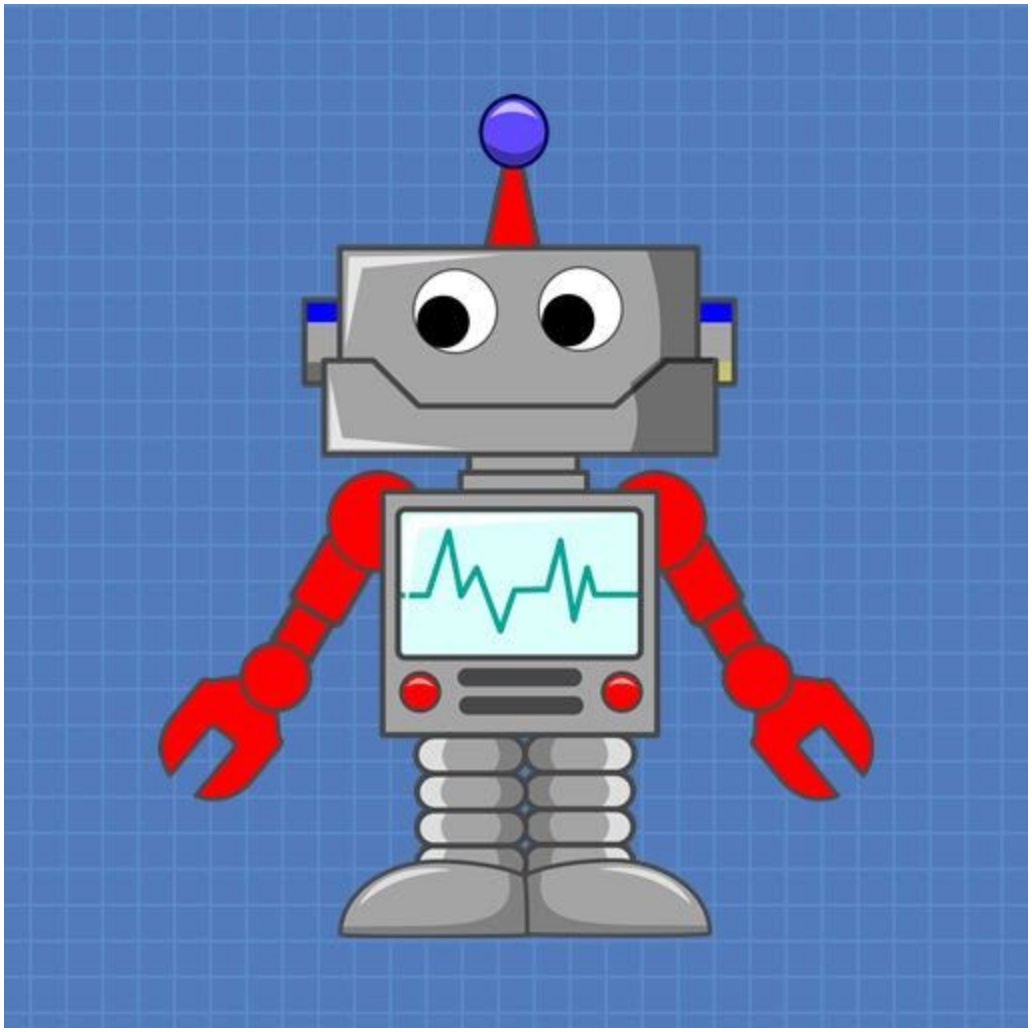
# Analog Feedback Servo Motor



DroneBot Workshop Tutorial

https://dronebotworkshop.com

# Table of Contents

Today we will look at a special type of servo motor, an “Analog Feedback” Servo. This type of servo motor has an additional connection that outputs the motor position. Interestingly, this motor can also be used as an input device.

# Introduction

Servo motors are pretty popular items here in the DroneBot Workshop. I’ve used them in several projects and demonstrations, and have also produced an article and video that covers them in-depth.

There is a very good reason for their popularity, both around the workshop and among hobbyists in general. Servo motors are inexpensive, have a great size-to-torque ratio and, most importantly, can be easily controlled with a great deal of precision.


But despite all of their virtues servo motors do have some limitations. They are essentially a “closed-loop” system, so once you send them a command to move into a

specific position you need to depend upon the servo motors internal controller to perform this accurately.

You have no way of easily determining a servo motors' actual shaft position, you have to assume that the servo has obeyed your command and is now resting where you want it to. This may not be true, especially if the motor has been subject to an external force.

An Analog Feedback Servo Motor can resolve these issues.

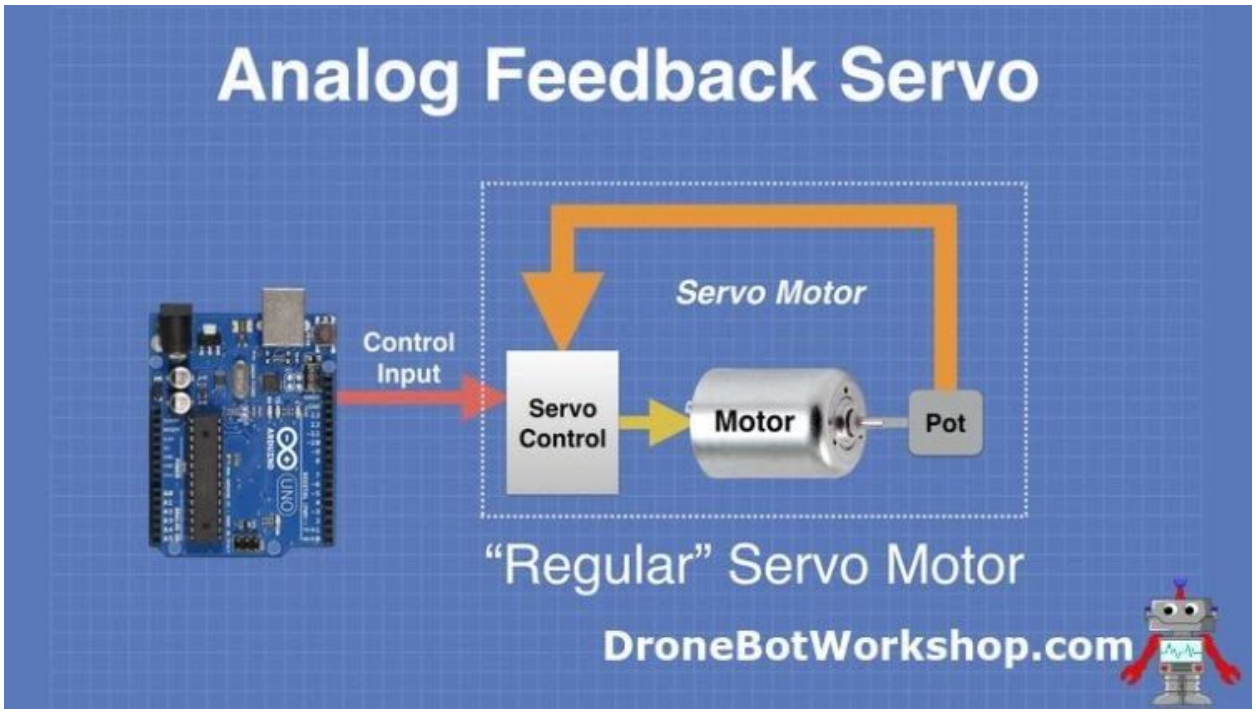
## Analog Feedback Servo

An Analog Feedback Servo Motor is essentially a "regular" servo motor with an additional connection. This connection is an output from the servo motors internal potentiometer.

To understand this a bit better let's take a look at how a "regular servo motor operates.

# Regular Servo Motor Operation

A servo motor operates using what is sometimes called a “closed-loop” system.




In this type of system, a microcontroller or other device is connected to the servo motors control input. A PWM (Pulse Width Modulation) signal is sent over this connection, the width of the pulse determines the desired motor position.

This signal is sent to the servo motors internal controller. The controller drives the motor, which is a DC motor with a lot of gearing on its output shaft. Most servo motors limit the amount of rotation on the output shaft, usually to 180 or 270 degrees of rotation.

The output shaft is also coupled to a potentiometer, which measures the shaft position. The output of the potentiometer is sent back to the internal controller, to let it know the

position of the motor. The controller uses this information to move the shaft into the correct position.

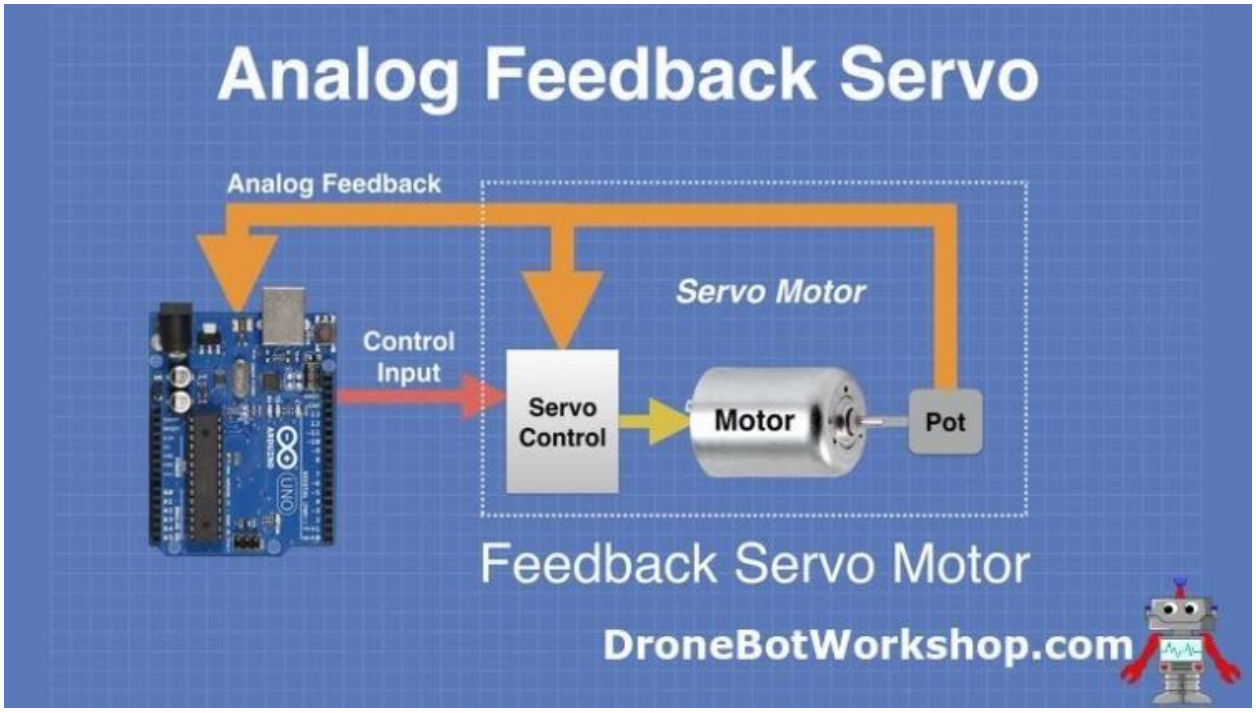
For the most part, this arrangement works pretty well, however, a few issues can arise:

- The accuracy of the positioning is determined by both the tolerance of the potentiometer and the resolution of the internal servo controller. Inexpensive hobbyist servo motors don't necessarily use the highest tolerance parts.
- After sending a command to move the shaft into a certain position we are "blind". We have no idea when the shaft has stopped in position, or even if it is in the correct position.
- If the components in the servo motor drift due to temperature or other factors the servo can find itself in between two reference points. This can cause the motor to "chatter".
- We have no way of dynamically measuring the current motor shaft position. If an external force moves the motor we can be completely unaware of it.

In many cases, most of these issues won't affect the performance of our project. But if precision positioning is a requirement then the Analog Feedback servo motor has some advantages.

# Analog Feedback Servo Operation

The Analog Feedback Servo Motor is simply a “regular” servo motor that has been modified to bring the output from its internal positioning potentiometer out to an external connection. As the potentiometer rotates it will change the voltage on this output, and this can be used to dynamically determine the motor shaft position.




By outputting the position information we gain a number of advantages over the “closed-loop” system used in a regular servo motor:

- The feedback output gives you a dynamic, “real-time” reading of the position of the motor shaft.
- It allows you to use the analog to digital converter (ADC) in your microcontroller to fine-tune the motor position. This is often of higher precision than the one in the servo motors internal controller.
- In many designs, it is customary to insert a time delay after driving the servo motor, in order to allow the motor to settle into position. With an analog feedback

servo, you can read the motor position and determine exactly when it has arrived at its destination.
- You can also use the servo motor as an input device, as the feedback signal output will vary as the motor shaft is turned.

Of course, it still isn't perfect, the feedback signal is coming from the same positioning potentiometer and is still affected by the pots tolerance.

Essentially, an Analog Feedback Servo Motor brings the external controller into the feedback loop.

# S1213 Analog Feedback Servo Motor

You can actually modify a standard servo motor to become an analog feedback motor, as long as you're willing to open up your motor and thus violate any warranty that it came with. Since most hobbyist servo motors are pretty cheap this usually isn't a great concern.

By connecting a wire from the center, or wiper, of the internal potentiometer, you can gain access to the feedback signal. You may need to "clean" the signal with a small filtering capacitor.

But the easiest way to obtain an analog feedback servo motor is to just go and buy one! They are not really that much more expensive than regular servo motors. In fact, you can ignore the feedback output if you wish and use them as a conventional servo motor.

For our experiments, we will be using the S1213 Analog Feedback Servo Motor.


The S1213 is a mid-sized plastic-gear servo motor with an additional feedback output. It operates on 5- VDC and has a torque rating of 4.5 kg-cm, which is robust enough for many applications.

The motor has four connections, three of them are on a standard servo cable and the feedback connection is on a separate wire.

- Feedback (White Wire) – This is the analog feedback output.
- Control (Orange Wire) – This is the control input, it accepts a logic-level PWM signal to position the motor.
- Power (Red Wire) – This is the 5-6 VDC power input.
- Ground (Brown Wire) – The ground connection.

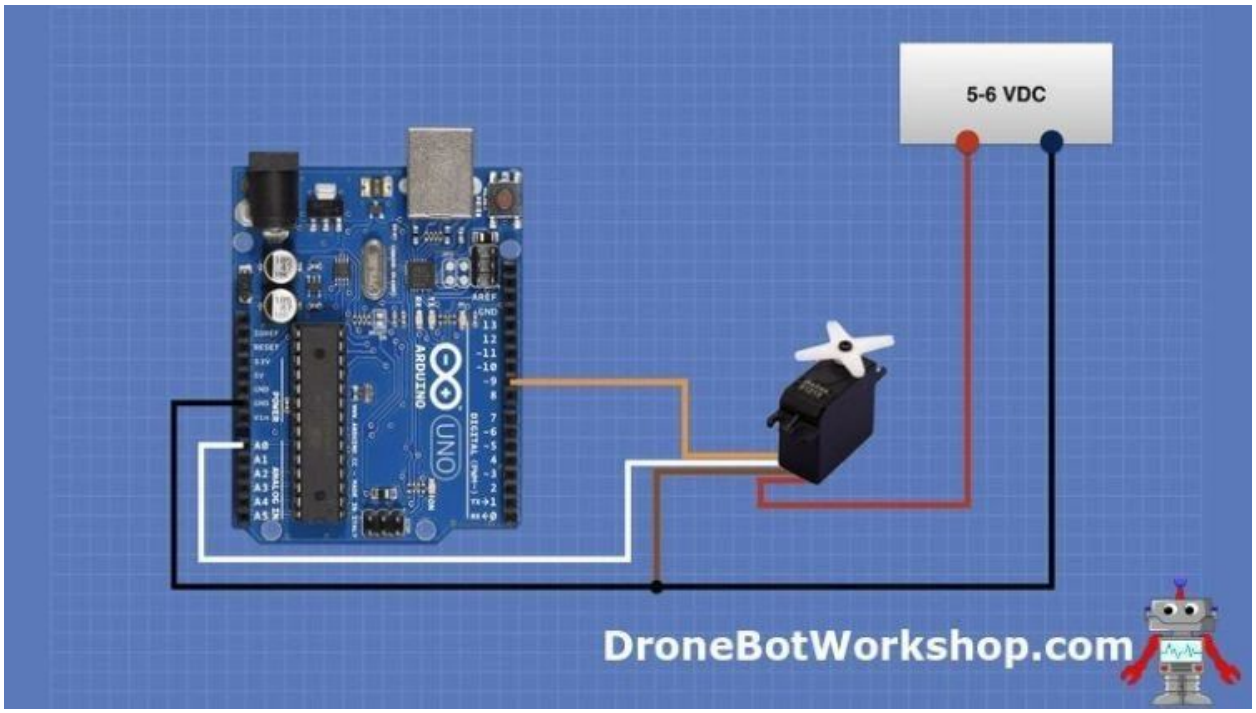
If you need a smaller analog feedback servo motor you might want to take a look at the S1123 instead, it uses the same form factor as the popular SG90 servos often used in hobbyist projects.

# Calibrating the Servo Motor

Our first experiment using the analog feedback servo motor will be a simple calibration sketch.  We will use it to correlate the position of the motor with the value returned by the feedback signal.

## Arduino Hookup

Wiring up our demo is quite simple.




In addition to the Arduino Uno and the S1213 analog feedback servo motor you'll also need a power supply. While you could use the 5-volt output from the Arduino it is much better to use a separate supply to keep any induced electrical noise from the motor away from the Arduino.

If you do decide to use the Arduino 5-volt output you should place a small electrolytic capacitor across the line, close to the motor. This will help reduce some of the electrical noise and will also provide a small current reservoir for the motor.

After everything is hooked up it's time to load a sketch to the Arduino.

## Motor Calibration Sketch

The calibration sketch is pretty simple. It homes the motor to the zero-degree position, then steps it in increments of 5 degrees. On each step, the value from the feedback signal is read and displayed on the serial monitor.  When we reach the 180-degree point (the limit of the servo motor travel) it returns to the zero position.

A one-second delay has been added in between each step, in order to make the display more readable and to give the motor time to settle into position, As the motor doesn't need a full second to stabilize you can always reduce this value if you wish.

```
/*
  Analog Feedback Servo Calibration Demo
  feedback_servo_calib.ino
  Uses S1213 Analog Feedback Servo Motor
  Results displayed on Serial Monitor

  DroneBot Workshop 2019
  https://dronebotworkshop.com
*/

// Include Arduino Servo Library
#include <Servo.h>

// Control and feedback pins
int servoPin = 9;
int feedbackPin = A0;

// Value from feedback signal
int feedbackValue;

// Create a servo object
Servo myservo;

void setup()
{
  // Setup Serial Monitor
  Serial.begin(9600);

  // Attach myservo object to control pin
```

```
30  myservo.attach(servoPin);
31
32  // Home the servo motor
33  myservo.write(0);
34
35  // Step through servo positions
36  // Increment by 5 degrees
37  for (int servoPos = 0; servoPos <=180; servoPos = servoPos + 5){
38
39    // Position servo motor
40    myservo.write(servoPos);
41    // Allow time to get there
42    delay(1000);
43
44    // Read value from feedback signal
45    feedbackValue = analogRead(feedbackPin);
46
47    // Write value to serial monitor
48    Serial.print("Position = ");
49    Serial.print(servoPos);
50    Serial.print("\t");
51    Serial.println(feedbackValue);
52
53  }
54
55  // Move back to home position
56  myservo.write(0);
57
58  // Print to serial monitor when done
```

```
59    Serial.println("Finished!");
60
61  }
62
63  void loop()
64  {
65    // Nothing in loop
66  }
```

The sketch starts by including the Arduino Servo Library, which is included within your Arduino IDE.

We then define a few integers to represent the pins used by the analog feedback servo motor and the value obtained from the motors' feedback output.

We create an object to represent the servo motor and then move into the Setup.

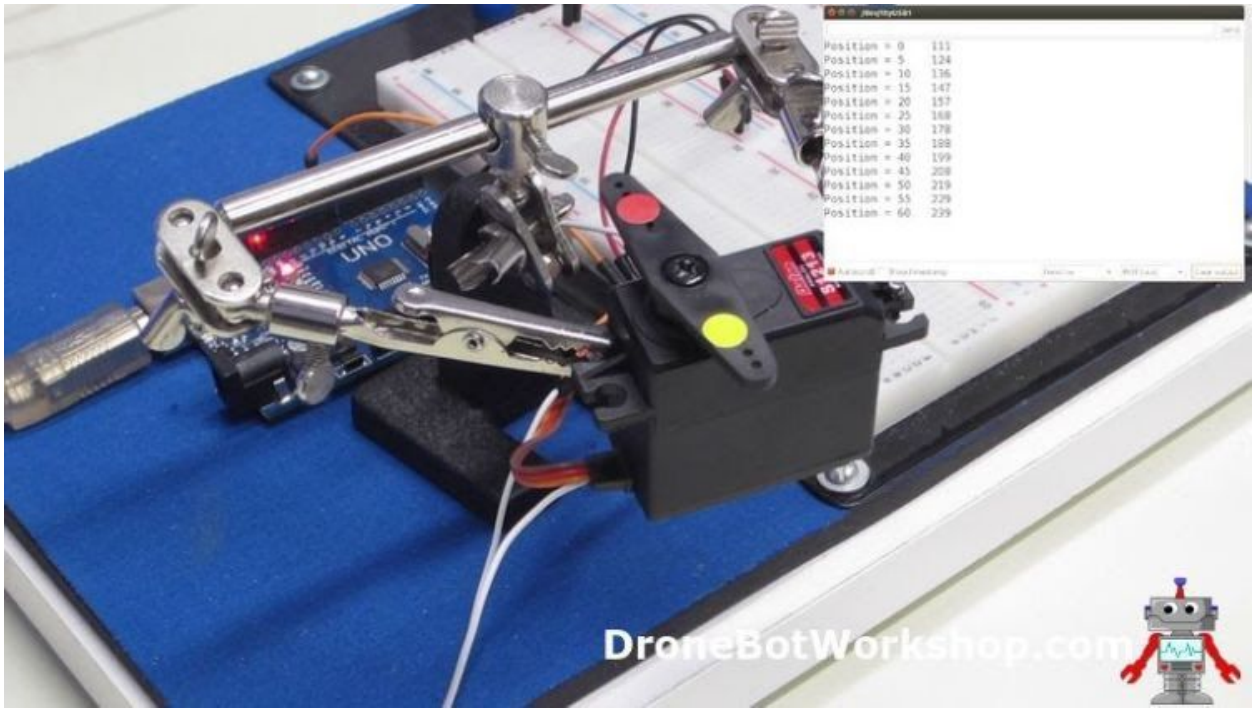
In the Setup, we start the serial monitor and attach the servo object to pin 9, where the servo control line is connected.

Next we "home" the servo motor by sending it to the zero-degree position.

We then go through a for-next loop using the *servoPos* variable which represents the servo position. We step through the for-next loop in increments of 5.

On each increment, we position the servo, allow a second for everything to stabilize and then read the value. We then print the results to the serial monitor.

Once the *servoPos value* reaches 180 we finish and home the motor to zero again.




Upload the sketch to your Arduino. Open the serial monitor and observe both the monitor and the motor itself. You will see the analog values returned from the motor.

You may use these values as calibration points for future analog feedback servo motor experiments.

# Using Servo as Input Device for Recording Position

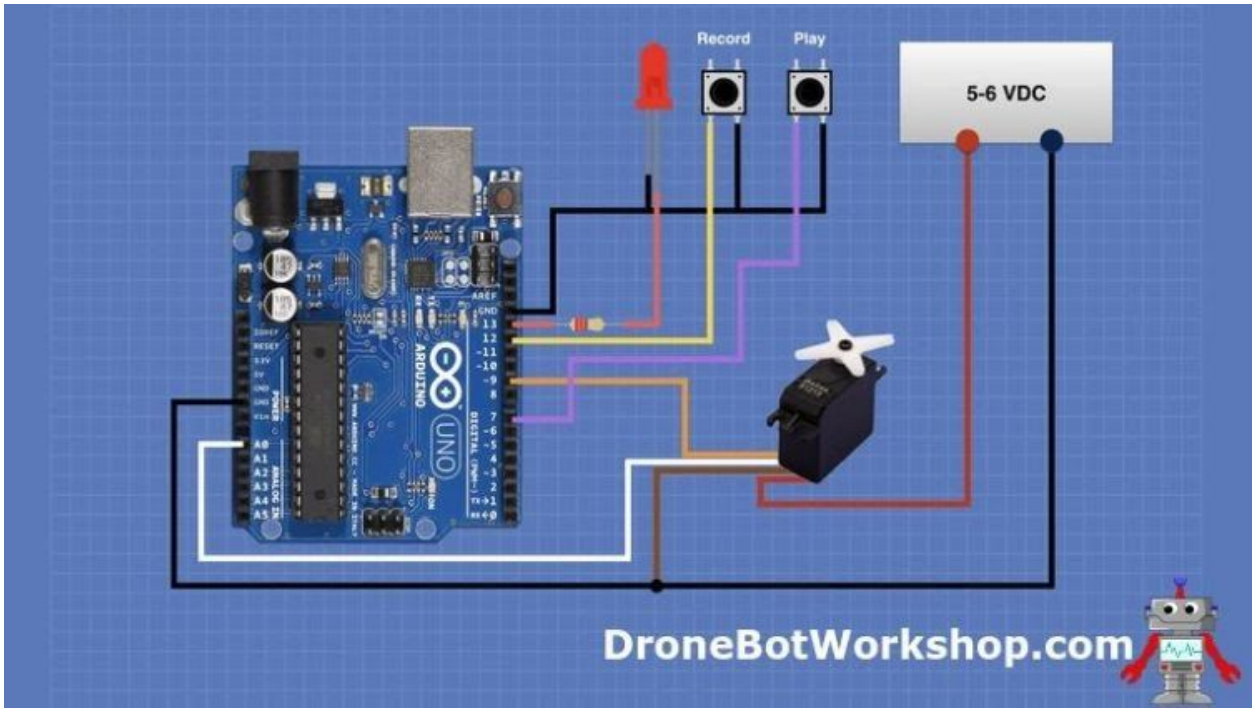
The inclusion of the feedback connection from the servo motors internal potentiometer gives an analog feedback servo motor the unique ability to also be used as an input device.

This ability can be exploited to create a servo motor project that “memorizes” its movements. A great application for this technique would be in the construction of a

robotic arm.  You could manually move the arm in a specific pattern and then playback those movements.

## Servo Record Hookup

The following project is not of my own creation. It is an open-source project developed by Adafruit, and the code is available on GitHub.




The only difference between my wiring diagram and the one provided by Adafruit is that theirs does not use a separate servo motor power supply.  You can wire it either way.

The connection to the servo and its power supply are identical to the previous experiment, so if you still have that one on your breadboard then you can just add to it.

You'll need a few additional components, specifically an LED (any size or color) and two pushbutton switches.  One switch will be for *Record*, the other for *Play*.

Note that there are no pull-up resistors used for the pushbutton switches. That will be explained when we look at the sketch.

After you get the circuit wired up we can use the code supplied by Adafruit on GitHub to run everything.

## Servo Recording Sketch

Adafruit's servo recording sketch is shown below.

```
1  // Example code for recording and playing back servo motion with a
2  // analog feedback servo
3  // http://www.adafruit.com/products/1404
4
5
6  #include <Servo.h>
7  #include <EEPROM.h>
8
9  #define CALIB_MAX 512
10 #define CALIB_MIN 100
11 #define SAMPLE_DELAY 25 // in ms, 50ms seems good
12
13 uint8_t recordButtonPin = 12;
14 uint8_t playButtonPin = 7;
15 uint8_t servoPin = 9;
16 uint8_t feedbackPin = A0;
17 uint8_t ledPin = 13;
18
19 Servo myServo;
20
21 void setup() {
22   Serial.begin(9600);
23   pinMode(recordButtonPin, INPUT);
24   digitalWrite(recordButtonPin, HIGH);
25   pinMode(playButtonPin, INPUT);
26   digitalWrite(playButtonPin, HIGH);
27   pinMode(ledPin, OUTPUT);
28
29   Serial.println("Servo RecordPlay");
```

```
30  }
31
32  void loop() {
33  if (! digitalRead(recordButtonPin)) {
34     delay(10);
35     // wait for released
36     while (! digitalRead(recordButtonPin));
37     delay(20);
38     // OK released!
39     recordServo(servoPin, feedbackPin, recordButtonPin);
40  }
41    if (! digitalRead(playButtonPin)) {
42     delay(10);
43     // wait for released
44     while (! digitalRead(playButtonPin));
45     delay(20);
46     // OK released!
47     playServo(servoPin, playButtonPin);
48  }
49  }
50
51  void playServo(uint8_t servoPin, uint8_t buttonPin) {
52    uint16_t addr = 0;
53    Serial.println("Playing");
54
55    myServo.attach(servoPin);
56    while (digitalRead(buttonPin)) {
57      uint8_t x = EEPROM.read(addr);
58      Serial.print("Read EE: "); Serial.print(x);
```

```
59      if (x == 255) break;
60      // map to 0-180 degrees
61      x = map(x, 0, 254, 0, 180);
62      Serial.print(" -> "); Serial.println(x);
63      myServo.write(x);
64      delay(SAMPLE_DELAY);
65      addr++;
66      if (addr == 512) break;
67    }
68    Serial.println("Done");
69    myServo.detach();
70    delay(250);
71  }
72
73  void recordServo(uint8_t servoPin, uint8_t analogPin, uint8_t buttonPin) {
74    uint16_t addr = 0;
75
76    Serial.println("Recording");
77    digitalWrite(ledPin, HIGH);
78
79    pinMode(analogPin, INPUT);
80    while (digitalRead(buttonPin)) {
81        uint16_t a = analogRead(analogPin);
82
83        Serial.print("Read analog: "); Serial.print(a);
84        if (a < CALIB_MIN) a = CALIB_MIN;
85        if (a > CALIB_MAX) a = CALIB_MAX;
86        a = map(a, CALIB_MIN, CALIB_MAX, 0, 254);
87        Serial.print(" -> "); Serial.println(a);
```

```
88          EEPROM.write(addr, a);
89          addr++;
90          if (addr == 512) break;
91          delay(SAMPLE_DELAY);
92        }
93        if (addr != 512) EEPROM.write(addr, 255);
94
95        digitalWrite(ledPin, LOW);
96
97        Serial.println("Done");
98        delay(250);
99      }
100
```

The sketch starts by loading the Servo Library and the EEPROM Library. Both of these libraries are included within your Arduino IDE.

Next, we define some constants to use as calibration values. You could, if you wish, substitute the calibration values you arrived at in the previous experiment to improve recording accuracy.

Next integer variables are assigned to the pins for the pushbuttons, the servo connections, and the LED connections, and we create an object to represent the servo motor.

In the Setup, we set the serial monitor up, And then we do something interesting to get around the lack of pullup resistors.

First, the *pinMode* of the switch pin is set (this is done for both pushbuttons, but I'll only describe one as they are identical in this respect) as an input. Then we write to the same pin and bring it HIGH.

While it may seem a bit counterintuitive to write to a pin you just defined as an input, it actually works. And since the pin is now HIGH it acts the same as if it was being pulled up to 5-volts!

We finish the Setup by defining the LED on pin 13 as an output.

In the Loop, we look at each pushbutton switch and test it to see if it has gone LOW, which would indicate that it has been pushed. We wait a bit to see if it gets released and then call a function – which function we call depends upon which pushbutton was pressed.

If the *Record* pushbutton is pressed we call the *recordServo* function. In this function, we turn on the LED and then read the voltage returned by the feedback signal and map it to a value between 0 and 254. We then store that in the EEPROM.

If you are wondering why the value is not mapped to a range of 0-180, which would match the servo values, it's done this way because it gives us more resolution.

If we get past the 512 entry point then our EEPROM will be full (assuming we are using an Arduino Uno, other Arduino's have different amounts of internal EEPROM). At this point, we turn off the LED.

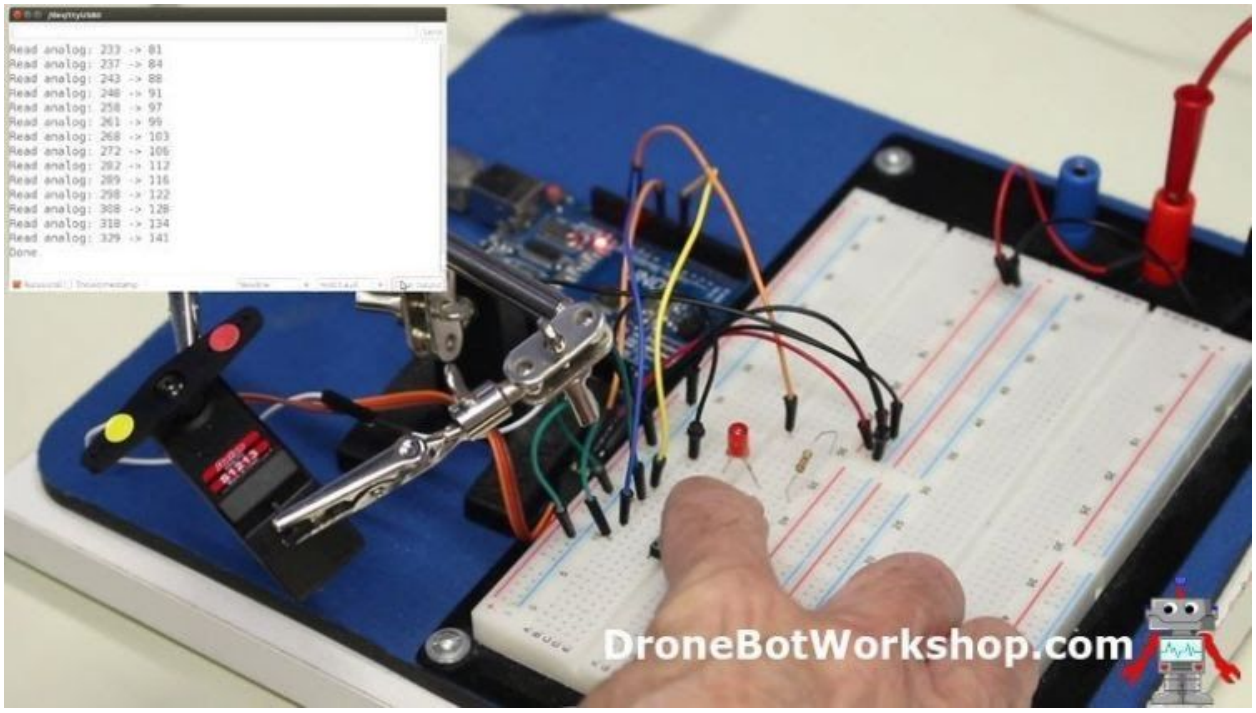
When the Play pushbutton is pressed we call the *playServo* function. In this function, we read the EEPROM and map its values to a range of 0-180. We then use that value to drive the servo motor and position it.

Both functions also display their status on the serial monitor.

# Testing the Servo Recorder

Testing the sketch is pretty easy. Load it onto the Arduino, and also open your serial monitor.

Now press the *Record* button, you should observe the LED illuminating.  Twist the motor shaft into different positions, as the shaft is geared it will be a bit difficult to turn but you can do it.  You can observe the serial monitor while you turn the motor shaft. Do this until the LED turns off, it will take about 15 seconds.




When you have finished recording press the *Play* pushbutton. You should observe the motor moving in the same pattern you recorded.

You can repeat playback as many times as you wish. When you record again the previous recording will be overwritten.

You could increase the time limit on the recording by adding external EEPROM or by using an SD or microSD card.

This sketch would be a great basis for building an automatic animated display!

# Conclusion

The analog feedback servo motor is a rather unique component that can improve or enhance your servo-based projects. The ability to use the motor as an input device as well opens up new design possibilities.

I hope you enjoyed the article!