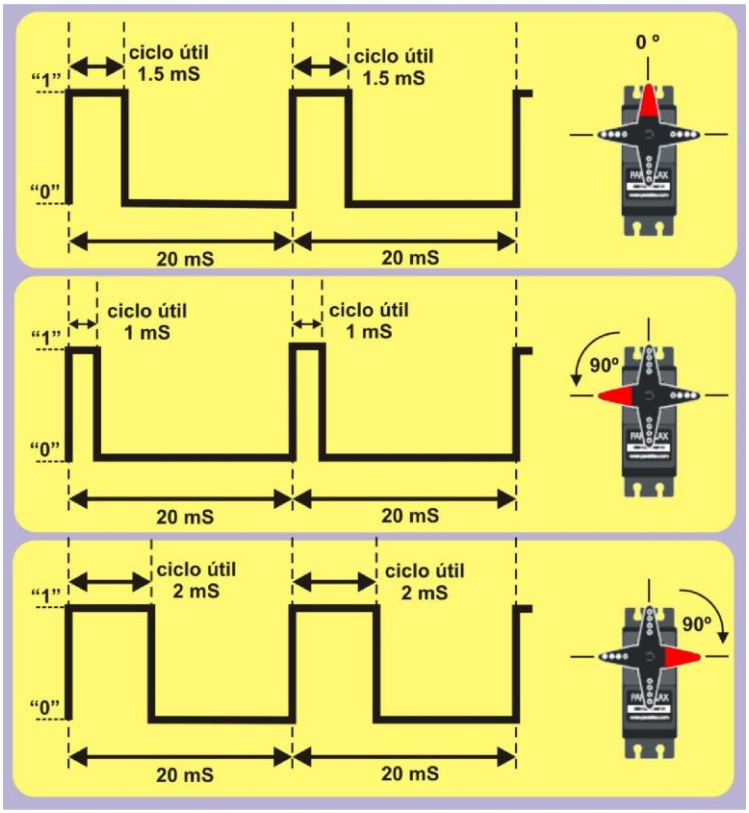

Figure 3: DUTY CYCLE

The period, T, of the signal is fixed and is usually 20 mS (F=50Hz). A duty cycle of 1.5 mS keeps the servo on 0º. A duty cycle of 1 mS causes the servo to turn 90º in one direction, all the way to the left, for instance. Between 1 mS and the 1.5 mS in the middle, the servo turns proportionately.

A 2 mS wide duty cycle enables the servo to turn another 90º to the right, this time in the opposite direction. Between the 1.5 mS in the middle and 2 mS, the servo turns proportionately.

Of course, a duty cycle of between 1 mS (90º to the left) and 2 mS (90º to the right) makes a turn of 180º possible.

2. THE "SERVO" LIBRARY AND ITS FUNCTIONS

When Arduino adds functions of general use with no specific application, they're usually incorporated directly into the language. Nevertheless, when new functions that have a specific purpose are created they're usually added to what are called "libraries". A library is nothing more than an auxiliary file, with an "h" extension that contains the code needed to implement a collection of new functions.

You'll find libraries with functions that make it possible to control screens, robots, servos, communications, memory cards, connections to Internet, motors and much more. Some manufacturers of peripherals, cards and accessories provide their own libraries so you can use their products conveniently and quickly. You may end up with an extensive collection of libraries with functions that help you to design all kinds of programs and applications.

Right now we're going to create a library called "Servo". It's a file called "Servo.h" provided by Arduino; it installed itself automatically with the IDE or Integrated Development Environment. In other words, it's already in your computer.

If you include this file in your programs, you've enriched the Arduino programming language by integrating new functions in it. You've now got six functions that were designed exclusively for the control of servos.

- **The #include function ()**

From now on the first thing you have to do with your programs is include the library or libraries with the functions you want to use. You do this by using the #include directive.

Syntax:

#include <file.h>

<file.h>: this is the name of the file that contains the library.

Example:

#include <servo.h> Includes the "servo" library in your program. This enables you to use all the functions contained in this library we'll be studying them very soon.

- **The attach() function:**

This function attaches the Servo variable to any of the available digital pins. Unlike the PWM signals, which are only available on six of the controller's pins, the control signals for servos can be generated on any pin at all.

Syntax:

servo.attach(pin, min, max);

servo: This is a variable that assigns a number to our servo. We'll be looking at it very soon. pin: This is the number of the pin you're going to connect the servo to.

min: This is optional; it enables you to establish the minimum value of the signal's duty cycle for a 0º turn in microseconds; it's 544 by default. Have a look at the datasheet provided by the manufacturer of the servo.

max: This is optional; it enables you to establish the maximum value of the signal's duty cycle for a 180º turn in microseconds; it's 2400 by default. Have a look at the datasheet provided by the manufacturer.

***Example**:*

```
#include <servo.h>                    //Includes the "servo" library.
Servo S3003;                          //Creates a variable called "S3003"
…
S3003.attach(3);                      //Connects the servo S3003 to pin 3
```

- **The writeMicroseconds() function**

This function generates a PWM signal with a duty cycle in microseconds (uS) that will set the angle of the servo's shaft.

On standard servos a parameter value of 1000 is fully counter-clockwise, 2000 is fully clockwise, and 1500 is in the middle.

Syntax:

servo.writeMicroseconds(value);

servo: the name of the servo

value: the integer (int) that establishes the duration of the PWM signal's duty cycle.

Example:

```
#include <servo.h>                    //Includes the library
Servo S3003;                          //The servo's name (S3003)
S3003.attach(3);                      //Connects the servo S3003 to pin 3
S3003.writeMicroseconds(300);         //0.3 mS duty cycle (0º turn)
delay(1000);                          //Pauses 1 second
S3003.writeMicroseconds(2300);        //2.3 mS duty cycle (180º turn)
```

- **The write() function**

This function writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. On a continuous rotation servo, this will set the speed of the servo (with 0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement).

Syntax:

servo.write(value);

servo: the servo's name.

value: a byte type value that establishes the turn degree (between 0º and 180º).

Example:

```
#include <servo.h>                    //Includes the library
Servo S3003;                          //The servo's name (S3003)
S3003.attach(3,300,2300);             //Connects the servo and sets the duty cycle
S3003.write(0);                       //Full turn in one direction (0º)
delay(1000);                          //Pauses 1 second
S3003.write(180);                     //Full turn in the opposite direction (180º)
```

- **The read() function:**

This function reads the current angle of the servo's shaft which is actually the value passed to the last call to write().

Syntax:

servo.read();

servo: the servo's name

Example:

```
#include <servo.h>          //Includes the byte A library;
Servo S3003;                //Servo's name (S3003)
S3003.attach(3,300,2300);   //Connects the servo and establishes the duty cycle
…
…
A= S3003.read();            //Reads the current position of the servo's shaft
```

- **The attached() function:**

This function checks whether or not the Servo variable is attached to a pin. It returns a "true" o "false".

Syntax:

servo.attached();

servo: The servo's name.

Example:

```
#include <servo.h>          //Includes the library
Servo S3003;                //The servo's name (S3003)
S3003.attach(3,300,2300);   //Connects the servo and establishes the duty cycle
…
if(S3003.attached())        //True if it's connected
```

7.6.7 The detach() function:

This function detaches the Servo variable from its pin. If a Servo is attached, pins 9 and 10 CANNOT be used for PWM output. If all Servo variables are detached, then pins 9 and 10 CAN be used for PWM output with analogWrite().

Syntax:

servo.detach();

servo: The servo's name

Example:

```
#include <servo.h>          //Includes the library

Servo S3003;                //The servo's name (S3003)

S3003.attach(3);            //Connects and attaches the servo to pin 3

…

…

S3003.detach()              //Detaches the servo from pin 3
```

You're probably thoroughly bored with so much theory by now and, believe me, I'm truly sorry. Unfortunately I had no choice but to introduce all the functions contained in the "servo.h" library.

But don't worry: the reward for all your patience is coming up in the practice area. At last you'll see a bit of action. It's all very well flashing LED lights on and off and making noises but now it's time for a bit of movement.

PRACTICE SECTION

3. EXAMPLE 1: THE SERVO

This is a first look at the servo included in the material kit.

Here's the program (Figure 4). We've highlighted the three steps you need to follow to work with a servo:

1. We include the "Servo.h" library that contains the control functions. We also create a "Servo" variable with the name we want to assign to our servo, the S3003.
2. We confirm that the servo is connected to pin 3 using attach(3).
3. We specify the width of the duty cycle for a given turn using writeMicroseconds().

```
*/
#include <Servo.h>     //Incluir la libreria Servo
Servo S3003;           //Crea la variable S3003 (modelo de futaba)

int ciclo=2300;          //Establece el ciclo útil (entre 300 y 2300

/* S3003 se controla desde la linea D3. */
void setup()
{
  S3003.attach(3);   //Conecta el servo
}

void loop()
{
   S3003.writeMicroseconds(ciclo);     //El servo se posiciona segú
   while(1);                           //Fin del programa
}
```

Figure 4

The program generates a PWM signal with duty cycle of 2300 µS (a variable "cycle"). According to the specifications of the servo model the maximum turn of servo's shaft is 180º in either direction.

- **And now it's your turn**

First of all, work out how many microseconds it takes for the shaft to turn 1º. The formula for this for the servo is the following:

$$\mu S/º = \frac{Max - Min}{180º} = \frac{2300 - 300}{180} = 11.11\ \mu S$$

Now it's easy to work out the length of the PWM signal's duty cycle so that the shaft turns a certain number of degrees (g):

$$PWM = 2300 - (g * 11.11)$$

For example:

- ✓ For a turn of 0º (one end of the shaft), PWM = 2300 – (0 * 11.11) = 2300
- ✓ For a turn of 90º (in the middle of the shaft), PWM = 2300 – (90 * 11.11) = 1300
- ✓ For a turn of 180º (the other end of the shaft), PWM = 2300 – (180 * 11.11) = 300

You're going to change the contents of the "cycle" variable to vary the length of the duty cycle and ensure the shaft turns the number of degrees you want it to turn. Make the appropriate calculations, complete the table below and then check the results of each turn by recording them in the controller.

Turn in º	0º	22º	45º	70º	90º	110º	135º	155º	180º
Duty cycle in µS									

4. EXAMPLE 2: THE SERVO V2

This is another version of the previous exercise. In this one you're going to use the write() function instead of writeMicroseconds().

This way you state the number of degrees you want the shaft to turn. You can forget all about the calculations you made in the last exercise to work out the length of the duty cycle based on the turn you want; the write() function does it all for you. In this exercise the shaft turns the number of degrees you indicate in the "angle" variable.

Here's a detail you should bear in mind. Have a look at the attach() function in setup(). As well as indicating the pin you've connected the servo to you should also indicate the minimum and maximum length of the duty cycle specified by the manufacturer.

```
void setup()
{
  S3003.attach(3,300,2300);
}

void loop()
{
   S3003.write(angulo);
   while(1);
}
```

Figure 5

5. EXAMPLE 3: THE SERVO V3

This is a consequence of the previous exercises. The servo moves from one end to the other (180º in all) at intervals of 5º. Then it returns to the initial position and the cycle is repeated.

We use the for() function so that the "Angle" variable goes from 0 to 180 in multiples of five. The current value of this variable is applied to the servo which makes the required turn every 0.1". After a pause of 2" the servo returns to the initial position of 0º and the cycle is repeated.

```
void loop()
{
  for(int Angulo=0; Angulo<=180;Angulo+=5)
    {
      S3003.write(Angulo);  //El servo gira los grados deseados
      delay(100);           //Intervalo entre un giro y el siguient
    }
  delay(2000);      //Se mantiene en esa mosición durante 2"
  S3003.write(0);   //El servo vuelve a la posición inicial (0°)
  delay(2000);      //Espera 2" antes de volver a repetir el ciclo
}
```

Figure 6

6. EXAMPLE 4: CONTROLLING A RUDDER

Here's a nice practical exercise to finish off the unit. Imagine you're controlling the movement of a rudder on a boat. When you put the rudder in the middle, the servo's shaft is at 90º and the boat goes in a straight line. The pushbuttons connected to pins 4 and 7 enable you to turn the rudder to the left or to the right at intervals of 5º; you can't turn more than 50º in either direction. In other words the rudder's total travel would be from 40º to 140º starting from the middle position of 90º.

Okay, now let's go sailing!

REFERENCES

BOOKS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

WEB SITES

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com