# Using a Servo Motor – Arduino

Objective:

Employ new functions in an Arduino sketch to continue to improve skills in programming embedded systems. The outcome of this lab will be to control a servo motor.

About:

Servo motors can be used with the Arduino, Raspberry Pi, or any other controller in a variety of embedded system tasks. Some examples could include a motion sensor, a controller, and a servo motor used with a trash can to make an automatic motion sensor trash can. Servo motors can be used in vending machines, and even in locks.

Hardware:

SG90 Micro Servo 9g

Jumper wire

Arduino

cables

Breadboard

You can use any other servo motor, but it is important to identify the possibly different wire colors when you connect them.

Methods:

Standard Methods

attach(int)

Turn a pin into a servo driver. Calls pinMode. Returns 0 on failure.

detach()

Release a pin from servo driving.

write(int)

Set the angle of the servo in degrees, 0 to 180.

read()

return that value set with the last write().

attached()

return 1 if the servo is currently attached.

Extra Methods

refresh()

You must call this at least once every 50ms to keep the servos updated. You can call it as often as you like, it won't fire more than once every 20ms. When it does fire, it will take from .5 to 2.5 milliseconds to complete, but won't disable interrupts.

setMinimumPulse(uint16\_t)

set the duration of the 0 degree pulse in microseconds. (default minimum value is 544 microseconds)

setMaximumPulse(uint16\_t)

set the duration of the 180 degree pulse in microseconds. (default maximum pulse value is 2400 microsconds)

Safety Quirk

Even though you attach a servo, it won't receive any control signals until you send its first position with the write() method to keep it from jumping to some odd arbitrary value.

Limitations

This library does not stop your interrupts, so millis() will still work and you won't lose incoming serial data, but a pulse end can be extended by the maximum length of your interrupt handles which can cause a small glitch in the servo position. If you have a large number of servos there will be a slight (1-3 degrees) position distortion in the ones with the lowest angular values.

## Part 1: Setting up the Arduino, Servo, and Code

Connecting the cables as following to the breadboard, then to the Arduino. You will use jumper cables for this.

* Power: Connect the red from servo to +5V on Arduino
* Ground: Connect black/brown from servo to Gnd on Arduino.
* Signal: Connect white/orange from servo to Analog in 0 on Arduino.

## Activities

Exercise 1 – Using Sweep

Change your breadboard to reflect the following sketch or change the sketch to match your pinout.

#include <Servo.h>

int servoPin = 9;

Servo servo;

int angle = 0; // servo position in degrees

void setup()

{

servo.attach(servoPin);

}

void loop()

{

// scan from 0 to 180 degrees

for(angle = 0; angle < 180; angle++)

{

servo.write(angle);

delay(2);

}

// now scan back from 180 to 0 degrees

for(angle = 180; angle > 0; angle--)

{

servo.write(angle);

delay(20);

}

}

You will now speed up and slow down the servo.

Explain how you accomplished this.

In order to speed up and slow down the servo I simply changed the delay so that any angles from 0 to 180 degrees have a delay of 2ms and those from 180 to 0 degrees have a delay of 20ms.

Exercise 2 - Row

You will now add a Potentiometer with knob to your lab.

Add this to the center of your board, then add a lead from its slider to A0 on the Arduino.

For this exercise we are adding a second variable of potPin. Note this pin takes analog readings, and this will give us a value of 0-1023, but the servo rotates 180 degrees. You will need to scale this down (divide by 6). The value we will use is between 0-170.

#include <Servo.h>

int potPin = 0;

int servoPin = 9;

Servo servo;

void setup()

{

servo.attach(servoPin);

}

void loop()

{

int reading = analogRead(potPin); // 0 to 1023

int angle = reading / 6; // 0 to 170-ish

servo.write(angle);

}

Explain the relationship between the potPins position and how the servo responded when you moved the pin.

With the potentiometer turned all the way down the servo motor arm would sit on the right side (at 0 degrees), and with the potentiometer turned all the way up the servo motor arm would sit on the left side (at ~180 degrees)

Exercise 3

Modify the code used in Exercise 2 so that instead of using the knob you added, it will use the Serial Monitor. For this you will use the function Serial.parseInt()

Explain what you needed to do, include in your answer the code you had after the change.

I needed to include Serial.begin(9600) in my setup method to establish communication over the serial port at 9600 baud, and then in the loop method I changed the declaration for the int member reading to Serial.parseInt() and then set the angle equal to this exactly by removing / 6.

#include <Servo.h>

// int potPin = 0;

int servoPin = 9;

Servo servo;

void setup()

{

  servo.attach(servoPin);

  Serial.begin(9600);

}

void loop()

{

  int reading = Serial.parseInt(); // 0 to 180

  int angle = reading; // 0 to 170-ish

  servo.write(angle);

}

**Picture of final project:**

