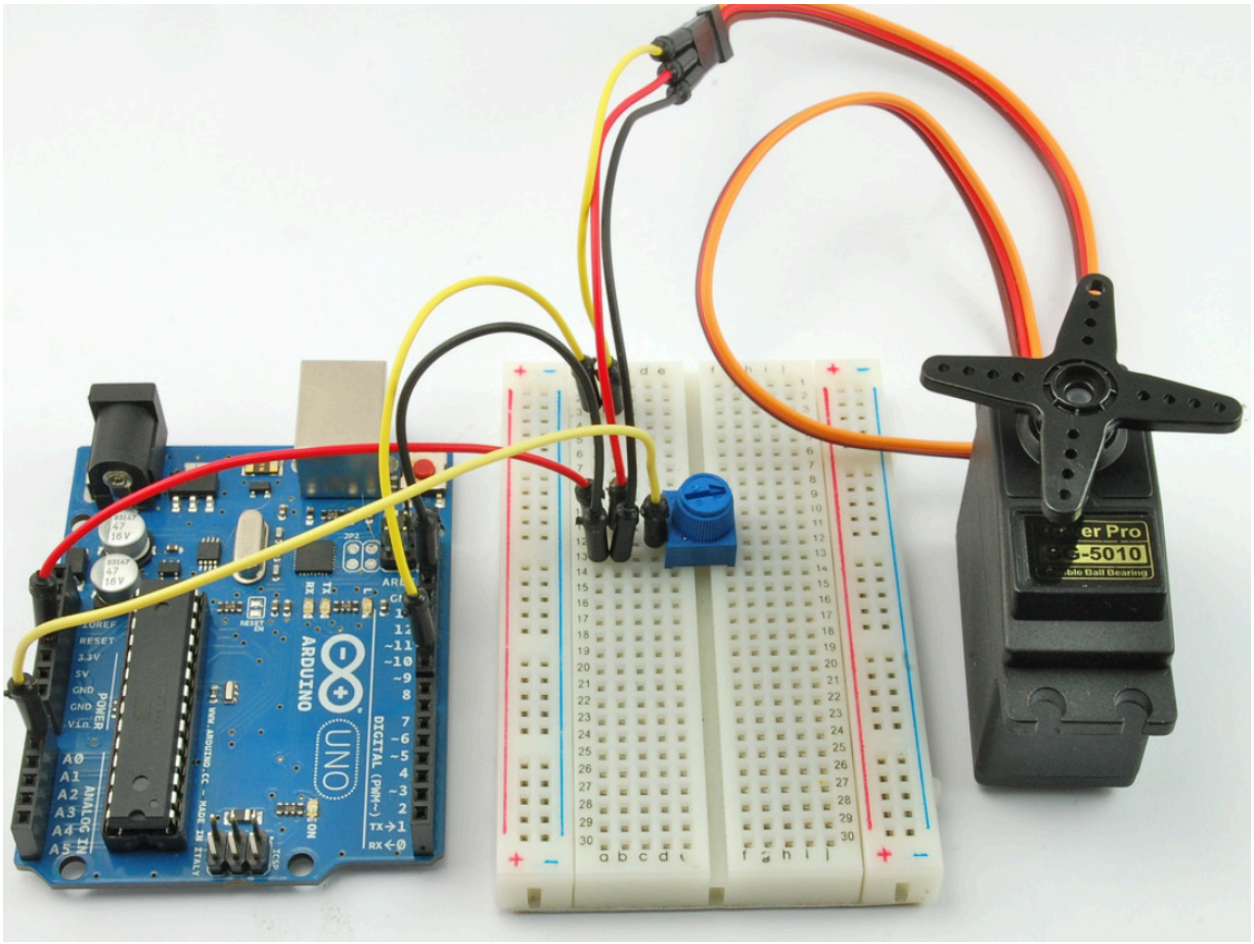
Servo Motors

Overview

In this lesson, you will learn how to control a servo motor using an Arduino.

Firstly, you will get the servo to sweep back and forth automatically and then you will add a pot to control the position of the servo.



Parts

To build the projects described in this lesson, you will need the following parts.

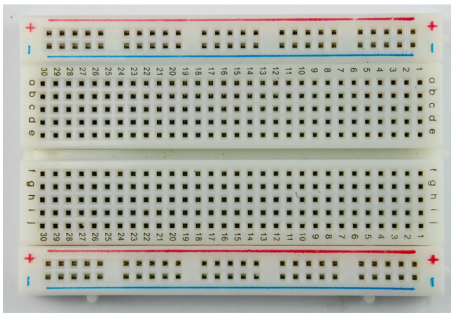
Part

Qty

Servo Motor 1

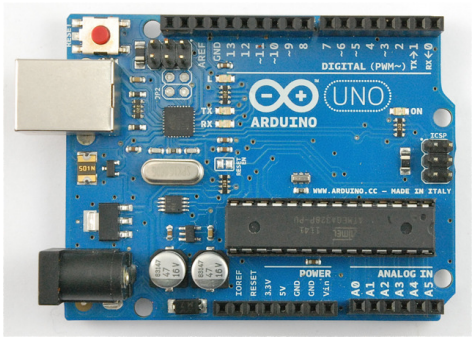
10 kΩ variable resistor (pot)

1



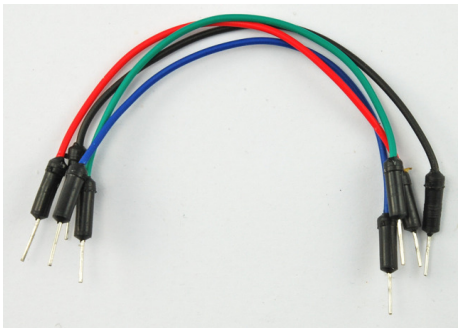
Half-size Breadboard

1



Arduino Uno R3

1



Jumper wire pack

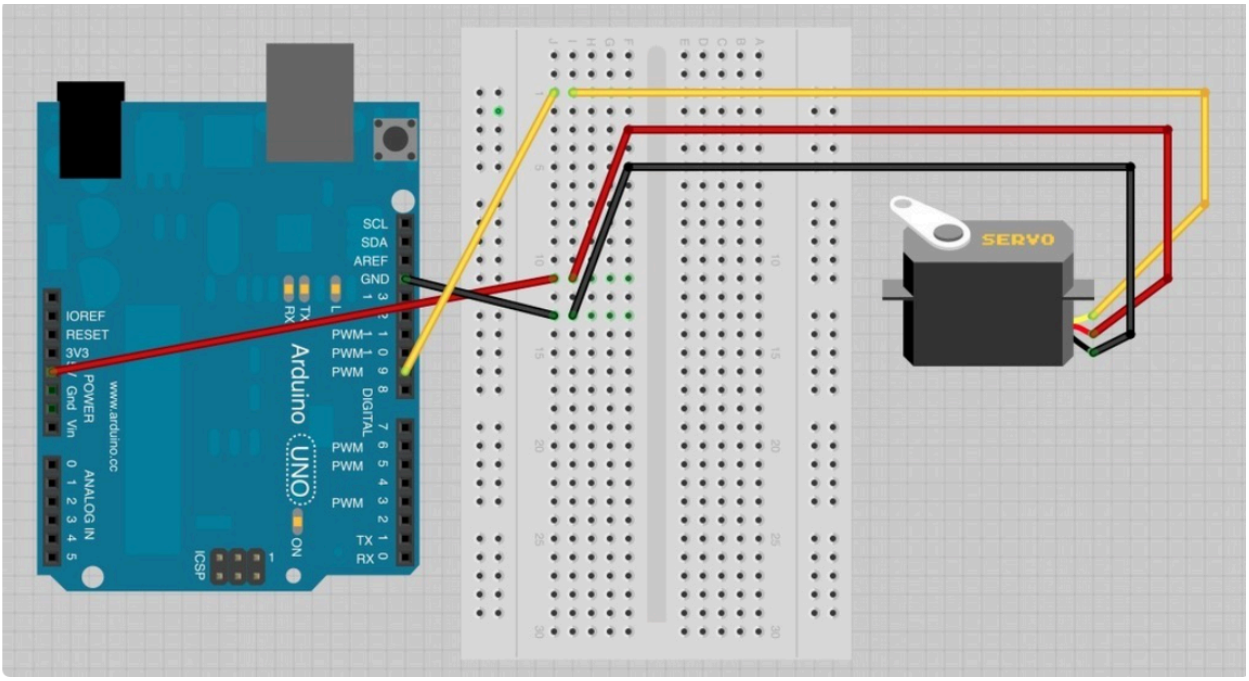
1

100 µF capacitor

Optional

The Breadboard Layout for 'Sweep'

For this experiment, the only thing connected to the Arduino is the servo motor.



The servo motor has three leads. The color of the leads varies between servo motors, but the red lead is always 5V and GND will either be black or brown. The other lead is the control lead and this is usually orange or yellow. This control lead is connected to digital pin 9.

The servo is conveniently terminated in a socket into which we can push jumper wires, to link it to the breadboard and then to the Arduino.

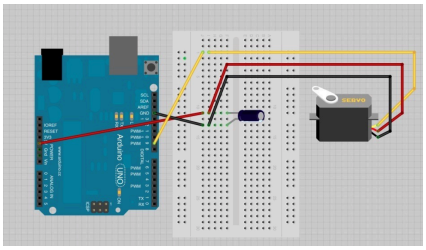
If the Servo Misbehaves

Your servo may behave erratically, and you may find that this only happens when the Arduino is plugged into certain USB ports. This is because the servo draws quite a lot of power, especially as the motor is starting up, and this sudden high demand can be enough to drop the voltage on the Arduino board, so that it resets itself.

If this happens, then you can usually cure it by adding a high value capacitor (470uF or greater) between GND and 5V on the breadboard.

The capacitor acts as a reservoir of electricity for the motor to use, so that when it starts, it takes charge from the capacitor as well as the Arduino supply.

The longer lead of the capacitor is the positive lead and this should be connected to 5V. The negative lead is also often marked with a '-' symbol.



Arduino Code for 'Sweep'

Load up the following sketch onto your Arduino. You should find that the servo immediately begins to turn first in one direction and then back in the other.

The sketch is based on the standard 'sweep' sketch that you can find in the Arduino Examples under the folder 'servo'. You can if you prefer just run that sketch.

```
1.
2.
3. #include <Servo.h>
4.
5. int servoPin = 9;
6.
7. Servo servo;
8.
9. int angle = 0;   // servo position in degrees
10.
11.void setup()
12.{
13.  servo.attach(servoPin);
14.}
15.
16.
17.void loop()
18.{
19.  // scan from 0 to 180 degrees
20.  for(angle = 0; angle < 180; angle++)
21.  {
22.    servo.write(angle);
23.    delay(15);
24.  }
25.  // now scan back from 180 to 0 degrees
26.  for(angle = 180; angle > 0; angle--)
27.  {
```

```
28.     servo.write(angle);
29.     delay(15);
30.   }
31. }
```

Servo motors are controlled by a series of pulses and to make it easy to use them, an Arduino library has been created so that you can just instruct the servo to turn to a particular angle.

The commands for using a servo are like built-in Arduino commands, but because you are not always going to be using a servo in your projects, they are kept in something called a library. If you are going to use commands in the servo library, you need to tell the Arduino IDE that you are using the library with this command:

```
1. #include <Servo.h>
```

As usual, we then use a variable 'servoPin' to define the pin that is to control the servo.

This line:

```
1. Servo servo;
```

defines a new variable 'servo' of type 'Servo'. The library has provided us with a new type, like 'int' or 'float' that represents a servo. You can actually define up to eight servos in this way, so if we had two servos, then we could write something like this:

```
1. Servo servo1;
2. Servo servo2;
```

In the 'setup' function we need to link the 'servo' variable to the pin that will control the servo using this command:

```
1.   servo.attach(servoPin);
```

The variable 'angle' is used to contain the current angle of the servo in degrees. In the 'loop' function, we use two 'for' loops to first increase the angle in one direction and then back in the other when it gets to 180 degrees.

The command:

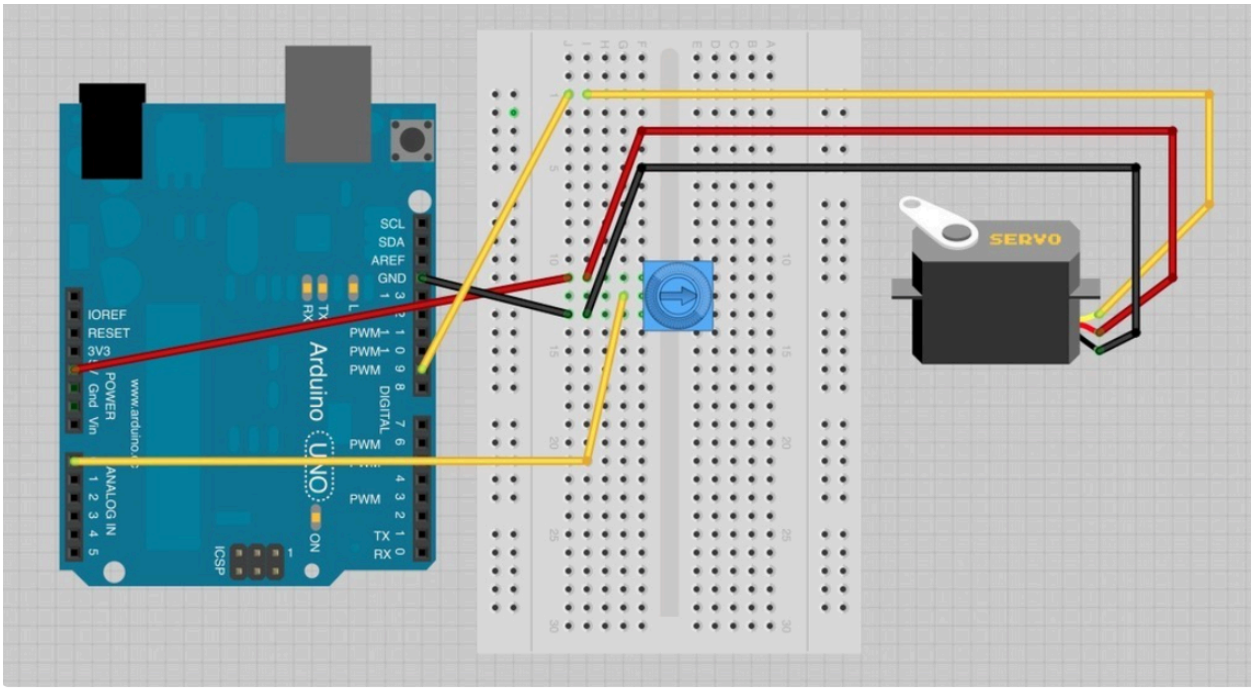
```
1.     servo.write(angle);
```

tells the servo to update its position to the angle supplied as a parameter.

The Breadboard Layout for 'Knob'

Our next step is to add a pot so that we can control the position of the servo by turning the knob.

You just need to add the pot and a lead from its slider to A0 on the Arduino.



Arduino Code for 'Knob'

The code to make the servo follow the knob's position is simpler than to make it sweep.

```
2.
3.
4. #include <Servo.h>
```

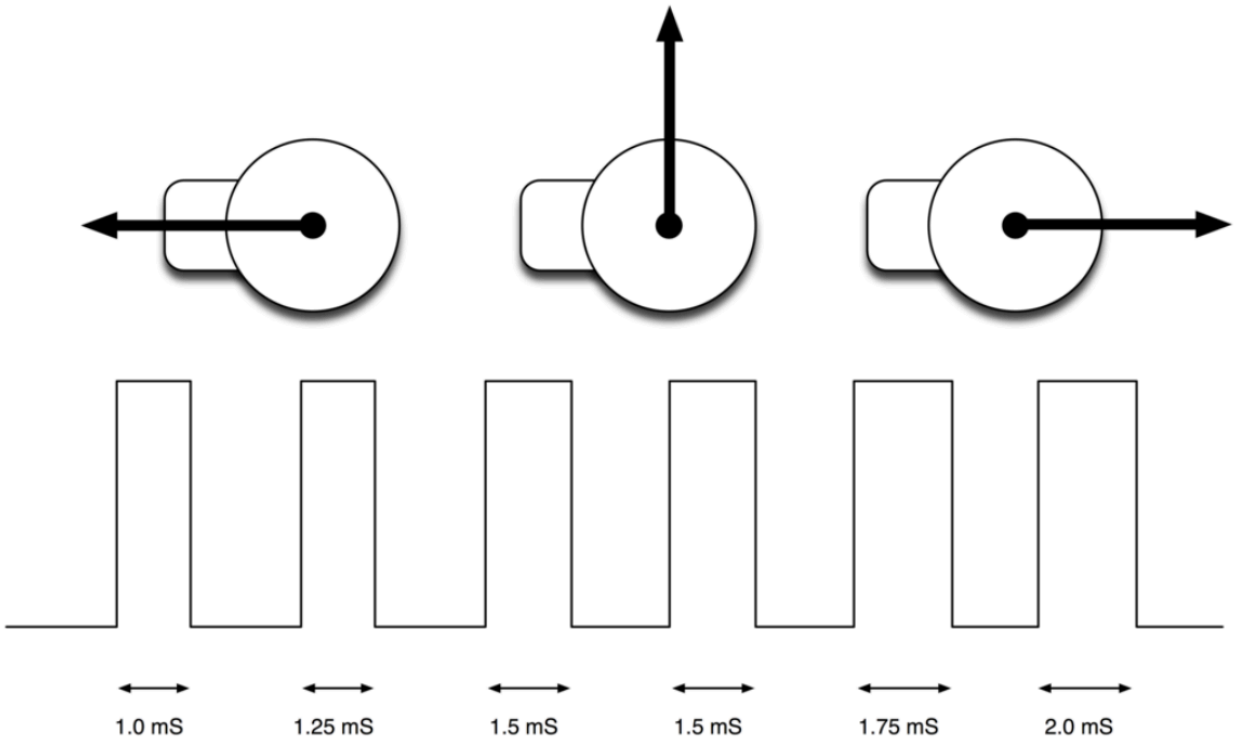
```
5.
6.  int potPin = 0;
7.  int servoPin = 9;
8.  Servo servo;
9.
10. void setup()
11. {
12.   servo.attach(servoPin);
13. }
14.
15. void loop()
16. {
17.   int reading = analogRead(potPin);     // 0 to 1023
18.   int angle = reading / 6;              // 0 to 180-ish
19.   servo.write(angle);
20. }
```

There is now a second variable called 'potPin'.

To set the position of the servo, we take an analog reading from A0. This gives us a value of between 0 and 1023. Since the servo can only rotate through 180 degrees, we need to scale this down. Dividing it by six will give us an angle between 0 and 170, which will do just fine.

Servo Motors

The position of the servo motor is set by the length of a pulse. The servo expects to receive a pulse roughly every 20 milliseconds. If that pulse is high for 1 millisecond, then the servo angle will be zero, if it is 1.5 milliseconds, then it will be at its centre position and if it is 2 milliseconds it will be at 180 degrees.



The end points of the servo can vary and many servos only turn through about 170 degrees. You can also buy 'continuous' servos that can rotate through the full 360 degrees.