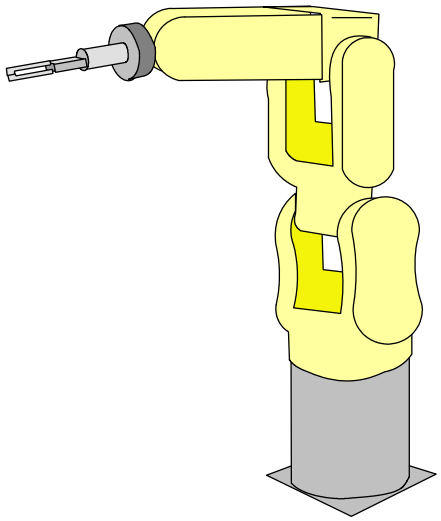
LAB 8 - FANUC ROBOT 2



ITEMS NEEDED

1. Fanuc Robot
2. Robot software
3. Fanuc Robot Handout (50 student-hours per Joe Ray, or 10 wks)

INTRO

Robots are often programmed to perform a set of movements (assembly, welding, packaging, etc.). Here students will learn to program the Fanuc to perform a simple sequence of movements. Movements are generally programmed by jogging the robot to a position and "teaching" that point. Then the robot is programmed through a sequence of these taught points.

Finally, students will program the robot program to LOOP through a set of code.

REFER to the FANUC HANDOUT by CT.

8.1 Exercises

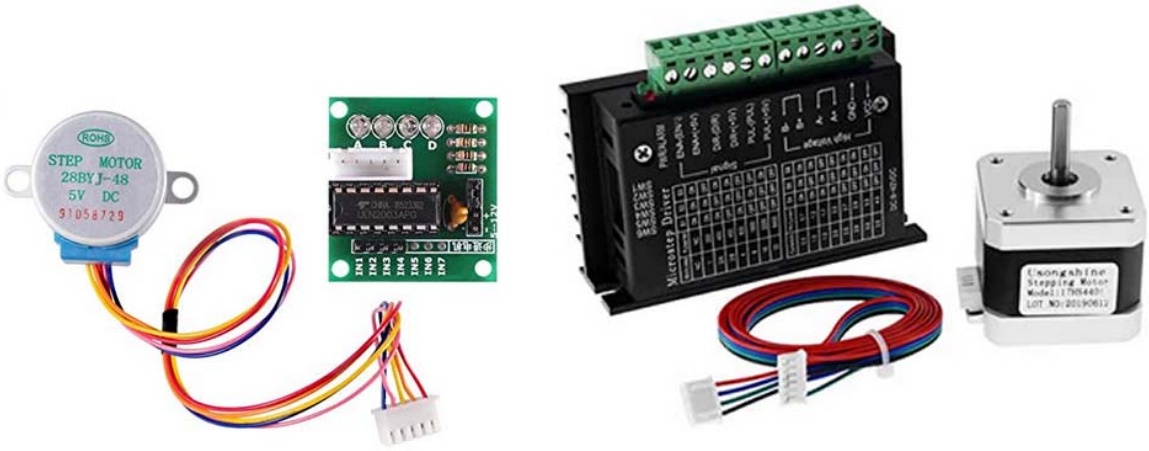
Students will perform a set of BASIC tasks as outlined below.

1. Create a program
2. Rectangle movement - program the robot to move in a rectangle
3. Loops - get the robot to repeat the movement
4. Save and open the program

OTHER POSSIBLE LABS

LAB 9 - POSSIBLE LABS ----------

LAB 10 - STEPPER MOTORS



ITEMS NEEDED

1. Computer (with Arduino IDE), Arduino, USB cable
2. Stepper motor (28 BYJ 48, unipolar), stepper driver/controller (ULN 2003)
   - specs: 32 steps/rev (but 2048 steps/rev with gearing), 5V,
3. Power supply, 5 VDC (adapter or bench power supply)
4. Connections, cables, including at least 6 male-female dupont cable wires

INTRO

A stepper motor is a type of brushless DC motor that moves in discrete steps. Here we will learn how to control the movement of an inexpensive stepper motor using Arduino.

Watch the 2 videos below that introduce stepper motors:
Learning Engineering
https://www.youtube.com/watch?v=eyqwLiowZiU
DroneBot Workshop
https://www.youtube.com/watch?v=0qwrnUeSpYQ
(1:44 – 3:36; 4:39 – 8:14; 8:50 – 11:30; exercise: 15:20 – 28:15; bipolar 29:20 – 36:40)

A stepper motor is a type of brushless DC motor that moves in discrete steps. The motor is then able to hold its position at any of these steps without a position sensor for feedback (an example of an open-loop control), as long as the motor has sufficient torque and speed for the application. Thus steppers provide precise positioning without feedback control.

Stepper motors are widely used in industrial and commercial applications. There are commonly used in machines like 3D printers, laser engravers, some CNC machines, disk players, and cameras. They are known for their low cost, high reliability, high torque, and low speeds.

HOW STEPPER MOTORS WORK

Stepper motors have a stationary stator and a rotating rotor. The rotor is a magnetized and has a number of rotor teeth. The stator surrounding the rotor has a number of electro-magnetic stator teeth (they have coils of wire). Magnetic fields are produced by passing electricity through the coils. The number of stator teeth and rotor teeth are designed so that only one set of teeth are in alignment at a given rotational position. Stepping occurs by energizing a set of stator coils with a PULSE of electricity (usually a square wave), producing an electro-magnetic field which attract the rotor teeth to pull them in alignment with the energized stator teeth. Each pulse results in a stepped movement of a particular angle (determined by the number of teeth). Pulses are sent in a specific sequence in the stator coils to produce desired movement. When pulses are sent at higher frequency, they produce a "continuous" rotation whose speed is proportional to the frequency of the pulses.

The sequencing of the pulses is achieved through an external driver circuit or a micro controller.

The circular arrangement of electromagnets is divided into groups, each group called a phase, and there are an equal number of electromagnets per group. The number of groups is chosen by the designer of the stepper motor. The electromagnets of each group are interleaved with the electromagnets of other groups to form a uniform pattern of arrangement. For example, if the stepper motor has two groups identified as A or B, and ten electromagnets in total, then the grouping pattern would be ABABABABAB.

There are 3 types of stepper motors: variable reluctance, permanent magnet, and hybrid.

There are two different coil wiring arrangements – bipolar and unipolar.

Bipolar has 2 coils, and 2 wires for each coil, and therefore have 4 total connections.
- needs voltage reversal (so a more advanced controller)
- higher torque
- slower

Unipolar has 2 coils, but each coil has a center tap, so they can have 6 connections, but often have 5 since often the center taps are tied together.
- no voltage reversal
- lower torque
- faster (but still slow)
- simpler controller

Stepper motor specifications include: voltage rating, current rating, step angle, inductance (higher = slower speed), holding torque (when energized), detent torque (when not energized), size (NEMA, or National Electrical Manufacturers Assoc).

SPECS

Basic stepper motor specifications are described below

1. Step angle (deg, or steps/revolution)
2. Voltage rating
3. Current – required,
4. Coil resistance

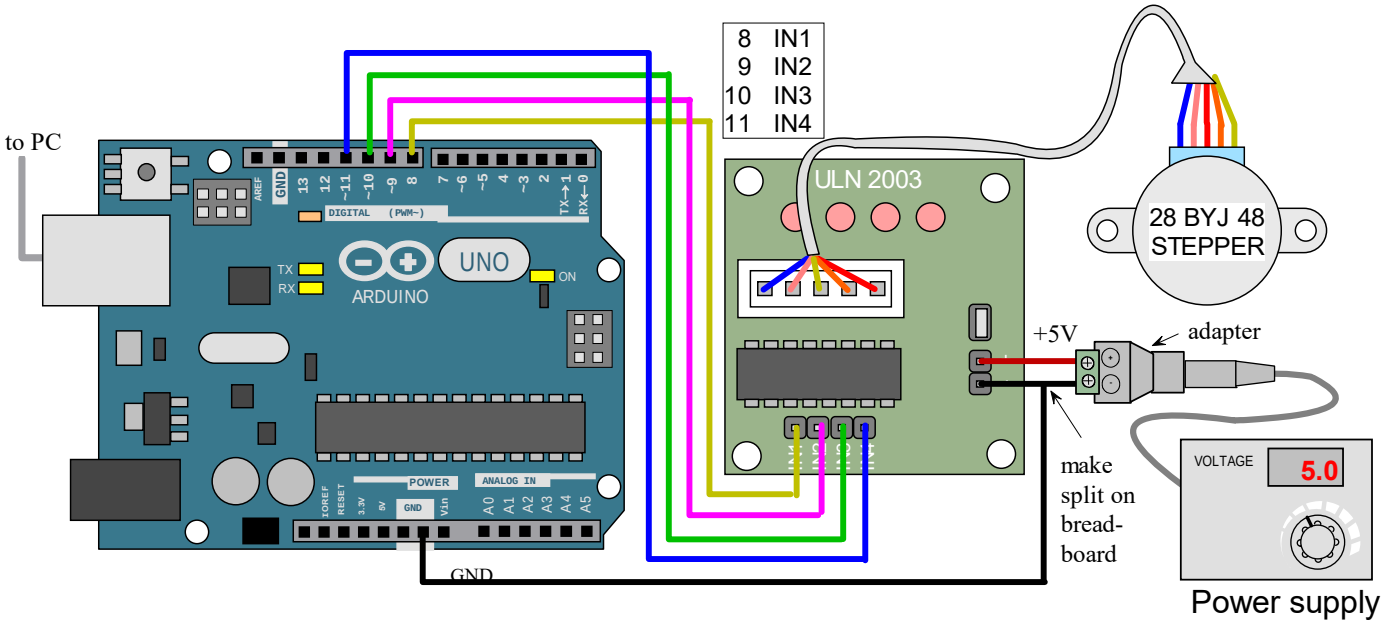
5. Inductance – higher means slower
6. Holding torque – torque it can exert when energized
7. Detent (or residual) torque – torque it exerts when not energized
8. Shaft styles – round, D shaft, geared, lead-screw shaft,
9. Motor size - NEMA (National Electrical Manufacturers Assoc), e.g., NEMA 17 (plate size 1.7 inches)

10.1 BASIC STEPPER CONTROL

We will control a stepper motor using a standard Arduino stepper library.

CONNECTIONS

See figure. The stepper motor takes 5VDC. NEVER apply more than ~5.3V to the motor - even for a moment or the motor could fry. Use the adjustable power supply and set to 5V (+/- 0.1V). The voltage is adjusted with a dial. Detach +V wire from the motor. Then turn on the supply by turning the dial (there's a "click"). Only turn the dial a small amount and adjust so output voltage is 5 +/- 0.1V (so 4.9 - 5.1V). Then re-attach +V wire. Again - do not - even for a moment - apply more than ~5.3V to the motor. The output of the power supply is a co-ax plug that requires a "break out" adapter so power and ground can be wired to a breadboard and the motor controller (note Arduino's power jack is 2.1 mm inner, 5.5 mm outer). Ground of the power supply, the controller, and Arduino must be connected together. Use the breadboard for the split.




```
// stepper_1.ino --------------------
// Drive stepper motor with Arduino

#include <Stepper.h>      // include stepper library

const float STEPS_PER_REV = 32;        // stepper's steps/rev
const float GEAR_RED = 64;             // stepper's gear ratio (may be diff)
const float STEPS_PER_OUT_REV = STEPS_PER_REV * GEAR_RED;   // steps/geared revol

int StepsRequired;

//Arduino pins      8,   9,    10,  11
```

```
//Motor driver pins  In1, In 2, In3, In4
//Sequence is        1    3      2   4

// argument order is critical
Stepper steppermotor(STEPS_PER_REV, 8, 10, 9, 11);

void setup()
{
}

void loop()
{
  //Slow – 4 step CW seq to observe LEDs sequence on driver board
  steppermotor.setSpeed(2);
  StepsRequired = 8;
  steppermotor.step(StepsRequired);
  delay(2000);

  // rotate CW 1/2 turn slowly
  StepsRequired = STEPS_PER_OUT_REV/2;
  steppermotor.setSpeed(200);
  steppermotor.step(StepsRequired);
  delay(1000);

  // rotate CCW 1/2 turn quickly
  StepsRequired = - STEPS_PER_OUT_REV/2;
  steppermotor.setSpeed(700);                    //near top range
  steppermotor.step(StepsRequired);
  delay(2000);
}

// end ----------------
```

Once you have control of the motor you can do more interesting things with it.
Alter the code a bit in order to get the servo to oscillate back and forth.

10.2 STEPPER USING ACCEL LIBRARY

Now you will control a stepper motor using the "AccelStepper" library which has more advanced functions for controlling: maximum speed, set speed, acceleration, and moveTo. This library is not part of the regular Arduino IDE software, so you must add it.

CONNECTIONS (same as before)

CODE

```
// stepper_2.ino --------------------
// Drive stepper motor with Arduino

#include <AccelStepper.h>       // include – must install

#define FULLSTEP 4
#define HALFSTEP 8
```

```
#define motorPin1 8       //driver 1 pin 1
#define motorPin2 9       //driver 1 pin 2
#define motorPin3 10      //driver 1 pin 3
#define motorPin4 11      //driver 1 pin 4

//#define motorPin4 4           //driver 1 pin 1
//#define motorPin5 5           //driver 1 pin 2
//#define motorPin6 6           //driver 1 pin 3
//#define motorPin7 7           //driver 1 pin 4

AccelStepper stepper1(FULLSTEP, motorPin1, motorPin3, motorPin2, motorPin4);
//AccelStepper stepper2(HALFSTEP, motorPin5, motorPin7, motorPin6, motorPin8);


void setup()
{
  //1 rev motor 1 CW
  stepper1.setMaxSpeed(1000.0);       //1000 is about motor's max speed
  stepper1.setAcceleration(50.0);
  stepper1.setSpeed(200);
  stepper1.moveTo(2048);

  //1 rev motor 2 CCW
  //stepper1.setMaxSpeed(1000.0);
  //stepper1.setAcceleration(50.0);
  //stepper1.setSpeed(200);
  //stepper1.moveTo(-2048);
}

void loop()
{
  //change dir at the limits
  if (stepper1.distanceToGo() == 0)
  stepper1.moveTo(-stepper1.currentPosition());
  //if (stepper2.distanceToGo() == 0)
  //stepper2.moveTo(-stepper2.currentPosition());

  stepper1.run();
  //stepper2.run();
}

//end -------------------------
```

10.3 STEPPER SINUSOID (your turn)

Once you have control of the motor you can do more interesting things with it. Start with the code from the first exercise (that uses the basic stepper library). Then do as Save As. Then alter the code so that it follows a sinusoid (use the function sin()) of a given frequency (say 0.3 Hz or cycles/second). Maximize the set speed to 700. You will need to completely replace the code in the loop ( ) function. Hint - you set the variable "StepsRequired" according to a sine wave.

LAB 11 - BRUSHLESS MOTORS

ITEMS NEEDED

1. Brushless motor (e.g., 1000 kV, 2S-4S Lipo)
2. ESC (30A?)
3. Arduino, cable, pc,
4. Power supply or LiPo battery
5. Wire
6. pot

INTRO

Here we will control the speed of a brushless DC (BLDC) motor using Arduino and a potentiometer.

Recall that brushed DC motors have a commutator which serves to switch the direction of the current in the motor's windings so that the motor always turns in one direction. Unfortunately, the rubbing of the brushes on the turning commutator contacts causes friction, sparks, and electrical noise. Friction eventually causes the commutator brushes to wear out, and they must be replaced. Thus brushed DC motors are not a good choice if a motor is turning continuously (e.g. – a fan or propeller on a plane).

Brushless DC motors (BLDCs) solve this problem by not having a commutator. Electrical switching is done electronically in the motor controller (they use transistors which are GREAT AT SWITCHING). However, the controller must do the switching at precise motor angle positions. Motor angular position is determined with a sensor (typically a Hall Effect sensor) or by using the motor's back EMF. All induction motors generate back EMF – an opposing voltage – when they turn. The back EMF signal is essentially a way of determining the motor's shaft angle.

Brushless motors are handy for things that turn continuously, like wheels, propellers, and fans. Sometimes AC motors are used for such applications (e.g. – fans), but it is more difficult to control the rotating speed of AC motors. Thus, you may find BLDC motors used in continuous turn applications where speed control is critical, such as with drones.

BLDC & ESC's

Brushless DC motors still work on induction principles, just like a normal DC motor. This means that the motor movement is driven by the interaction of 2 magnetic fields, often one from a permanent magnet and the other from an electro-magnet (magnetism produced when electricity flows through coiled wires). It is also possible to use 2 electro-magnets and no permanent magnets. Many BLDC's are designed to work in drones or to drive propellers. They are often designed with permanent magnets mounted to the rotor (the rotating part and the electro-magnet coils in the stator. Note this is the opposite of DC servo motors, where the stator holds the permanent magnets and the rotor contains the coils. In addition, "outrunner" BLDC's have the rotor on the outside and the stator on the inside. BLDC's have several sets of coils. As the coil is energized, it produces a magnetic field that interacts with the magnetic field of the permanent magnets in the rotor. This will cause the rotor to rotate. As the rotor rotates, another set of coils is energized causing the rotor to continue to rotate. The proper sequence of coil energizing will cause the BLDC to spin. In a BLDC with 3 coil sets, typically 2 coil sets are energized at any one time.

A BLDC requires an electronic speed controller (or ESC) in order to operate. The ESC is responsible for sending electricity to the proper motor coil at the proper time. The ESC must energize the appropriate set of coils based on its angular position. Motor position sensing can be done with either a Hall sensor or through the use of the motor's "back EMF". In the case of BLDC, there is always one set of coils that is not energized. However as that coil moves through the magnetic field of the permanent magnet, a voltage is induced in it which is called "back EMF". The ESC can infer the motor's angular position by sensing the voltage in the de-energized coil. Therefore, an ESC – at any given moment – is usually driving 2 coils and using the 3rd coil to determine motor shaft angle.

BLDC's, ESC, and LiPo Ratings

We will work with ESC's and BLDC's designed for hobby use (in RC drones, planes, etc.). Such BLDC's generally have a so-called "KV" rating. ESC's have a current rating and can be pretty high (e.g., 30A. BLDC's can draw quite a lot of current!). Hobby ESC's will often have a "battery eliminator circuit", a small plug that not only takes the control signal from Arduino, but also routes 5V power from the battery to power the Arduino (so no extra battery is needed to power Arduino). Lithium-polymer (LiPo) batteries are often used to power the ESC's and they have ratings such as 2S, 3S or 4S for example. Each "S" represents how many cells the battery has. LiPo batteries provide 3.7V per cell. So a 2S battery is 7.4V (2x 3.7V) and a 3S battery is 11.1V (3x 3.7V). The KV rating on the BLDC is a way to measure how fast it can spin. For instance a 1000 KV BLDC can turn at 7400 RPM if powered with a 2S battery (see equation below).

RPM = KV x volts
RPM = 1000 x 7.4 = 7400 rpm

BLDC SPEED CONTROL

The control signal for our BLDC's is a 50 Hz PWM signal ranging from 1000 us (min speed) to 2000 us (max speed). Note "us" = micro-second, or one-millionth of a second. This PWM range is similar to that for servo motors. Hobby type BLDC's and ESC's (like the kind we are using) have internal electronics that are set up to be interfaced through radio control and incorporate certain safety protocols. For instance, many hobby ESC's must be "armed" before driving the motor (a safety procedure). During the arming procedure, the control signal must be equal to or less than the minimum value of 1000 us (or 1 ms) during an initial time period (usually 3-5 seconds). Once done, the motor sounds some confirmation beeps. This arming procedure must be part of your Arduino program. There is also a calibration procedure that allows you to tune the min and max pulsewidth values to the specific hardware.

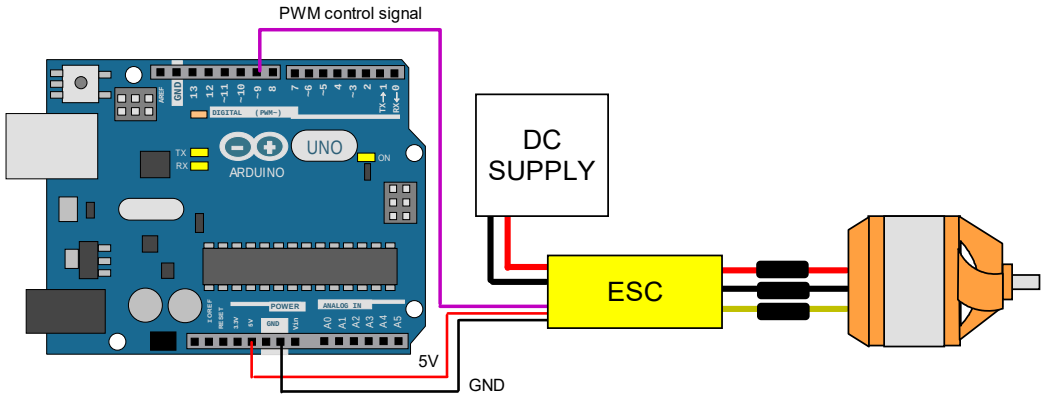
Please watch this helpful video (How To Mechatronics):

https://www.youtube.com/watch?v=uOQk8SJso6Q

11.1 BLDC CONSTANT SPEED

Here we will command a BLDC motor to turn at constant speed. The ESC uses the PWM signal from Arduino to determine how much electrical power to route from the DC power supply to the BLDC motor. This, in turn, controls the motor turning speed.

CONNECTIONS



Make the connections as shown in the figure. Set the DC supply to 7-11 volts, 5 A (if your supply can control current).

CODE

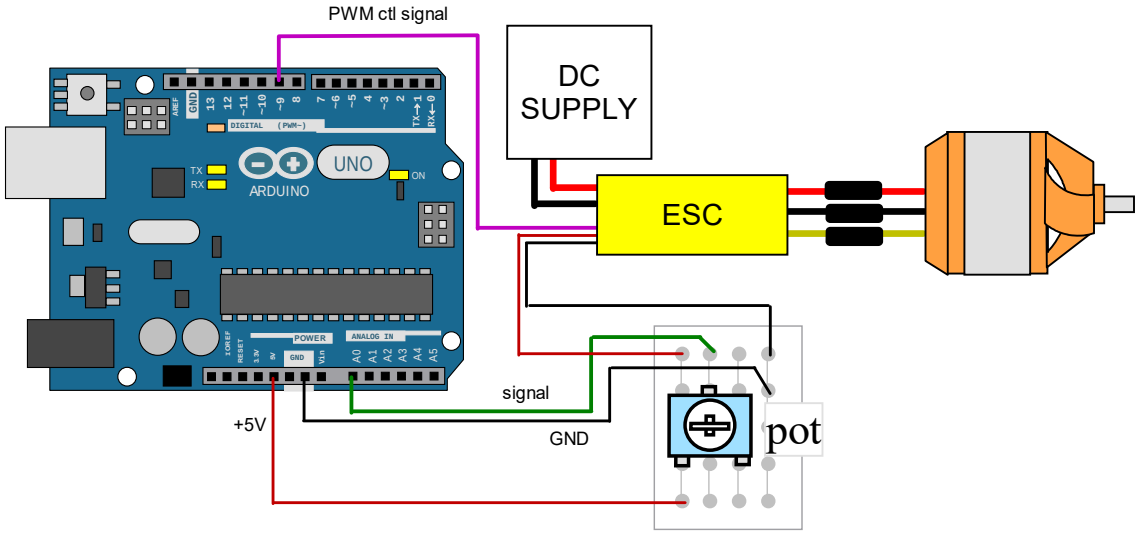
```
// bldc1_constSpeed.ino -------------------------------------
// control the speed of a BLDC motor using Arduino
// submit video: BLDC1_constSpeed.mp4


#include <Servo.h>

Servo bldc;

void setup()
{
  bldc.attach(9,1000, 2000);          //bldc signal pin (8 is not pwm)
  bldc.write(0);                      // arm the ESC
  delay(1000);                        // 2000 to 3000 ?
  Serial.begin(9600);
}
void loop()
{
  bldc.write(50);               //map to 0 - 180 (0 = 1000 us, 180 = 2000 us)
}
```

11.2 BLDC SPEED & POT

Here we will control the speed of a BLDC motor by turning a potentiometer. The pot angle is measured on an analog input pin.

CONNECTIONS

Add the potentiometer connections as shown. Use a breadboard to split the 5V and GND signals.



CODE

```
// bldc2_pot.ino ---------------------------------------
// Control the speed of a BLDC motor using Arduino & pot
// submit video: BLDC2_pot.mp4


#include <Servo.h>

Servo bldc;

void setup()
{
  bldc.attach(9,1000, 2000);          //bldc signal pin (8 is not pwm)
  bldc.write(0);                      //
  delay(1000);
  Serial.begin(9600);
}

void loop()
{
  int potval;                    // pot value
  potval = analogRead(A0);       //wiper of pot connected to analog in A0
  potval = map(povval, 0, 1023, 0, 180);    // 0 - 180 (0 = 1000 us, 180 = 2000 us)
  bldc.write(vin);
}
```

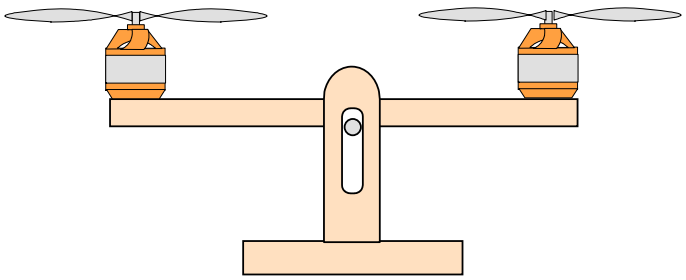
11.3 BLDC SINE

Here we will have the BLDC speed oscillate with a sine wave instead of using a potentiometer. Remove the pot and simply oscillate the speed of the motor with an Arduino program. Have the motor oscillate between max and min speed with a sine wave of a relatively low frequency. The connections are unchanged.

CODE

```
// bldc3_sine.ino ------------------------------------
// oscillate the speed of a BLDC per a sine wave
// submit video: BLDC3_sine.mp4

#include <Servo.h>

Servo bldc;
int motorPin = 9;    // pwm pin

float ampl = 20.0;     //amplitude of sin wave, half of 180
float f = 0.3;        //freq in Hz (1/period T)


void setup()
{
  bldc.attach(motorPin, 1000, 2000);  //bldc signal pin (8 is not pwm)
  bldc.write(0);                      // arm the ESC
  delay(1000);                        // 2000 to 3000 ?
  Serial.begin(9600);
}

void loop() // -------------------------------------------------
{
  float tnow;
  float motorSpeed;
  float sig;

  tnow = millis() / 1000.0;

  sig = ampl * sin(2 * PI * f * tnow) + ampl;       // 0 to 180
  motorSpeed = (int) round(sig);
  bldc.write(motorSpeed);
}

//end ----------------------------------------------------------------
```

11.4 BLDC BALANCER



ITEM LIST

1. (2) Brushless motors
2. (2) Propellers + mounting brackets

3. (2) ESC's (electronic speed controllers)
4. (1) Arduino, cable
5. (1) Power supply (LiPo battery or DC supply)
6. (1) Accelerometer/gyro sensor
7. (1) See-saw hardware (made on laser)
8. Wire, connectors,

INTRODUCTION

Here we will get a see-saw arm to balance under the influence of 2 brushless motors mounted on either end of the arm. The motors will be connected to propellers. Electronic speed controllers (ESC's) will control the motors, and an Arduino micro-controller will control the ESCs. An accelerometer sensor mounted to the arm will sense the arm's angle. Arduino will control the speed of the propellers based on the accelerometer sensor feedback. The controller will attempt to keep the arm horizontal. Another OPTION for this exercise is to have only 1 BLDC. We will use a PID controller. Due to the constant force of gravity on the system it is likely we will use a PI or PID controller.

REFER to the CT GUIDE on PID controllers.

This system is a simplified version of a flight controller for a quad-copter drone.

STEPS

Mount the accelerometer to the arm at the pivot point. If mounted there, then accelerations in the arm's direction should be zero. The accelerometer mounted to the end can also work but is more complicated as you must consider motino (kinematics) effects there. For instance, the accelerometer will move in circular motion as the arm rotates. This causes tangential and centripetal accelerations that must be accounted for.

Start with P control only first. Adjust the proportional gain (Kp) until you get the arm to oscillate a bit around the horizontal position. Add D control to reduce oscillations. The system will likely continue to SAG due to the constant effect of gravity. Now add I control to correct for that. Be careful with I control. It is easy to create instability with I control (eg - integral windup). You may have to implement clamping or other techniques when using I control.

CODE

```
// bldc4_balancer.ino ------------------------------------
// control the angle of a pivot arm with BLDC's mounted to it
// this is a test fixture for drone control

#include <Servo.h>

Servo bldc;
int motorPin = 9;    // pwm pin

// YOUR TURN
// collect sensor data
// incorporate PID controller
```

LAB 12 - PNEUMATICS CONTROL

ITEMS NEEDED

1. Arduino, cable, computer with Arduino IDE
2. Pneumatic air cylinder
3. Servo and or solenoid valves
4. Air lines, fittings, compressor.
5. Power supply for servo/solenoid vales

INTRO/BACKGROUND

Pneumatics uses air pressure to achieve motion. Air cylinders are typical pneumatic actuators. They are usually cylindrical in shape (circular cross section). Air cylinders have a piston and shaft. Pressure differences across the piston cause the piston and translate in or out.

Air pressure originates from air compressors that reside either in the lab or on the roof of the building. Air pressure from centralized air compressors is referred to as "shop air". Shop air provides pressure of around 100 - 120 psi (pounds per square inch).

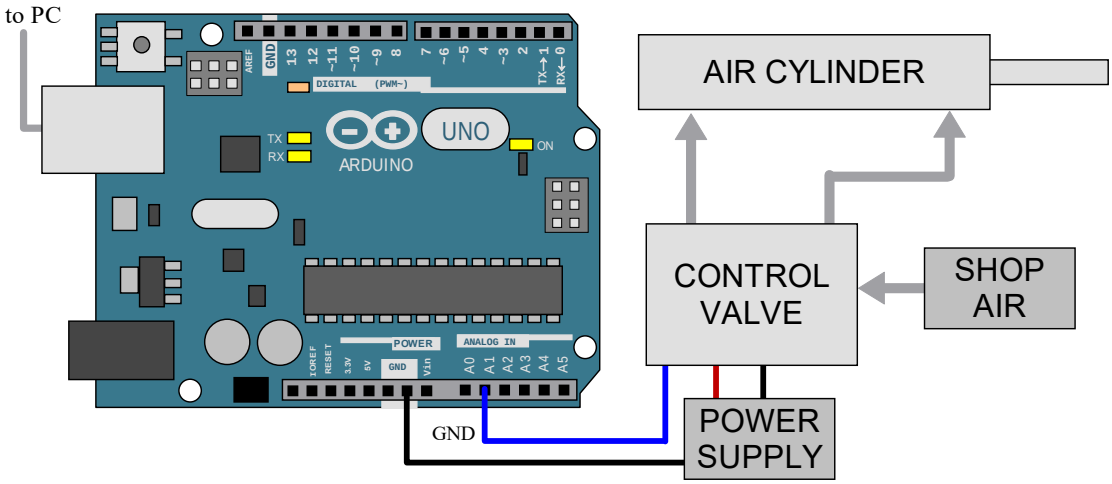
Air pressure is directed to the air cylinder through a control valve. Two common types of pneumatic control valves are solenoid and servo. Solenoids are on/off. Servo-valves can modulate or meter air flow over a continuous range. Servo-valves are more expensive than solenoid valves.

Pneumatic actuation is very strong compared to that of electric motor actuation. For instance, a 1-inch diameter air cylinder can produce almost 80 pounds of force when driven with 100 psi pressure.

12.1 SOLENOID VALVE AIR CYLINDER CONTROL

Here we will use Arduino to output a command signal to a control valve. A separate power supply is needed to power the control valve.

HARDWARE & CONNECTIONS



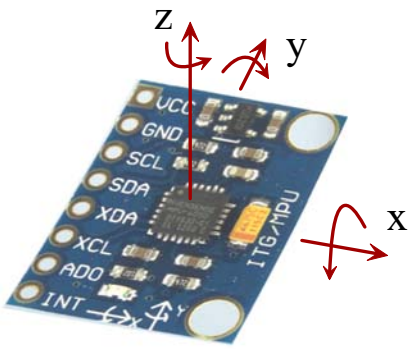
CODE

```
// pneum1_solenoid.ino -------------------------------------
// control movement of air cylinder using SOLENOID VALVE
// TO DEVELOP THIS CODE

#include <Servo.h>

Servo bldc;

void setup()
{
  bldc.attach(9,1000, 2000);          //bldc signal pin (8 is not pwm)
  bldc.write(0);                      // arm the ESC
  delay(1000);                        // 2000 to 3000 ?
  Serial.begin(9600);
}
void loop()
{
  bldc.write(50);               //map to 0 - 180 (0 = 1000 us, 180 = 2000 us)
}
```

12.2 SERVO VALVE

Here we will control an air cylinder using a servo valve and position feedback. With this system we should be able to control the linear position of the air cylinder.

LAB 13 - ACCEL & GYRO



IN 134 OLD LABS

ITEMS NEEDED

1. Arduino, cable, computer with Arduino IDE
2. Inertial motion unit (MPU6050, 6-axis accelerometer/gyro)
3. Bb and wire

INTRO/BACKGROUND

A drone or balancing robot needs motion information in order to perform properly. Inertial motion units (IMU's) sense motion data. A popular IMU is the **MPU-6050** sensor. It is a 6-axis accelerometer and gyroscope (meaning it senses linear accelerations in the x,y, and z axes, and rotation RATES about the x,y, and z axes with its gyroscope). The MPU-6050 also has a digital motion processor (DMP)

The MPU-6050 communicates with Arduino using a **serial** communication protocol called **i2C** ("i-squared C" or inter-integration circuit bus).

I2C

The i2C communication protocol allows a master controller to control or sense many different SLAVE devices using just 2 lines of communication. It can control up to 112 devices when using 7-bit addressing. It can control up to 1024 devices with 10-bit addressing. In our case, the Arduino will be the master. The IMU will be the slave. Each device has a pre-set ID (usually in hex form). There are 2 lines of communication. The serial clock line (SCL) synchronizes the data transfer. The serial data (SDA) line carries the data.

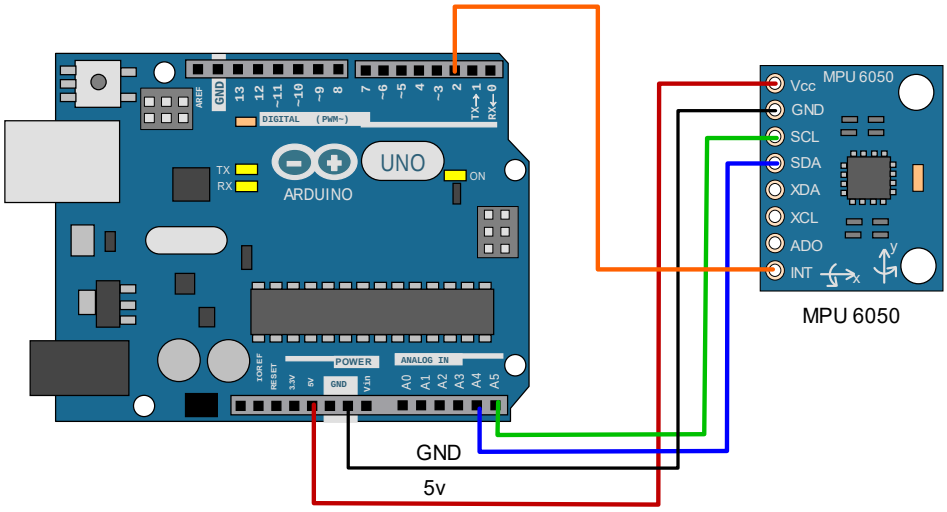
GYRO DRIFT & KALMAN FILTERS

To get accurate acceleration readings it is necessary to obtain acceleration data from both the accelerometers and the gyroscopes. Gyroscopes measure rotation RATE, so to get rotation angle you must INTEGRATE the signal. The problem with this is that you get DRIFT. When you integrate there is a constant of integration. The drift starts to add up over time. One way of dealing with this unwanted drift is to use a KALMAN FILTER (or linear quadratic estimation). This algorithm uses a series of measurements over time, which include noise and other inaccuracies, to produce estimates of unknown variables that tend to be more accurate than those based on a single measurement. The code below does NOT include a Kalman filter.

13.1 EXERCISE 1

Here XXX

HARDWARE & CONNECTIONS



MPU 6050

CODE

Include the wire.h library to implement i2c.
The first step will be to initialize the MPU (wake it up, specify ranges).
The next step will be to calibrate the MPU (the MPU should be on a level surface and not moving).
The final step will be to collect data from MPU.

Here are some informative videos.

How To Mechatronics.
https://www.youtube.com/watch?v=UxABxSADZ6U

explains mpu, transfer yaw
https://www.youtube.com/watch?v=4BoIE8YQwM8
mpu - drift
https://www.youtube.com/watch?v=j-kE0AMEWy4

The code requires that you read data directly from the MPU registers.

```
// // MPU6050_i2c.ino ------------------------------------
// Communicate motion data with the MPU6050 IMU using i2c protocol
// fm How to Mechatronics

#include <Wire.h>                       //nec for i2c

//place MPU on level surface and obtain pitch & roll values
//enter the values below
#define pitchAtLevel -0.5
#define rollAtLevel -0.4
```

```
const int mpuAddr = 0x68;       //i2c address for mpu

int gyroX, gyroY, gyroZ, temp;
long  gyroXcal, gyroYcal, gyroZcal;
long accX, accY, accZ, accMag;
float accelMag, pitch_fm_accel, roll_fm_accel, pitch, roll;
float pitchOut, rollOut;
boolean set_gyro_angles = 0;

long loop_timer;
long tlast;


void setup()  //-------------------------------------------------------
{
  Serial.begin(9600);
  //Serial.begin(115200);

  setup_mpu();                        //initialize & configure mpu
  calib_mpu();

  loop_timer = micros();

}

void loop()         // ---------------------------------------
{
  read_mpu();

  //implement offsets (like zeroing mpu)
  gyroX -= gyroXcal;
  gyroY -= gyroYcal;
  gyroZ -= gyroZcal;

  //compute pitch & roll angles from gyro raw data
  //  250 cyc * 65.5 out    = xx cyc*out/deg --> recip = deg/cyc/out
  //       s       1 deg/s

  pitch += gyroX/250/65.5 //250 Hz, 65.5 rawOut/1 deg/s
  roll += gyroY/250/65.5

  //couple yaw with pitch and roll
  // nec when getting angles from integrating angle rates (gyros)
  //(xx/25/65.5*PI/180) = angle in deg
  pitch += roll * sin(gyroZ/250/65.5*PI/180);
  roll -= pitch * sin(gyroZ/250/65.5*PI/180);

  accMag = sqrt(accX*accX + accY*accY + accZ*accZ);

  pitch_fm_accel = asin((float)accY/accMag)*180/PI;
  roll_fm_accel = asin((float)accX/accMag)*180/PI;
  //180/PI – to convert to degrees

  //implement accelerometer offsets
  pitch_fm_accel -= pitchAtLevel;
  roll_fm_accel -= rollAtLevel;

  //compensate for drift using accel data (only sm amt .0004 is nec)
  //accel output = 4096/1g (@ +/- 8G setting)
  if(set_gyro_angles)
  {
```

```
    pitch = pitch*0.9996 + pitch_fm_accel*0.0004;
    roll =  roll*0.9996  + roll_fm_accel*0.0004;
  }
  else  // enter this 1st time around
  {
    //pitch/roll fm gyros must be initialized (what is init angles?)
    //set = pitch/roll fm accelerometers (ok if mpu is level and still)
    pitch = pitch_fm_accel;
    roll = roll_fm_accel;
    set_gyro_angles = true;
  }

  //complementary filter to combine accel & gyro data
  pitchOut = pitchOut *0.9 + pitch*0.1;
  rollOut = rollOut *0.9 + roll*0.1;

  //serial monitor output, quotation marks could be problematic
  if (millis() - tlast > 300)  //write to SM ev 0.3 sec.
  {
    Serial.print("accX = "); Serial.print (accX);  Serial.print("   ");
    Serial.print("accY = "); Serial.print (accY);  Serial.print("   ");
    Serial.print("gyroX = "); Serial.print (gyroX);  Serial.print("   ");
    Serial.print("gyroY = "); Serial.print (gyroY);  Serial.print("   ");
    Serial.print("gyroZ = "); Serial.print (gyroZ);  Serial.print("   ");
    Serial.print("pitch = "); Serial.print (pitchOut);  Serial.print("   ");
    Serial.print("roll = "); Serial.println (rollOut);
    tlast = millis();
  }

  //run loop() at 250 Hz (timer = 4000 us (250 Hz))
  while(micros() - loop_timer < 4000);      //stops
  loop_timer = micros();

}


void setup_mpu()//---------------------------------------------------
{
  Wire.begin();                              //init communic

  Wire.beginTransmission(mpuAddr);    //start communic w/ mpu
  Wire.write(0x6B);             //talk to register 6B
  Wire.write(0x00);             // place 0 into register 6B (wakes 6050)
  Wire.endTransmission();

  //set accel scale (+/- 8G)
  Wire.beginTransmission(mpuAddr);    //start communic
  Wire.write(0x1C);             //talk to
  Wire.write(0x10);             //set requested starting register
  Wire.endTransmission();

  //set gyro scale to (500 deg/s full scale)
  Wire.beginTransmission(mpuAddr);    //start communic
  Wire.write(0x1B);             //send requested starting register
  Wire.write(0x08);             //set requested starting register
  Wire.endTransmission();

}

void calib_mpu() //--------------------------------------------------
{
  itot = 2000;      //# of samples to take
```

```
  for (int i = 0; i < itot; i++)
  {
    read_mpu();
    gyroXcal += gyroX;
    gyroYcal += gyroY;
    gyroXcal += gyroZ;
    delay(3);
  }

  gyroXcal /= itot;
  gyroYcal /= itot;
  gyroZcal /= itot;
  //keep MPU level & still & get avg of 2000 readings – these are offsets
}


void read_mpu() //-----------------------------------------------------
{
  Wire.beginTransmission(mpuAddr);
  Wire.write(0x3B);                   //starting reg
  Wire.endTransmission();

  Wire.requestFrom(mpuAddr,14); //request 14 registers (3 accel, 3 gyro, 1 temp)

  while(Wire.available() < 14);       //wait til all 14 bytes rec'd

  //data for ea access stored in 1 bytes
  //left shift 8 bits, turn 2 8-bit vals into 1 16-bit val
  //read and combine 2 registers to get single value for ea sensor (eg accelX)
  //as ea sensor uses 2 registers to store data
  //use "bit shifting"
  accX = Wire.read() << 8|Wire.read(); //combine 0x3B(accelX_H) & 0x3C (accelX_L)
  accY = Wire.read() << 8|Wire.read(); //accelY
  accZ = Wire.read() << 8|Wire.read();      // accelZ
  temp = Wire.read() << 8|Wire.read();      // temp
  gyroX = Wire.read() << 8|Wire.read();     // gyroX
  gyroY = Wire.read() << 8|Wire.read();     // gyroY
  gyroZ = Wire.read() << 8|Wire.read();     // gyroZ
}

// end -----------------------------------------------------------
```

LAB 14 - HAND TOOLS

ITEM LIST

1.) Hand tools

14.1 HAND TOOLS

INTRO

Even with the advent of automation and industrial machinery it is important for technicians to know how to properly use hand tools. This includes knowing the proper tool for the job, how to use the tool, and how to use it safely.

COMMON TOOLS

Below is a list of commonly-used tools you might find in a garage, or a fabrication laboratory.

Screwdrivers – flat head, Phillips (different sizes)

Wrenches – SAE (inches) vs. Metric (SI).
1. Open-ended, box, combination
2. Adjustable wrenches (sometimes called "crescent" wrenches)
3. Pipe wrenches

Saws – hack, wood saws,
- hack saws – pull vs. push,
- kerf,

Ratchets, sockets

Files, grinders

Tap and die – to create female and male threads

Power tools – jig saws, power drills, circular saws, grinders

Safety and eye protection

PROPER USE OF TOOLS

Take apart an assembly, device, or machine.
Put it back together.
Take it home (don't leave it in the lab).

EXERCISE

Take apart an assembly, device, or machine.
Put it back together.
Take it home (don't leave it in the lab).

LAB 15 - EXTRA FIGURES

ITEM LIST