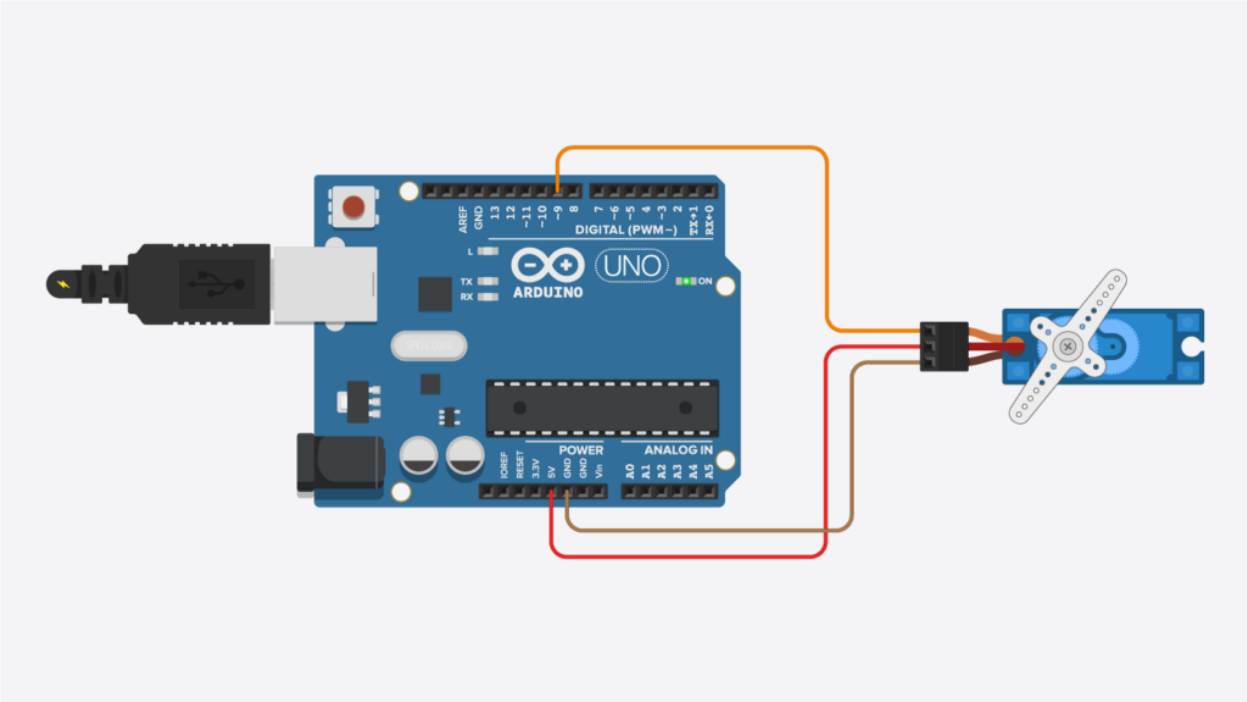
Arduino Server Control tutorial

Reference: https://www.makerguides.com/servo-arduino-tutorial/

How to control servo motors with Arduino



In this tutorial, you will learn how servo motors work and how to control them with Arduino. I have included wiring diagrams and several example codes! Servo motors are often used in robotics projects but you can also find them in RC cars, planes, etc. They are very useful when you need precise position control and/or high torque.

In the first part of this article, we will look at the inner workings of a servo and what type of control signal it uses. I also explain what the differences between a standard and a continuous servo are. Next, I will show you how to connect a servo motor to the Arduino.

With the first code example, you can control both the position as well as the speed of the servo motor. After that, we will look into controlling a servo with a potentiometer and how you can modify the code to control multiple servo motors at the same time. Lastly, at the end of this article, you can find the specifications and dimensions of some of the most popular servo motors on the market.

Supplies

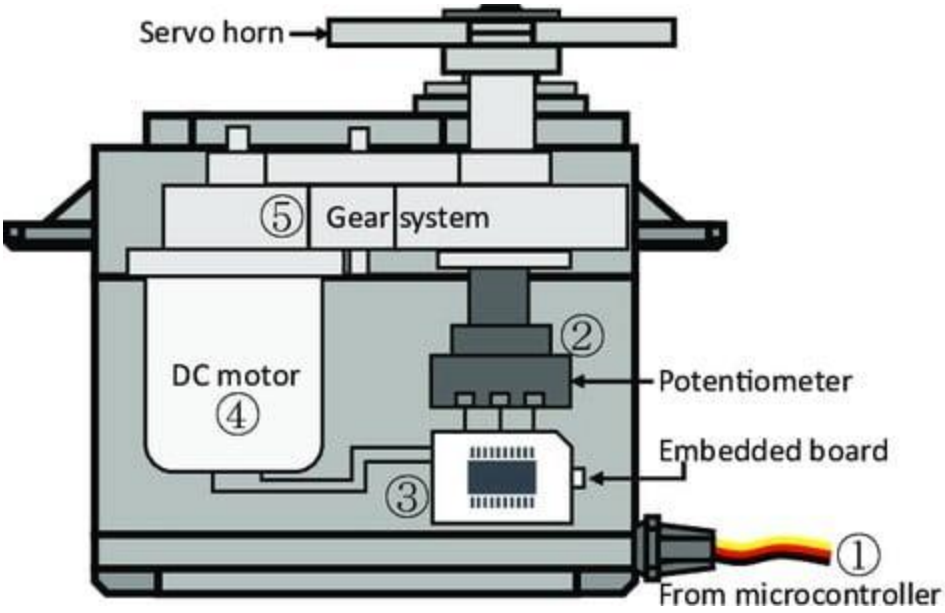
Hardware components

SG90 micro servo	× 1	Amazon
MG996R high-torque servo	× 1	Amazon
Arduino Uno Rev3	× 1	Amazon
Jumper wires	× 15	Amazon
Breadboard	× 1	Amazon
10 kΩ potentiometer (breadboard type)	× 1	Amazon
USB cable type A/B	× 1	Amazon
5V power supply (optional)		

How does a servo work?

A standard hobby servo typically consists of a small electric motor, a potentiometer, control electronics, and a gearbox. The position of the output shaft is constantly measured by the internal potentiometer and compared with the target position set by the controller (e.g. the Arduino).

According to the error, the control electronics adjust the actual position of the output shaft so that it matches the target position. This is known as a closed-loop control system.



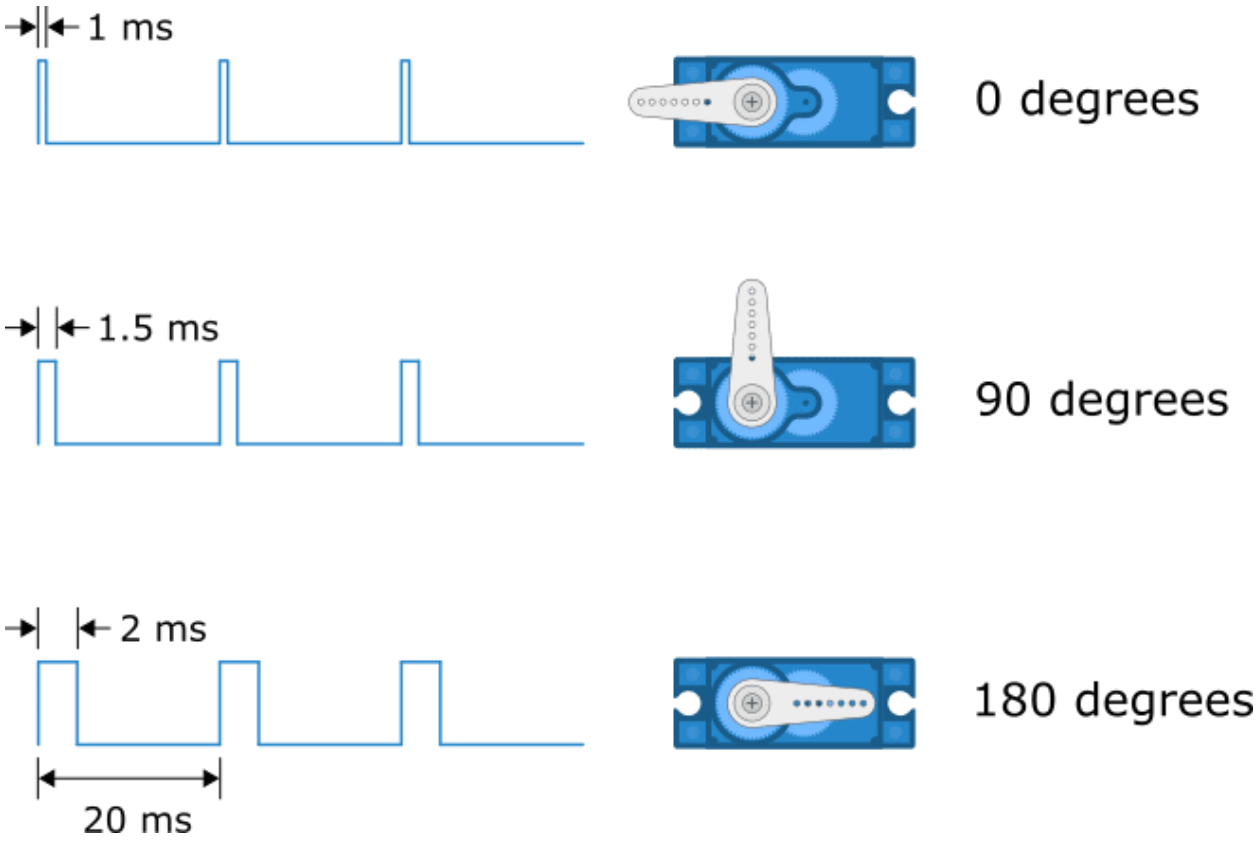
Schematic of an RC servo motor (Hwang et al. 2018)

The gearbox decreases the speed of the motor, which increases the torque at the output shaft. The maximum speed of the output shaft is usually around 60 RPM.

Servo Control

Servo motors are controlled by sending a PWM (pulse-width modulation) signal to the signal line of the servo. The width of the pulses determines the position of the output shaft. When you send the servo a signal with a pulse width of 1.5 milliseconds (ms), the servo will move to the neutral position (90 degrees). The min (0 degrees) and max (180 degrees) position typically correspond to a pulse width of 1 ms and 2 ms respectively. Note this can vary slightly between different types and brands of servo motors (e.g. 0.5 and 2.5 ms). Many servos only rotate through about 170 degrees (or even only 90) but the middle position is almost always at 1.5 ms.

For adjusting the min and max position in the code, see the section below.



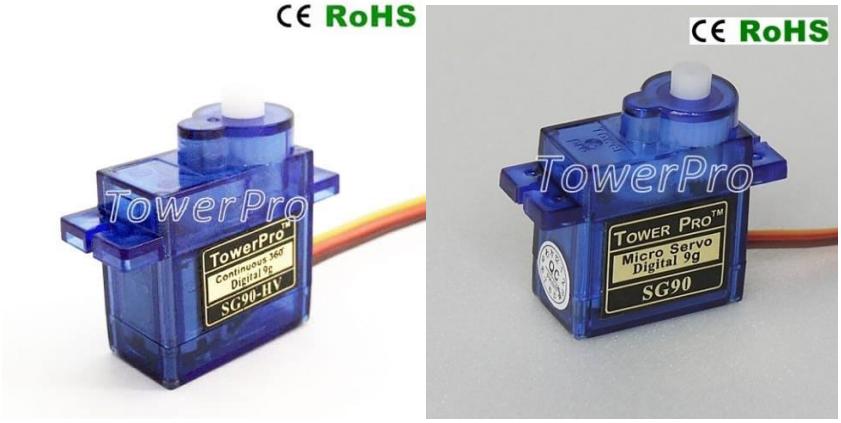
Servo motors generally expect a pulse every 20 milliseconds or 50 Hz but many RC servos work fine in a range of 40 to 200 Hz.

360-degree (continuous) vs 180-degree (standard) servo

Most RC servos are from the 180-degree variety, which means that they can only rotate in a range of 0 to 180 degrees. However, continuous rotation, also known as 360-degree servo motors, are also available.

Continuous rotation servos react differently to the control signal than standard 180-degree servos. **With a continuous rotation servo, you can not control the exact position of the output shaft, only the speed and the direction**. A 1 ms pulse will set the speed of the servo motor to full speed in one direction and a 2 ms pulse to full speed in the other. A value near 1.5 ms lets the motor stop.

If your servo behaves in an unexpected way, you might be using a continuous servo instead of a standard one.



TowerPro SG90-HV Continuous 360° Digital 9g servo (left) vs TowerPro SG90 Micro Servo Digital 9g (right). Notice that they look almost identical on the outside.

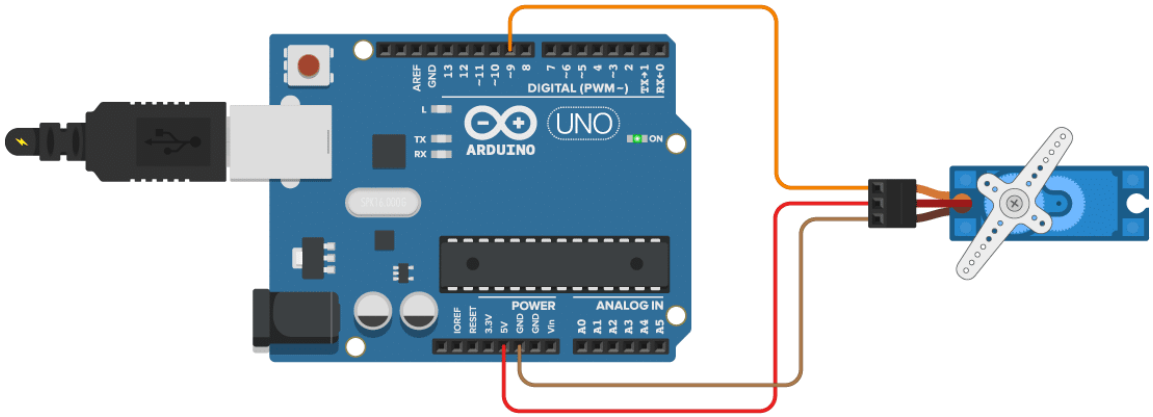
How to connect a servo motor to the Arduino?

Wiring a servo motor is very easy because you only need to connect three wires: power, ground, and signal. The power wire is typically red and needs to be connected to 5 V.

A micro servo like the SG90 consumes around 10 mA when it's idle and 100 – 250 mA when rotating, so you can power it directly with the 5 V output of the Arduino. However, you need to be careful when using multiple or larger servo motors. **If your motor(s) consume more than 300 mA you should use an external power supply to avoid damaging the Arduino!** See the schematic below for using external power supplies.

The ground wire is typically black or brown and should be connected to the ground pin of the Arduino. When using a separate power supply, connect the ground wire to both the Arduino and the power supply ground.

The signal wire is typically yellow, orange, or white can be connected to any of the digital pins of the Arduino. In this case, I connected it to digital pin 9.



Servo motor with Arduino Uno wiring diagram.

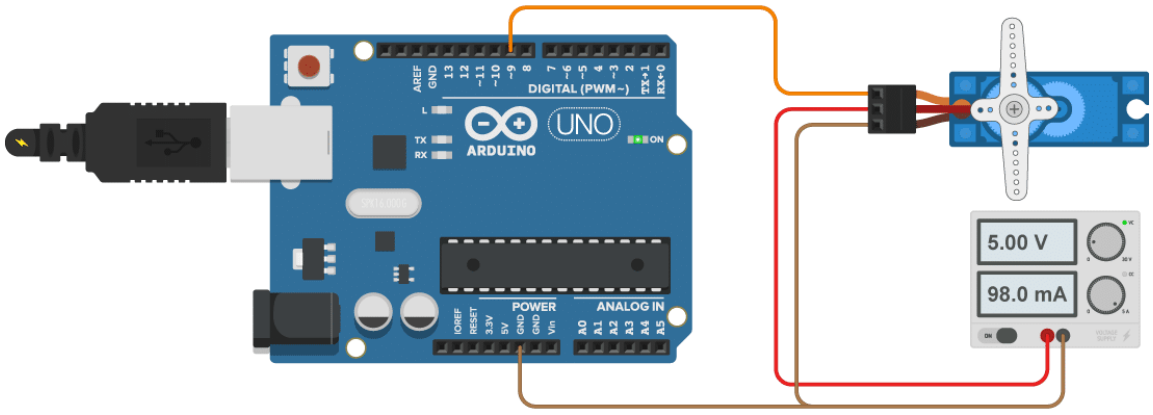
The connections are also given in the table below.

Servo Motor Connections

Servo motor	Arduino
Power (red)	5 V
Ground (black or brown)	GND
Signal (yellow, orange or white)	Pin 9

As I mentioned before, if you are using large or multiple servo motors you should use an external power supply. Simply connect the power supply as shown in the wiring diagram below. Make sure to connect the GND pin of the Arduino and the power supply together.

You can also use this setup if your servo motor requires a different voltage than the Arduino can provide e.g. 6 V or higher.



Servo motor powered with an external power supply.

Servo Motor with external power supply connections

Servo motor	Connection
Power (red)	5 V power supply
Ground (black or brown)	Power supply ground and Arduino GND
Signal (yellow, orange or white)	Pin 9 Arduino

Servo motor with Arduino example code

To control the servo motor we will be using the **Servo.h** library which comes pre-installed with the Arduino IDE. With the example code below, you can control the exact position of the servo motor and it also includes code to sweep the servo arm back and forth automatically.

You can upload the example code to your Arduino via the Arduino IDE. Next, I will explain how the code works.

You can copy the code by clicking on the button in the top right corner of the code field.

```
/* Servo motor with Arduino example code. Position and sweep. More info: https://www.makerguides.com/ */

// Include the servo library:
#include <Servo.h>

// Create a new servo object:
```

```
Servo myservo;

// Define the servo pin:
#define servoPin 9

// Create a variable to store the servo position:
int angle = 0;

void setup() {
  // Attach the Servo variable to a pin:
  myservo.attach(servoPin);
}

void loop() {
  // Tell the servo to go to a particular angle:
  myservo.write(90);
  delay(1000);
  myservo.write(180);
  delay(1000);
  myservo.write(0);
  delay(1000);

  // Sweep from 0 to 180 degrees:
  for (angle = 0; angle <= 180; angle += 1) {
    myservo.write(angle);
    delay(15);
  }

  // And back from 180 to 0 degrees:
  for (angle = 180; angle >= 0; angle -= 1) {
    myservo.write(angle);
    delay(30);
  }
  delay(1000);
}
```

How the code works

The first step is to include the required Arduino library. You can also find this library under Sketch > Include Library > Servo.

```
// Include the servo library:
#include <Servo.h>
```

Next, you need to create a new object of the Servo class. In this case, I called the servo ‘myservo’ but you can use other names as well. Note that you will also have to change the name of the servo in the rest of the code.

```
// Create a new servo object:
Servo myservo;
```

After that, I defined to which Arduino pin the servo motor is connected.

```
// Define the servo pin:
#define servoPin 9
```

The statement #define is used to give a name to a constant value. The compiler will replace any references to this constant with the defined value when the program is compiled. So everywhere you mention servoPin, the compiler will replace it with the value 9 when the program is compiled.

The variable angle is used to store the current position of the servo in degrees.

```
// Create a variable to store the servo position:
int angle = 0;
```

In the setup section of the code, we link the servo object that we created to the pin that will control the servo. The attach() function also has two optional parameters, which I discuss in the section below.

```
void setup() {
  // Attach the Servo variable to a pin:
  myservo.attach(servoPin);
}
```

Control angle/position

In the first part of the loop, we simply tell the servo motor to move to a particular angle with the function write(). Note that you need a delay between the commands to give the servo motor some time to move to the set position.

```
  // Tell the servo to go to a particular angle:
  myservo.write(90);
  delay(1000);
  myservo.write(180);
  delay(1000);
  myservo.write(0);
  delay(1000);
```

Control speed

In the last part of the code, I used two for loops to sweep the servo motor back and forth. This piece of code can also be useful if you want to control the speed of the servo motor. By changing the delay value at the end of the for loop, you can adjust the speed of the servo arm.

```
  // Sweep from 0 to 180 degrees:
  for (angle = 0; angle <= 180; angle += 1) {
```

```
  myservo.write(angle);
  delay(15);
 }
 // And back from 180 to 0 degrees:
 for (angle = 180; angle >= 0; angle -= 1) {
  myservo.write(angle);
  delay(30);
 }
```

Why doesn't my servo turn a full 0 – 180 degrees?

As I discussed in the introduction, the angle of the output shaft of the servo motor is determined by the width of the electrical pulse that is applied to the control wire. Generally, a pulse width of about 1 ms (millisecond) corresponds to the minimum position, 2 ms to the maximum position, and 1.5 ms to 90° (neutral position). However, this can vary slightly between brands and even different servos of the same brand. This means that you will have to adjust the minimum and maximum values in the code to match the servo that you are using.

The Arduino Servo library makes it very easy to tune the min and max angle of the servo motor by specifying two optional parameters in the attach() function. In this function, the first parameter is the number of the pin that the servo is attached to. The second parameter is the pulse width, in microseconds (µs), corresponding to the minimum (0-degree) angle of the servo motor. The third parameter is the pulse width, in microseconds, corresponding to the maximum (180-degree) angle of the servo motor.

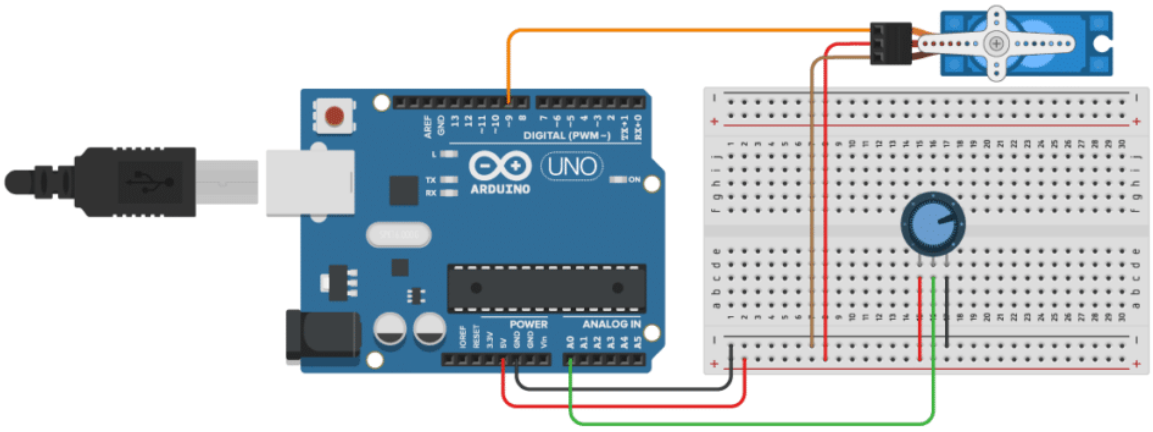
By default, the min and max pulse width is set to **544** and **2400** microseconds. These values work for most common servos, but sometimes you have to adjust the values slightly.

I recommend adjusting the min and max values in small increments (10-20 microseconds) to avoid damaging the servo. If the servo arm is hitting the physical limits of the motor, increase the min value, and decrease the max value.

```
#define servoPin 9
int min = 480;
int max = 2500;
Servo myservo;

void setup() {
  myservo.attach(servoPin, min, max);
}
```

Control a servo with a potentiometer and Arduino



Wiring diagram for controlling a servo motor with a potentiometer and Arduino.

Controlling the position of a servo motor with a potentiometer is very easy and can be very useful if you want to adjust the motor position by hand. As you can see in the wiring diagram above, the servo motor is wired in the same way as before. The only difference is that I used a breadboard to distribute the power from the Arduino.

The potentiometer has three pins, connect the outside pins to 5 V and GND. The middle pin of the potentiometer is connected to the analog pin A0 of the Arduino.

Servo motor with potentiometer Arduino example code

The example code below lets you control a servo motor with a potentiometer.

You can copy the code by clicking on the button in the top right corner of the code field.

```
/* Servo motor with potentiometer and Arduino example code. More info: https://www.makerguides.com/ */

#include <Servo.h> // include the required Arduino library

#define servoPin 9 // Arduino pin for the servo
#define potPin A0 // Arduino pin for the potentiometer

int angle = 0; // variable to store the servo position in degrees
int reading = 0; // variable to store the reading from the analog input

Servo myservo; // create a new object of the servo class

void setup() {
  myservo.attach(servoPin);
}
```

```
void loop() {
  reading = analogRead(potPin); // read the analog input
  angle = map(reading, 0, 1023, 0, 180); // map the input to a value between 0 and 180 degrees
  myservo.write(angle); // tell the servo to go to the set position
  delay(15); // wait 15 ms for the servo to reach the position
}
```

Notice that before the setup and loop section of the code a new variable `reading` is added and the potentiometer input pin is defined.

In the loop section of the code, we read the value from the analog pin A0 with the function `analogRead()`.

```
reading = analogRead(potPin); // read the analog input
```

Arduino boards contain a 10-bit analog to digital converter (ADC), so this gives us a value between 0 and 1023 depending on the position of the potentiometer.

Because the servo motor can only rotate between 0 and 180 degrees, we need to scale the values down with the map() function. This function re-maps a number from one range to another.

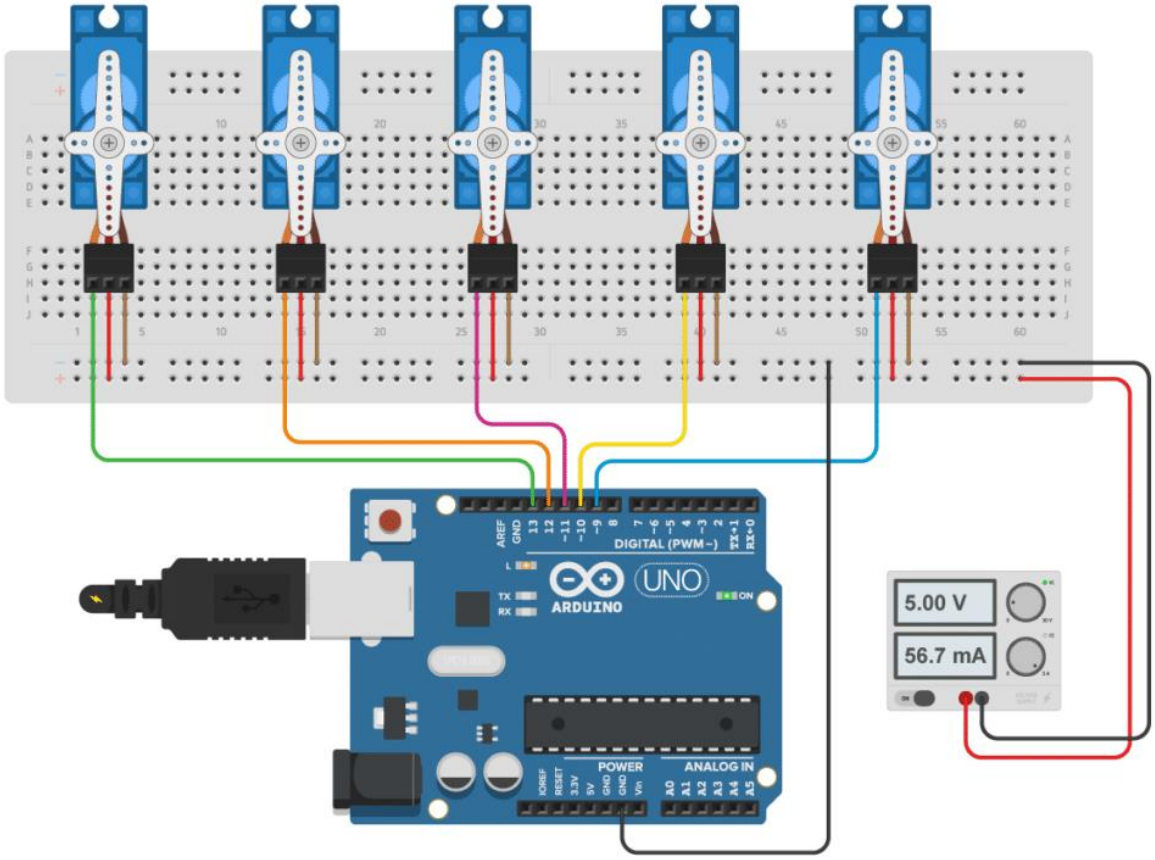
```
angle = map(reading, 0, 1023, 0, 180); // map the input to a value between 0 and 180 degrees
```

Lastly, we write the angle to the servo motor:

```
myservo.write(angle); // tell the servo to go to the set position
delay(15); // wait 15 ms for the servo to reach the position
```

Controlling multiple servo motors

Controlling multiple servos is just as easy as controlling only one but I often get questions about how to modify the code. Therefore, I have added a simple example below.

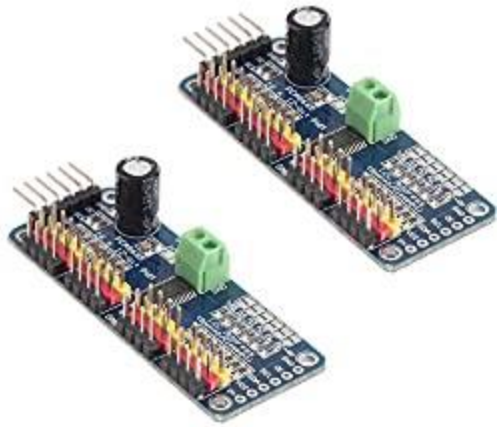


Multiple servo motors connected to the Arduino Uno and an external power supply.

Note that you will have to use an external power supply to power the servos because the Arduino can not provide enough current to power all of the motors.

For this example, we just use more Arduino pins for the additional servos. However, this means that you are limited to 12 servos when using an Arduino Uno, and you might not have enough pins left over for other components.

Another option is to use one or multiple PCA9685 PWM/servo drivers. This driver allows you to control 16 servos with just 2 pins from the Arduino by using I2C. Adafruit also sells these in the form of an Arduino shield.



PCA9685 PWM/servo drivers

Because the setup of these servo drivers is a bit more difficult, I will cover this in a separate tutorial.

Arduino with multiple servos example code

As you can see in the example below, you just have to create more objects of the Servo class with different names. You can address each servo by using the correct name in the setup and loop section of the code.

```
/* Arduino with multiple servos example code. More info: https://www.makerguides.com/ */

#include <Servo.h>

Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;

void setup()
{
  servo1.attach(9);
  servo2.attach(10);
  servo3.attach(11);
  servo4.attach(12);
  servo5.attach(13);
}

void loop()
{
  servo1.write(0);
  servo2.write(0);
  servo3.write(0);
  servo4.write(0);
```

```
  servo5.write(0);
  delay(2000);
  servo1.write(90);
  servo2.write(90);
  servo3.write(90);
  servo4.write(90);
  servo5.write(90);
  delay(1000);
  servo1.write(180);
  servo2.write(180);
  servo3.write(180);
  servo4.write(180);
  servo5.write(180);
  delay(1000);
}
```

Conclusion

In this tutorial, I have shown you how to use servo motors with Arduino. We looked at the basics of controlling the position and speed of servo motors, how to control a servo motor with a potentiometer, and how to control multiple servo motors at the same time.