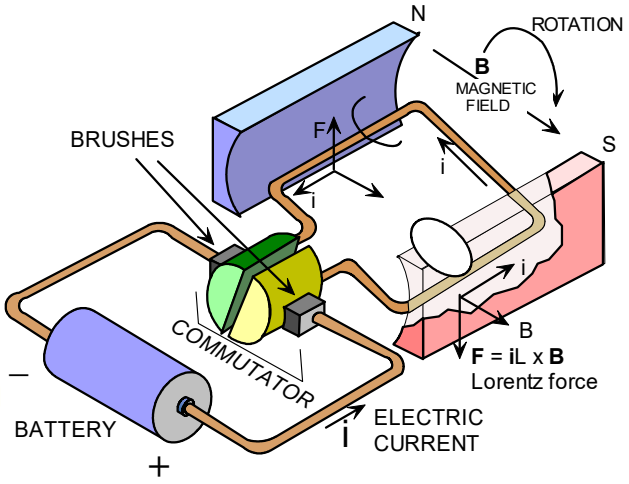
MECHATRONICS I

LABORATORY LECTURE & EXERCISES

LAB 4 - SERVO (H)



ITEM LIST

1.) Computer, Arduino, & USB cable
2.) Servo motor (e.g., Hitech HS-422, or similar)
3.) Hook up or jumper wire
4.) Optional: DC power supply (or 9V battery w/ snaps), voltage regulator

EXERCISE

Here we will use Arduino to control a small hobby servo motor.

BACKGROUND

REFER to CT Guide (Ch 16.1 - DC motors, 16.2 - Servo motors)

ABOUT DC MOTORS

DC motors can be found everywhere. They produce rotational motion when electricity is applied to their input pins. There are many different kinds of motors so we will focus on brushed DC motors. The simplest DC motor is pictured above right. The **STATOR** (the stationary part of the motor) has permanent magnets that produce a magnetic field. The **ROTOR** (or armature) is the rotating part of the motor. In this simple case, the armature is a simple coil of wire (sometimes called a loop or winding). The armature is connected to DC power through a **COMMUTATOR**. When electric current flows through the coil an electromagnetic **LORENTZ** force is induced in it by the magnetic field. This force causes the coil to rotate. The commutator serves to switch the direction of electric current so that the forces turn the armature in the same

direction regardless of its orientation. The commutator uses graphite brushes that ride along the contacts. Because of friction, these brushes eventually wear out.

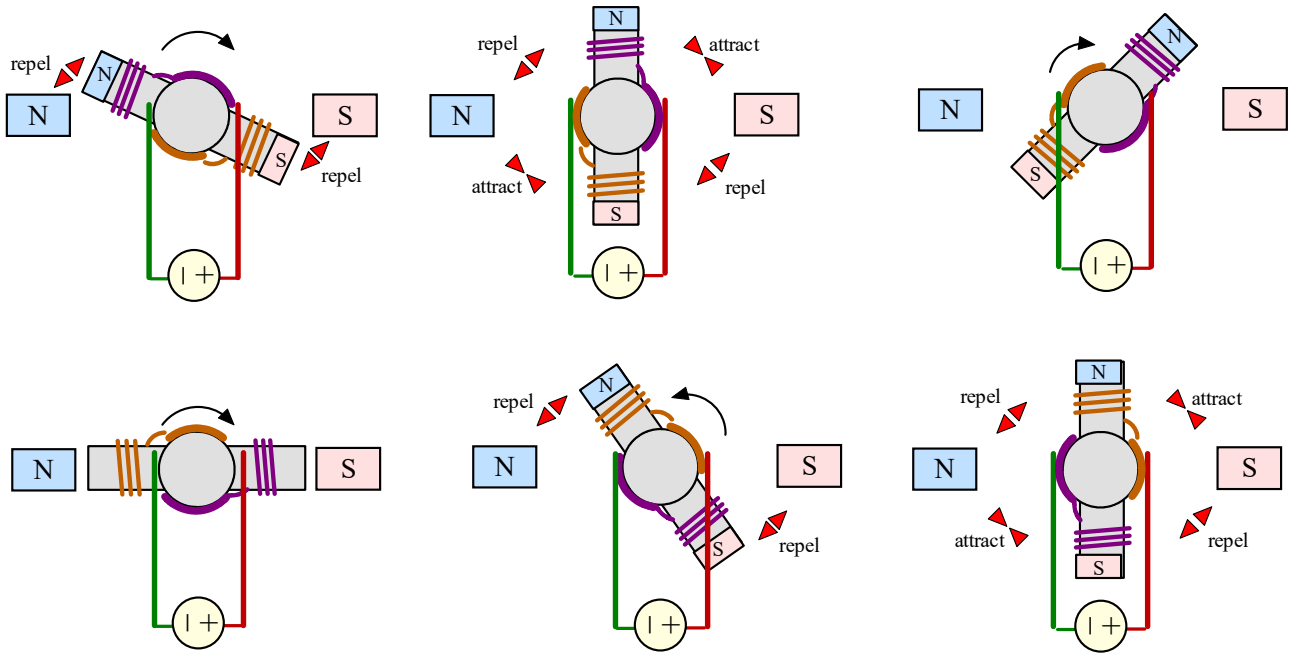
Here's a nice video that describes how DC motors work:
https://www.youtube.com/watch?v=LAtPHANEfQo

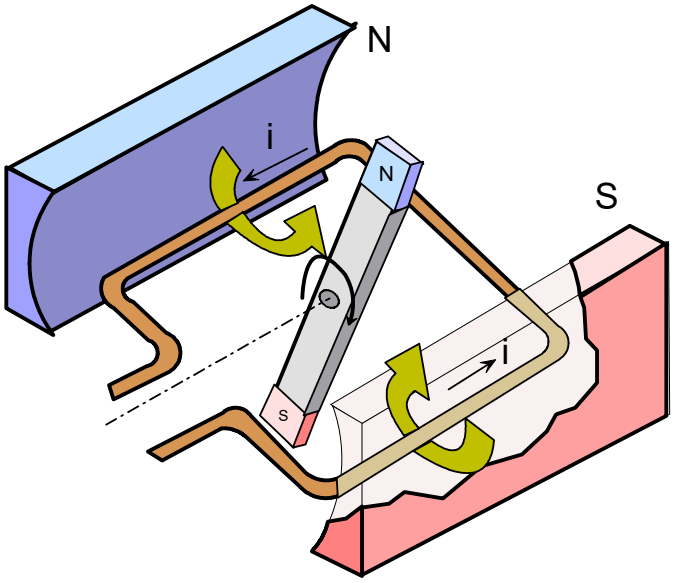
Another way to envision the motor's operation is in terms of the interaction of 2 magnetic fields - one produced by the stator permanent magnet and the "electro-magnet" produced in the armature. Magnets have a north (N) and south (S) pole. N repels N. S repels S. N attracts S. The electro-magnetic field produced in the armature can be thought of as a bar magnet whose N & S poles switch based on the rotor's angle via the commutator. The commutator switches rotor's magnetic poles in a way that keeps the rotor turning.

There's a nice video that helps illustrate this: (https://www.youtube.com/watch?v=wxG3cwugXgs).

An armature with just 1 loop of wire would only produce a very weak magnetic field. So in real motors the armature has many windings of wire wrapped around an iron core (or some ferro-magnetic material). The multiple windings and the iron core strengthens the magnetic field.

The result of this is that more electric current (or more voltage) results in a faster turning motor. In general, torque of a DC motor is proportional to the current flowing through its windings. In order to change the direction that the motor spins, the voltage "polarity" applied to the motor must be reversed.





DC HOBBY SERVO MOTORS

Standard hobby servo motors ("servos") contain a brushed DC motor, internal control circuitry, and often a "gear head". They are designed to rotate its shaft to a specific orientation (or "motor shaft angle") based on a pulse-width signal received. The internal control circuitry senses the actual shaft angle and corrects errors by driving the shaft in the opposite direction of the error. Servo motors generally rotate either 90° or 180°. Hobby servos are designed to move radio-controlled airplane flaps, boat rudders, and car steering among other applications.

Continuous rotation servos receive the same electronic signals, but instead turn at certain speeds and directions. Continuous rotation servos are handy for controlling wheels and pulleys.

Servos usually have 3 electrical connections: power, ground, and signal (where the pulsewidth control signal is sent). An important servo-motor spec is the supply voltage value (5 VDC is common for hobby servos). This spec means you must apply that much voltage for the motor's power. ONLY APPLY VOLTAGE TO THE MOTOR THAT IS WITHIN ITS SPECIFICATIONS! A higher voltage will blow out the motor! For instance if you apply 12V or more to a motor rated for 5-7V, you will likely destroy that motor.

Servos are controlled using PWM (pulse-width modulation) signals, which are digital pulses that are high (5 VDC) for a specific amount of time and lower otherwise (0 VDC). The pulses must occur at 50 Hz (cycles per second) for hobby servos.

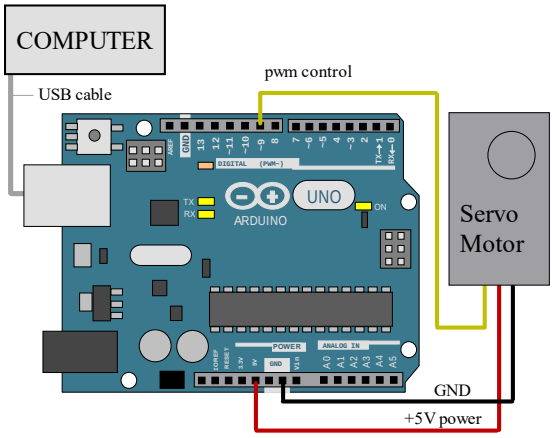
Arduino makes things easy by having a "servo" library that has many of the functions we need to drive a servo-motor. For instance, we can command the motor to a certain angle using the function servo.write (angle), where you specify the motor angle instead of the pulsewidth value. In addition, the functions in the servo library can run in the background while the CPU does other things. This is GOOD, and makes our programming easier! It does this by interrupting the execution of other code when it needs to send those pulses, and it does so very quickly so it is almost unnoticeable. This is an example of multi-tasking.

Servos draw varying amount of current depending on the "loading" conditions. A high load condition for instance would be when something is physically resisting the motor's rotation. A freely turning motor draws much less current. (e.g., the Hitech servo 8 mA at idle, 150 mA turning at no load, 800 mA at "stall"). Arduino may be able to provide sufficient current if the motor is allowed to turn freely. If the motor is being resisted, Arduino will not be able to provide enough current, and a separate power supply (e.g. – batteries or a DC power supply) must be used.

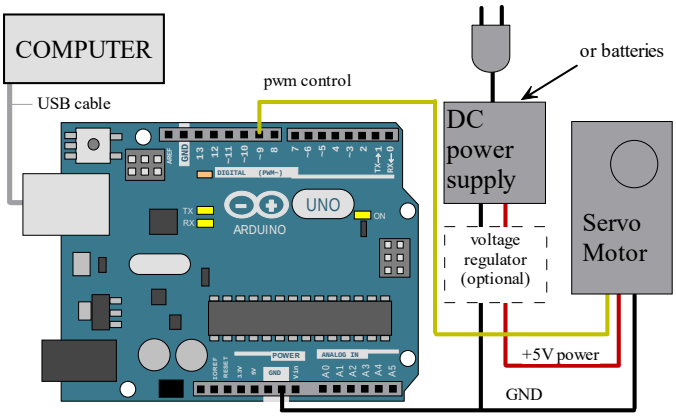
Generally it is better to use an alternate source of power for servos instead of relying on Arduino's power. DC voltage regulators are often used because they output a constant voltage signal. They generally take a higher DC voltage input (e.g. 9 VDC, from say a battery) and output a very consistent output voltage (e.g. 5 VDC) that is slightly lower than the input voltage. Often you can forego the voltage regulator and draw power from a battery pack. The motor will still work but it may not position as accurately. You can use alkaline batteries, but many RC control hobbyists use rechargeable NiCd, nickle-metal-hydride, or lithium polymer batteries.

4.1 SERVO TO POSITION

CIRCUIT & CONNECTIONS



POWER MOTOR WITH ARDUINO
(motor must NOT be loaded)



SET UP WITH DC POWER SUPPLY

Wire up according to the wiring diagram. We will work with the LEFT figure. The right figure is just for reference in case we ever decide to use a DC power supply and regulator (for now we will not). The control signal is shown connected to I/O pin 9. Another PWM pin may be used but the circuit but the program must match the circuit! The 3 connections to the servo are: power, GND, and signal. Be careful to connect this to the correct wire. Generally the wires are color coded, but the colors may differ on different motors. Generally the colors are:

Power	red, (usually it's the middle wire)
GND	black or brown
signal	yellow or orange (or possibly white)

USING POWER SUPPLY & REGULATOR (ignore for now)

It is usually suggested to use a separate power supply to drive the motor, and not use the Arduino. The Arduino may be sufficient for very small servo motors (micro-servo's) whose motion is not resisted. Arduino may not be able to drive a motor larger than a micro-servo. Options for powering the motor include power supplies, power bricks, or batteries. Regulated power supplies provide accurate consistent output voltages. Batteries and non-regulated power supplies don't produce highly consistent voltages. They can still work, but the motor may not perform as well. A downstream voltage regulator will help maintain a consistent voltage and improve servo performance. The input voltage to the voltage regulator must be a little

bit greater than the output voltage (in our case, output is 5 VDC). 5V power to the Arduino should be separate from the 5VDC output of the voltage regulator that is meant to power the motor. However, all ground wires should be connected together. For instance, the (-) side of the power supply (or battery) should be wired to Arduino GND.

CODE

We must include the servo.h file which has libraries of functions for driving a servo motor.

```
// servo1_position.ino --------------------
// drive a servo motor to an angular position


#include <Servo.h>
Servo myservo;

int servoPin = 9;          //this # must match the circuit & pin must be pwm
int servoAngle = 90;

void setup()
{
  myservo.attach(servoPin);
  //myservo.writeMicroseconds(900);
  myservo.write(servoAngle);
}

void loop()
{
}
```

HOW IT WORKS

The servo.write ( ) function takes an angle argument so, servo.write (angle), where "angle" is an INTEGER from 0 to 180 (in degrees). Pulses must occur at 50 Hz but the servo.write function handles this for us.

It should be noted there is another way of commanding the motor using servo.writeMicroseconds (uS), where "uS" is the time of the pulsewidth in microseconds. You must look up the spec sheet of your motor to find out what angle is associated with what pulsewidth value.

A MICROSECOND (μs) is one-millionth of a second (1 million us = 1 s). A MILLISECOND (ms) is one-thousandth of a second. So 1000 μs = 1 ms.

OUTCOME & BEHAVIOR

The servo motor should move to the angular position programmed. Try changing the servo angle to another value between 0 and 180 degrees. Then upload and run and observe that the motor "servos" to the programmed angular position.

4.2 SERVO MOVING IN STEPS

CIRCUIT & CONNECTIONS (same as before)

CODE

We must include the servo.h file which has libraries of functions for driving a servo motor.

```
// servo2_steps.ino --------------------
// Servo motor should move from 0 to 180 degrees, in 20 deg steps
// it then moves back to 0 deg and start over


#include <Servo.h>
Servo myservo;

int servoPin = 9;          //this # must match the circuit & pin must be pwm

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

void loop()
{
  //turn servo from 0 to 180 in 20 deg increments
  //when done it automatically resets to 0 deg
  for (int i = 0; i <= 180 ; i = i + 20)
  {
        myservo.write(i);       //i = angle
        delay(200);
  }
}
```

HOW IT WORKS

The "**for**" loop causes a portion of the code to repeat a certain number of times. It starts with index variable i = 0, runs the code in the braces ({ }). Then it adds 20 to i (**i = i + 20**), and then repeats the code with the updated i. Then i is incremented again (to 40) and the code repeats. This process repeats until i reaches 180, and then the **for** loop is completed. However, the end of the for loop also marks the end of the **loop ( )** function, which itself repeats. So the for loop starts again with i = 0. And the process repeats forever.

OUTCOME & BEHAVIOR

The servo motor moves from 0 to 180 degrees in 20 degree steps. When complete, the servo sweeps all the way back to 0 degrees and repeats the behavior.

4.3 SERVO STEP OSCILLATION

Here we will alter the behavior again. This time the motor will oscillate back and forth in steps.

CODE

```
// servo3_stepOscill.ino --------------------
// Servo oscillates between 0 & 180 deg, in 20 deg increments (back and forth)


#include <Servo.h>
```

```
Servo myservo;

int servoPin = 9;

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

void loop()
{
  //turn servo from 0 to 180 in 20 deg increments
  for (int i = 0; i <= 180 ; i=i+20)
  {
        myservo.write(i);
        delay(80);
  }

  //turn servo from 180 to 0 in 20 deg increments
  for (int i = 180; i >= 0 ; i=i-20)
  {
        myservo.write(i);
        delay(80);
  }
}
```

OUTCOME & BEHAVIOR

The servo motor moves from 0 to 180 degrees in 20 degree steps. Then it reverses, going from 180 to 0 degrees in 20 degree steps. This repeats over and over.

4.4 OSCILLATING WITH SINE WAVE

Here we can oscillate the motor using the sin( ) wave function on Arduino.
We need to access the time using the millis ( ) function.

CODE

```
// servo4_sin.ino --------------------
// servo oscillates using sin() function

#include <Servo.h>
Servo myservo;

int servoPin = 9;
long ti;
float servoAngle;
float freq = 0.5;  //freq is in Hz or cyc/sec

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

void loop()
{
  ti = millis();
  servoAngle = 90 + 80*sin(2 * PI * freq * ti/1000);
  myservo.write(servoAngle);
```

```
}
```

OUTCOME & BEHAVIOR

The servo motor should oscillate back following a sine wave. The movement should be smoother than in the last exercise. This program should be easier to adjust the frequency of the oscillation. Adjust the f value and observe the changes in motor behavior.

Hopefully you can see how easy it is to change the motor behavior with simple changes in code and NO changes to circuitry, motor, or connections.

LAB SIGN-OFF FORM
(for students to print out)

Student Name: _____

Student Name: _____

1. _____	Servo to position	servo moves to specified angle
2. _____	Servo moving in steps	turn servo 0-180 in 20 degree steps
3. _____	Servo step oscillation	oscillate 0 to 180, then 180 to 0 in 20 degree steps
4. _____	Servo oscillating w/ sin( )	oscillation use sin() function

SUBMISSION:

Students will demonstrate the working exercises to the instructor in class.

A possible option is uploading videos to Canvas.
The instructor will let you know which is acceptable.
See the rules for file uploads to Canvas.