# Arduino Advanced Motors

Written by Ousema Zayati, contributions by Connor Sheeran, and Mark Webster

***Summary***

Learn to use an Arduino to drive stepper and servo motors without the aid of a library. Also learn to control valves and muscle wires.

***Recommended Proficiencies***

Ability to write code and upload it to a microcontroller.

Power supply safety.

Basic Motors Workshop completion.

***Background***

You will learn to control a variety of electromagnetism-based actuators without relying on a library like you would have done in previous workshops.

***Materials***

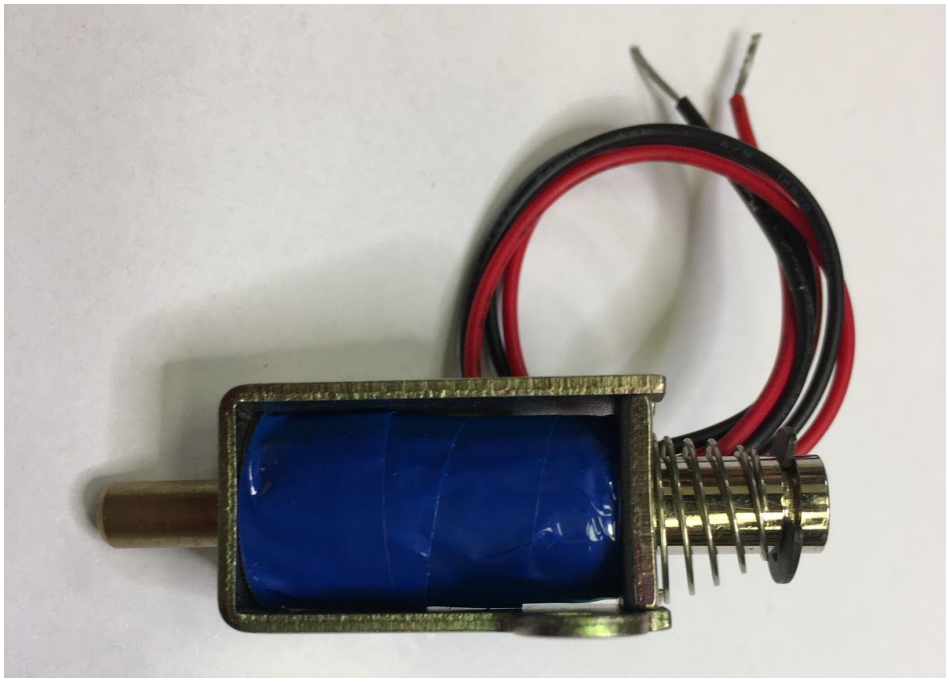
- Arduino microcontroller
- Power supply
- Breadboard
    - Jumper wires
    - Male-female ribbon cables
- Button switch
- Potentiometer
- Solenoid

    - Optional NPN power transistor
- Muscle wire
- SG90 servo
- Lab Stepper
    - Motor controller

# 1: Solenoids and Binary Valves

***Principles***

Solenoids take a binary supply of power and turn it into a binary action. The most common applications are within valves and relays. The most common configuration sets an electromagnet around a hollow channel. Residing inside the channel is a sliding actuator with a ferromagnetic core, resting against a spring. On activation, the electromagnet pulls the actuator against the spring and further into the body of the solenoid, closer to the electromagnet's center. Deactivation causes the spring to return the actuator to its rest state.



For more information on solenoids, see the Wikipedia article:

https://en.wikipedia.org/wiki/Solenoid

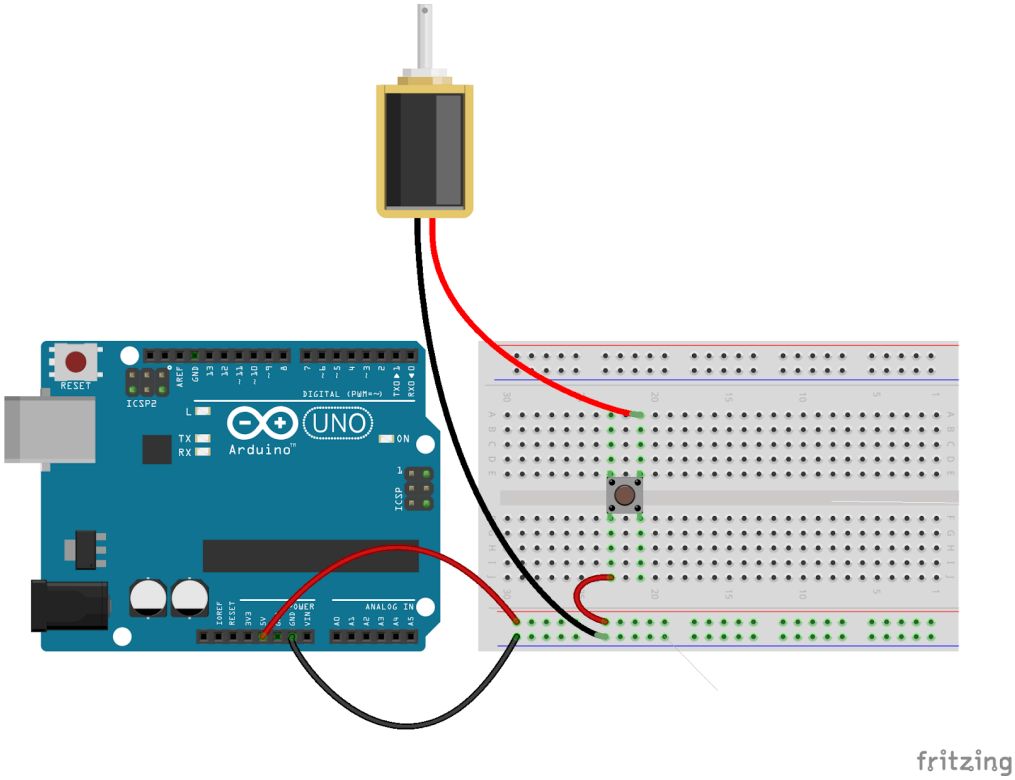
## Exercise 1: Solenoid Circuit

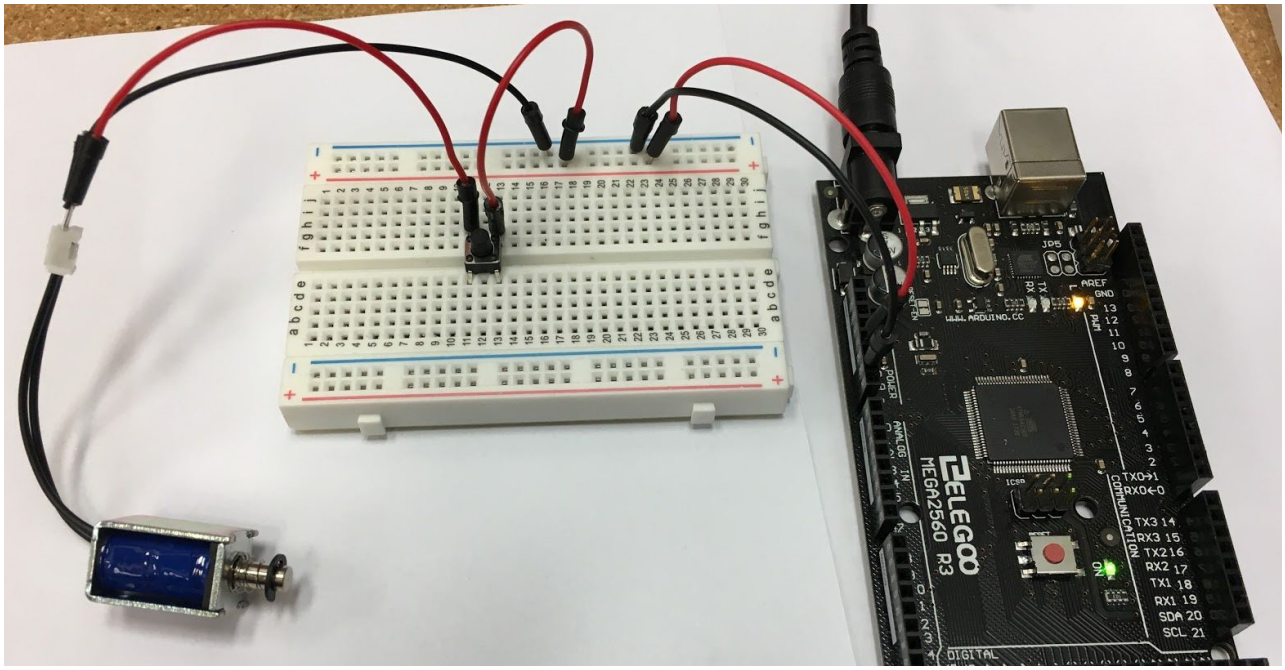
***Setup***

*Use a pushbutton to power the solenoid.*

If the solenoid needs more than +5 V or more current than the Arduino can provide, use a proxy NPN transistor. If voltage control is needed, read the button from a microcontroller and use a motor controller as the proxy.

In this example, the Arduino is only used as a +5 V power source. If desired, the Arduino could read the button state on a digital input pin, and a digital output pin would trigger the solenoid to engage. An Arduino might be useful if a sequence of solenoids was to be engaged, or if a sensor reading such as a moisture sensor, would trigger the solenoid to open a water value.

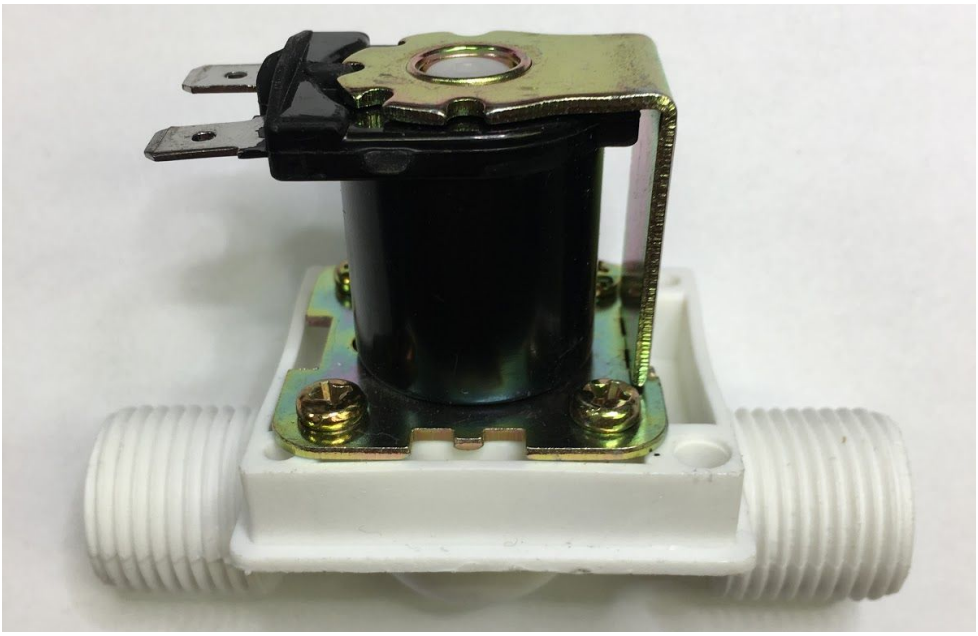




(Photographed: Stealing 5V power from an Arduino.)

## Real World Example: Solenoid water valve

**Minimum hardware:** Arduino or other microcontroller, NPN power transistor, 12V battery, 12V solenoid valve, water tank and tubes



**Applications:** Gardens, plumbing, or hydroponics systems often use automated watering systems. With these setups, we can create an automatic watering schedule in the same manner as a sprinkler. Obviously, the portability of this example has useful tradeoffs when compared to a mains-connected system.

# 2: Muscle Wires

***Principles***

Muscle wires create motion by assuming their strongest state and contracting into a “memorized” shape when a specific electrical current or heat is applied. Opposite of what we would normally expect, the 50/50 Nickel/Titanium alloy contracts when heated to about 100C or when energized. Interatomic attraction causes it to reset most deformations made to it in its weaker cooled state, as long as its crystalline structure is not broken.

Further heating causes it to behave more as expected of a metal. Heating to about 500C allows one to set the memory shape of the material, as its crystal structure is sufficiently weakened.

## Exercise 2: Muscle Wire (When materials become available.)

Shape memory wires will change shape when a DC current is applied, and return to their original shape when the current is removed.

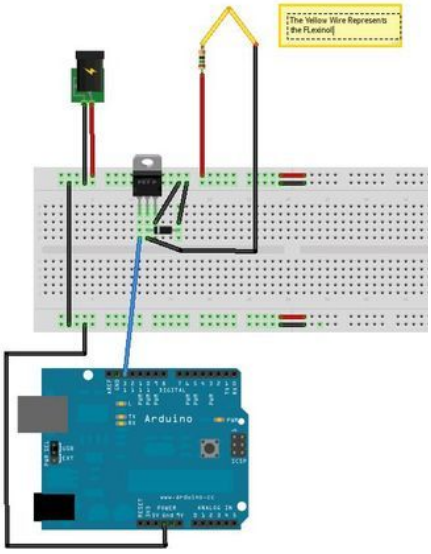
For more information, consult the Wikipedia page
https://en.wikipedia.org/wiki/Shape-memory_alloy

***Setup***

*Use a pushbutton to tell an arduino to power the wire through a proxy.*

The output voltage from the Arduino goes through a transistor to increase the current supplied to the flexible muscle wire. Typically flexinol contracts 2% - 5% when current is applied. The resistance of the wire varies with diameter. A current limiting resistor is used to prevent damage to the muscle wire.

A digital pin is set for input from the pushbutton. and a digital pin is for output. When the button is pressed, the output pin is set high.




(replace)

<FRIZING>

## 3: Stepper Motors Without a Library

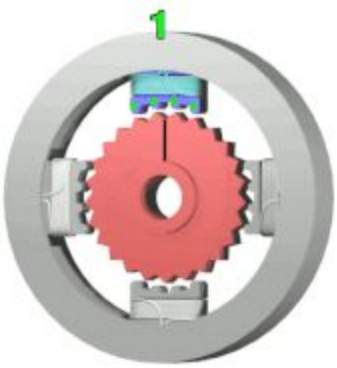
***Principles***

Stepper motors are like normal motors in that they energize electromagnets to pull the shaft further through its rotation. They diverge conceptually in that the control of coil activation is removed from the motor itself and put in the hands of the driver. The most common configuration uses a 90 degree offset of coils around a crankshaft of sorts. This is then geared

down to create a small step on the drive shaft. To turn the motor, coils are activated in sequence to move the crankshaft 90 degrees to the next step repeatedly. This coil activation will be controlled by your program.

Wikipedia describes the stepper motor in depth.

https://en.wikipedia.org/wiki/Stepper_motor




Each magnet is activated in turn which rotates the shaft one “step”.

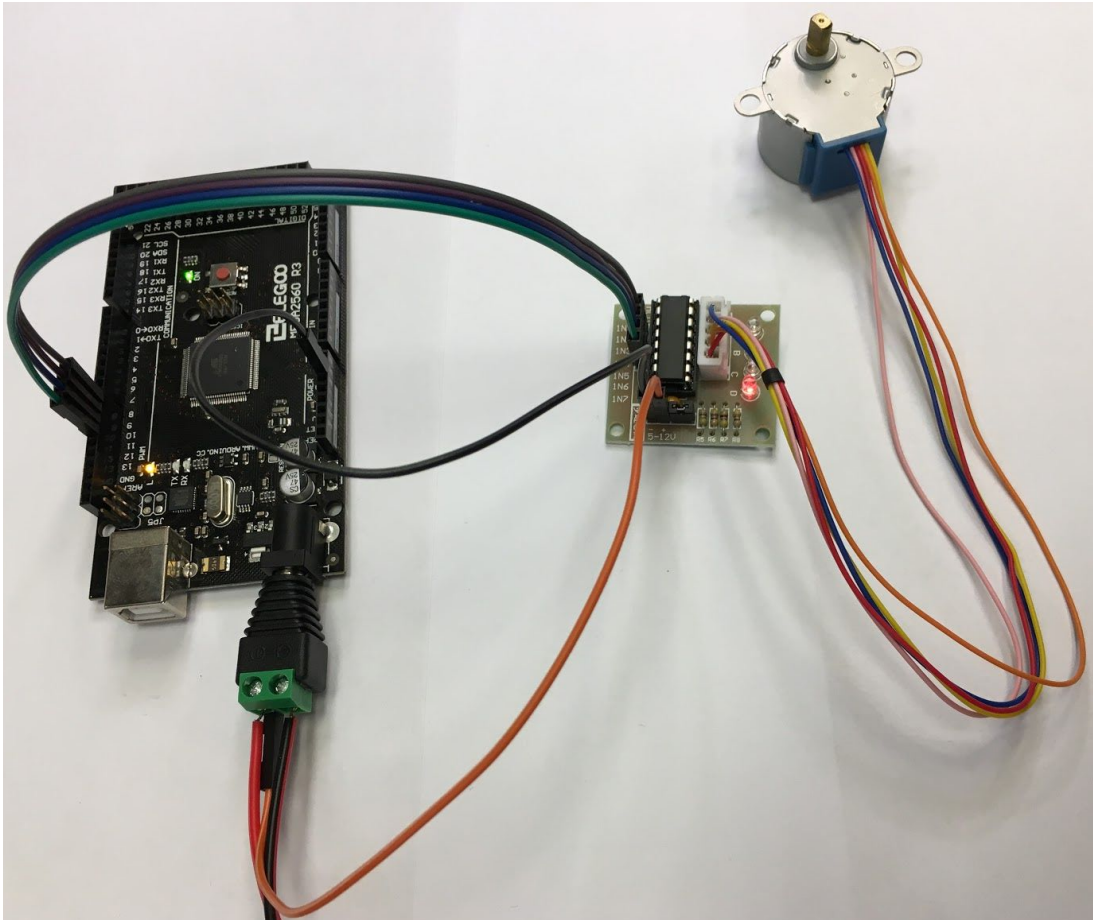
## Exercise 3: Stepper Circuit and Program

***Setup***

*Drive the stepper motor through steps based on parameters defined in the code.*

There are 4 magnets which are connected to 4 pins of the Arduino. These are specified in the void setup() function. In the void loop() function, each magnet (pin) is activated or set high for 5 ms, then turned off for 100 ms before the next magnet is activated. The less time between successive magnet (pin) activations, the faster the rotation.

<FRIZING>



**Stepper Code Answer Key:**

```
int p1 = 4;
int p2 = 5;
int p3 = 6;
int p4 = 7;
int motorPulseMS = 5;
int downTimeMS = 100;  //Zero for max speed

void setup() {
  Serial.begin(9600);  //Initialize serial for debugging
  pinMode(p1, OUTPUT);  //Set up pins
  pinMode(p2, OUTPUT);
  pinMode(p3, OUTPUT);
```

```
  pinMode(p4, OUTPUT);
  digitalWrite(p1,LOW);  //Initialize pins to LOW
  digitalWrite(p2,LOW);
  digitalWrite(p3,LOW);
  digitalWrite(p4,LOW);
}

void loop() {
  digitalWrite(p1, HIGH);  //Pulse first coil positive
  delay(motorPulseMS);
  digitalWrite(p1, LOW);
  delay(downTimeMS);

  digitalWrite(p2, HIGH);  //Pulse second coil positive
  delay(motorPulseMS);
  digitalWrite(p2, LOW);
  delay(downTimeMS);

  digitalWrite(p3, HIGH);  //Pulse first coil negative
  delay(motorPulseMS);
  digitalWrite(p3, LOW);
  delay(downTimeMS);

  digitalWrite(p4, HIGH);  //Pulse second coil negative
  delay(motorPulseMS);
  digitalWrite(p4, LOW);
  delay(downTimeMS);
}
```

## 4: Servo Motors Without a Library

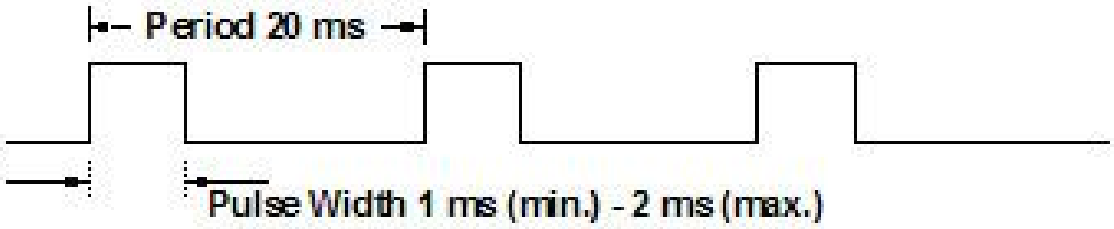
***Principles***

Servos take a position signal and power as inputs and output movement to a desired position. The signal is a specially formed square wave (ie a complex PWM), with a duty cycle varied between 5% and 10%, and a period of 20 milliseconds. (Please ask your instructor to define these terms if you are unsure.) Internally, they use a DC motor and feedback loop circuit

to move the output shaft until it reaches the position determined by the signal wave. The pinout is Signal/Power/Ground. On SG90 servos these are Orange/Red/Brown respectively.

More details about servo motors can be found in the Wikipedia

https://en.wikipedia.org/wiki/Servomotor




## Exercise 4: Servo Circuit and Program

***Setup***

*Define the pwm signal output to the servo based on the position of a potentiometer.*

Create a continuous sequence of square pulses, with a positive pulse width between 1 ms and 2 ms.

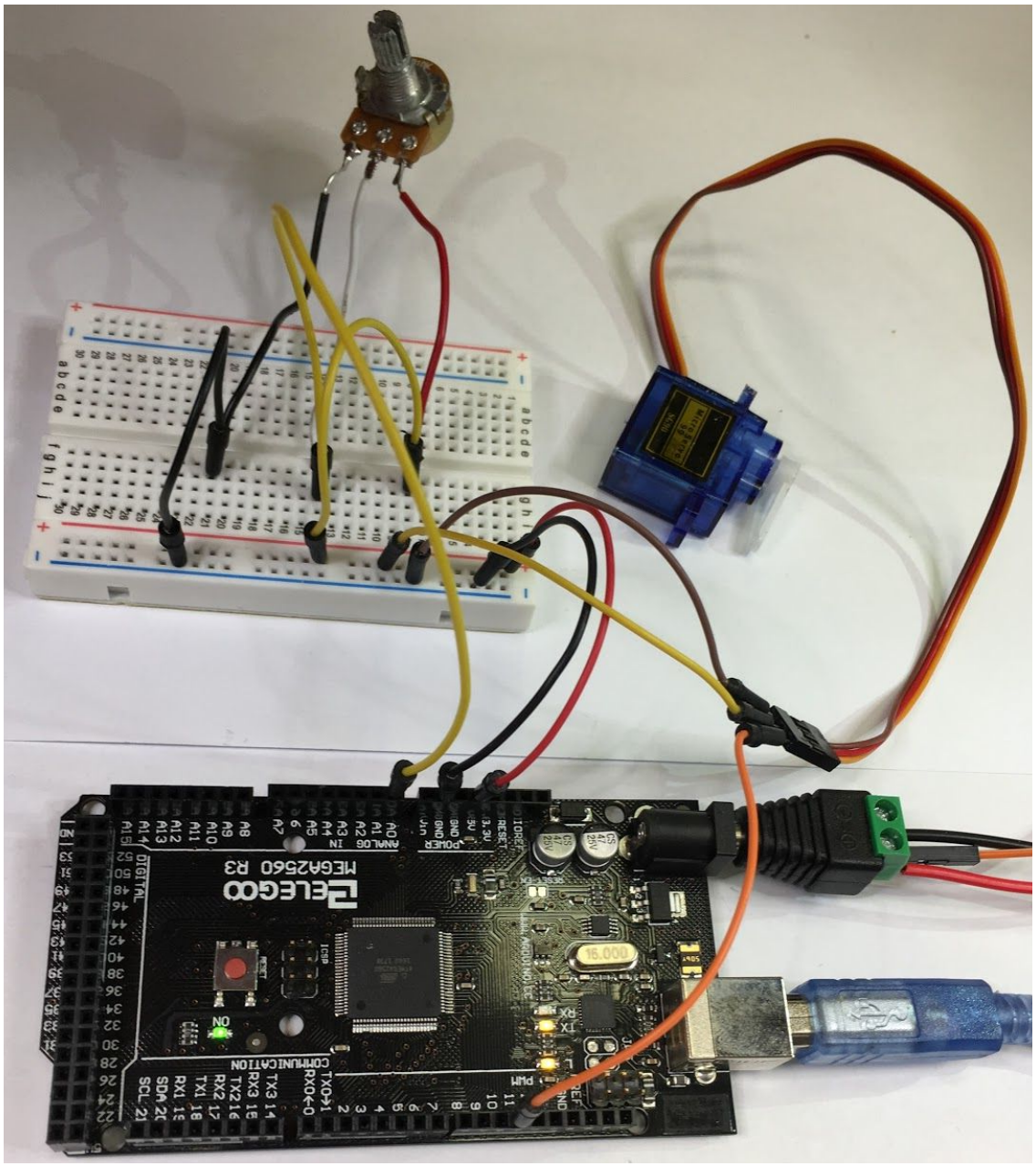
The servo motor has three wires: power, ground, and signal. The signal wire is connected to the Arduino.

In the Arduino sketch define two pins: one as analog input to read the potentiometer, the other as output to send pulses to the servo motor. Usually pins 10 or 11 are used as output for servos on the UNO since they support PWM. The width of the signal pulse determines the angle the servo will turn toward. The pulse should happen once every 20 ms. A 1 ms width pulse is 0 degrees, a 2 ms pulse is 180 degrees. Convert the reading on the potentiometer input (0 - 1023) to a pulse width of 1 to 2 ms.

In the void setup() function, define the pinmode for the output signal.

In the void loop(), define the start of a pulse where the signal pin is set high. Depending on the potentiometer value calculate how long the pulse will be in microseconds. After that many microseconds, set the signal pin low. Wait for the remaining part of 20 ms. Then repeat forever.

<FRIZING>



**Servo Code Answer Key:**

```
unsigned long deltaT;
int signalPin = 11;  //Change this number
int inputPin = 0;  //Change this number
int signalVal;
unsigned long microsCheckpoint;
bool cycleReset = 0;
```

```
void setup() {
      Serial.begin(9600);  //Initialize serial for debugging
      pinMode(signalPin, OUTPUT);
}

void loop() {
      if(cycleReset) {  //Start of pulse
            deltaT = 0;
            microsCheckpoint = micros();
            digitalWrite(signalPin, 1);
            cycleReset = 0;
      }

      signalVal = analogRead(inputPin) + 1000;  //Poll potentiometer
      deltaT = micros() - microsCheckpoint;  //Measure time since
                                                cycle start

      if(deltaT > signalVal) {  //End of pulse
            digitalWrite(signalPin, 0);
      }

      if(deltaT > 20000) {  //Wait to start next cycle and pulse
            cycleReset = 1;
      }
}
```