Lab4

Negin Kheirmand

9831023

This week's lab is about working with Servo Motor.

IC & Modules I used:

- Arduino Nano
- Servo Motor (sg90)
- 4*4 Matris Keypad
- a 100k potentiometer (& some jumper wires)

Codes for the Lab:

1. Part1 : sketch_90DegreeMadness directory
2. Part2 : sketch_ServoCommander directory
3. Part3 : sketch_ChangePosition directory
4. Part4 : sketch_potentiomotor directory

What I learnt in this experiments:

- **Servo motor has the spinning range of 0 to 180 degrees.** Meaning if it has to go counter-clockwise you first have to make sure it's at a position where the action can be carried out (not in 0 degree, since the servo cannot go back the starting point)
- **I can use the map function to map a range of integers to another range and get the equivalent of a value in the secondary range.**

Experiment Reports:

First experiment is a simple sketch to change the servo motor from 0 to 90 degrees, wait 1 second and go back to zero (and then repeat this all over again)


```
#include <Servo.h>


// Declare the Servo pin
int servoPin = 3;
// Create a servo object
Servo servo;

void setup() {
  Serial.begin(9600);
  // We need to attach the servo to the used pin number
  servo.attach(servoPin);
}

void loop() {
  // Make servo go to 0 degrees
  servo.write(0);
  Serial.println("0");
  delay(1000);
  // Make servo go to 90 degrees
  servo.write(90);
  Serial.println("90");
  delay(1000);

}
```

Pinout:

D3 of Arduino used to connect to data pin of servo.
The gnd and power pin of the servo goes to the gnd and +5v pins of the Arduino respectively.

This is going to be the pins I used to connect the servo to my Arduino Nano in all of the other sketches too.

Second experiment: In this sketch I get the spinning degree from the keypad and proceed to show it in the Servo (if possible).

In this Sketch the C Button of Keypad works as a delete key and

the D Button of the Keypad works as the command key

main part of the code:

```
    input = String(input + String(keyPressed));
  }
}else if( keyPressed =='C'){
  //is the delete key
  input = input.substring(0, input.length()-1);
}else if( keyPressed =='D'){
  //is the empty key
  input="0";
}else if( keyPressed =='B'){
  //is the "show me" key
  long a = atol(input.c_str());
  if(a<=180 && a >=0){
    servo.write(0);
    delay(2000);
    servo.writeMicroseconds(1000);
    servo.write(a);
    Serial.print("servo is showing the angle: " );
    Serial.println(a);
  }else if(a>180&& a<=360){
    servo.writeMicroseconds(1000);
    servo.write(180);
    delay(2000);
    servo.writeMicroseconds(2000);
    a = 360-a;
    servo.write(a);
    Serial.print("servo is showing the angle: " );
    Serial.println(input);
    Serial.print("a is  " );
    Serial.println(a);
  }else{
    Serial.print("servo cant show the angle: ");
    Serial.println(a);
  }
```

Pinout:

The same D3 to connect Arduino to Servo

And for the keypad:

Arduino	Keypad
6	1(R1)
5	2(R2)
4	3(R3)
2	4(R4)

Arduino	Keypad
7	5(C1)
8	6(C2)
9	7(C3)
10	8(C4)

Third experiment is a simple sketch that gets input from serial monitor and changes the angle of the servo motor according to the value of the input

Main logic of the code:

```
'
void loop() {
  if (Serial.available() > 0) {
    input = Serial.readString();
    Serial.println(input);
    int a = input.toInt();
    //if cant be casted to int the return value will be 0
    if( a!=0 ){
      int degreeNow = servo.read();
      if(degreeNow+a<=180 and degreeNow+a>=0){
        if(a>0){
          servo.writeMicroseconds(1000);
          servo.write(degreeNow+a);
        }else if(a<0){
          servo.writeMicroseconds(2000);
          servo.write(degreeNow+a);
        }
          Serial.println("degree now is ");
          Serial.println(degreeNow+a);
      }
    }
  }
}
```

Forth experiment was to change the angle of the servo motor using the potentiometer

sketch_potentiomotor

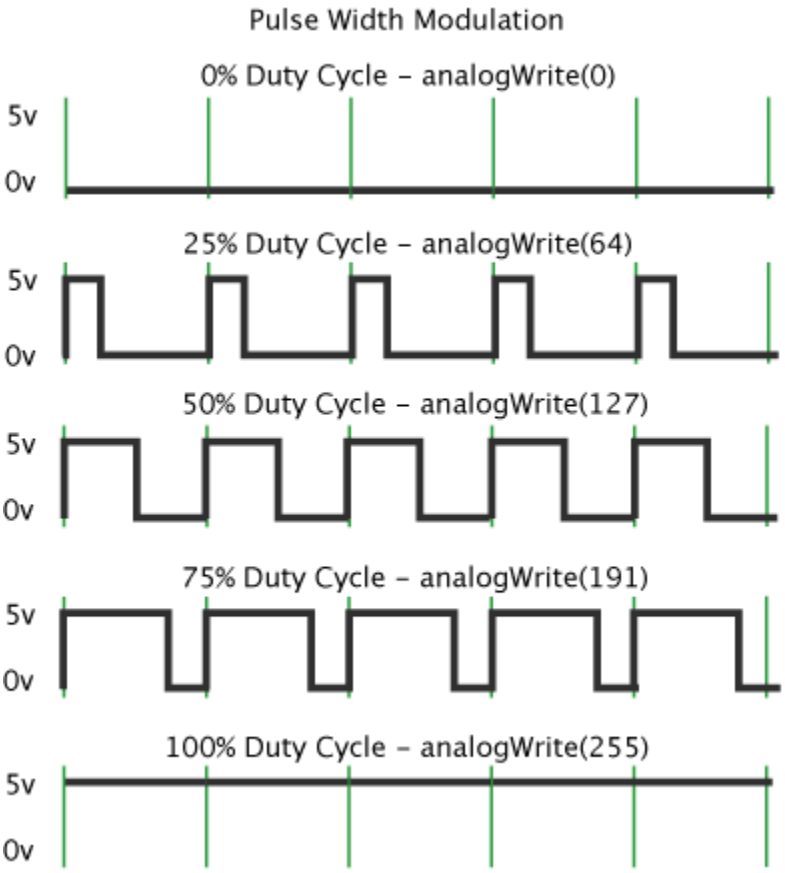
```
#include <Servo.h>

// Declare the Servo pin
int servoPin = 3;

int analogPin = 3;
// Create a servo object
Servo servo;

String input = "0";
int incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(9600);
  // We need to attach the servo to the used pin number
  servo.attach(servoPin);
  delay(1000);
}

void loop() {
  long value = analogRead(analogPin);
  value = map(value, 0,1023,0,180);
  servo.write(value);
  Serial.println(value);
}
```

Questions:

1. What is PWM:

Pulse Width Modulation, or PWM, **is a technique for getting analog results with digital means**. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on Uno, 3.3 V on a MKR board) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and Vcc controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



Something really interesting I found in the documentation while searching for this:

If you connect an led to a pin and use PWM you would not be able to tell the LED is turning on and off, since this happens really fast you would just think the LED was on all along, but how can we actually tell?

I quote:

“grab your arduino and shake it back and forth. What you are doing here is essentially mapping time across the space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.”

That is honestly so interesting!

2. What is Servo Motor used for and Where is it Used?

   Source: https://www.heason.com/news-media/technical-blog-archive/what-is-a-servo-motor-used-for-

Servo motors are very diverse and as well as being used throughout a wide range of industries for various applications they can also be used for more commonplace applications such as toys and home electronics.

Within specific industries, **servo motors are used to replace conventional AC motors, stepper motors and hydraulic and pneumatic systems**.

Servo Motor uses include:

- **Robotics** – Servo motors are lightweight and small which makes them ideal for use in this upcoming industry.
- **Simulation Applications** – another modern industry which involves the use of Servo motors. Their desired features for simulation applications include speed, torque and smoothness.
- **Defence** – within defence applications servo motors are exposed to harsh environments including high temperatures and high shock loads.
- Machine Tools – Machine tool applications rely heavily on the high accuracy servo motors provide.
- Printing Press – within printing press applications, servo motors are often selected for their high accuracy and speed.
- Conveyer and Handling Systems - Servo Motors are often selected for their high speed, high accuracy and torque.
- Food and Beverage – We have a range of servo motors which are suitable for use in washdown environments with constant temperature changes.
- Subsea and Oil & Gas - Servo motors are exposed to high pressure, high temperatures and may have explosion proof requirements for this environment.

3. Analog Signals and the analogRead() function:
   Source: https://www.allaboutcircuits.com/projects/using-the-arduinos-analog-io/
   The Arduino can input and output analog signals as well as digital signals.
   An analog signal is one that can take on any number of values, unlike a digital signal which has only two values: HIGH and LOW. To measure the value of analog signals, the Arduino has a built-in analog-to-digital converter (ADC). The ADC turns the analog voltage into a digital value. The function that you use to obtain the value of an analog signal is analogRead(pin). This function converts the value of the voltage on an analog input pin and returns a digital value from 0 to

1023, relative to the reference value. The default reference voltage is 5 V (for 5 V Arduino boards) or 3.3 V (for 3.3 V Arduino boards). It has one parameter which is the pin number. The Arduino does not have a built-in digital-to-analog converter (DAC), but it can pulse-width modulate (PWM) a digital signal to achieve some of the functions of an analog output. The function used to output a PWM signal is analogWrite(pin, value). pin is the pin number used for the PWM output. value is a number proportional to the duty cycle of the signal.

analogRead() function:

source :

analogRead() - Arduino Reference

http://spolearninglab.com/curriculum/lessonPlans/workshops/ttt/images/pcomp/servos/Arduino%20playground%20-%20MegaServo.html

description:
Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit. See the table below for the usable pins, operating voltage and maximum resolution for some Arduino boards.

4. Servo.h module functions:

- **Functions:**
  - attach()
    - Description: Attach the Servo variable to a pin. Note that in Arduino 0016 and earlier, the Servo library supports only servos on only two pins: 9 and 10
  - write()
    - Description: Writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. On a continuous rotation servo, this will set the speed of the servo (with 0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement).
  - read()
    - Description: Read the current angle of the servo (the value passed to the last call to write()).
  - writeMicroseconds()
    - Description: Writes a value in microseconds (uS) to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft. On standard servos a parameter value of 1000 is fully counter-clockwise, 2000 is fully clockwise, and 1500 is in the middle.
  - readMicroseconds()
    - returns angle (in uS), renamed from read_us