How to Use an Arduino with Firgelli Automations OS series Actuators

When you need to know exact positioning of your actuators, the OS Series is what you need. This powerful, reliable actuator comes with a built in optical sensor that provides feedback when you need it. The OS Series have no built-in controller, but do provide single phase pulses as position feedback signal that can be input to an external controller such as an Arduino.

The OS series is equipped with a 10 hole optical encoder disk that with the gearing to the actuator output shaft will send 50 pulses/inch of stroke for the 35lbf unit and 100 pulses/inch of stroke for the 200lbf and 400lbf units. (+/- 5 pulses).

The optical encoder is incremental , not absolute. So you need to count the pulses from a reference position to determine the position.

Typically, you retract the actuator, reset a pulse_counter variable to zero, and then as the actuator extends , count the pulses and increment the pulse_counter .

If you retract the actuator you decrement the pulse_counter.

For use with the Arduino, the actuator needs to beinterfaced via a motor-driver as the Arduino will be unable to provide the voltage and current requirement for the actuator. Typically 12VDC and a current of 3-5A.

A recommended motor driver is the IBT_2 H-Bridge : https://www.firgelliauto.com/products/high-current-dc-motor-drice-43a , this unit interfaces with an Arduino conveniently.

The OS series optical encoder needs to have an external “pull-up” resistor to +5VDC, 10Kohm or 15Kohm is nominal from the sensor output of the Actuator.

Or you can set the Arduino input pin to mode INPUT_PULLUP as shown in the example below:

```
const int  OPTICAL_Pin = 2;    // the pin that the optical sensor o/p

pinMode(OPTICAL_Pin, INPUT_PULLUP); // initialize the Optical Sensor pin as a input
```

Without the pullup,-- external resistor or in software , you will get erroneous results.

You check the pulses by using the Arduino interrupts or a simple polling. Interrupts are sensitive to noise, so be aware.

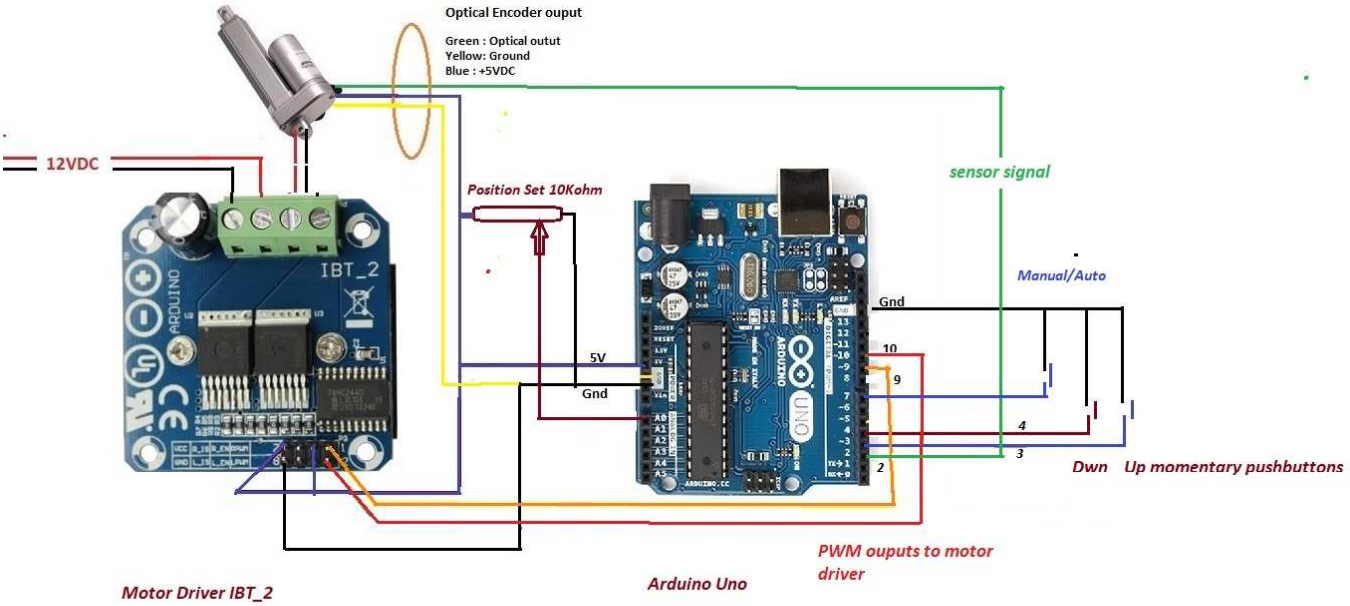
The attached code uses interrupts.

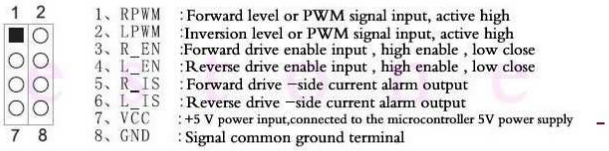
NOTE: You also have to load the elapsedMillis and Bounce2 libraries from Arduino library. Else the code will not compile.

In the example outlined here, a simple potentiometer controls the position set.

Position setting of the actuator is achieved by a simple 3 band control via nested if statements, you could use the Arduino pid library, if better control is desired. The complexity of pid is not really required in all cases. Follow the KISS principle as far as possible (Keep It Simple Stupid).

Wiring Diagram:





VCC pick MCU 5V power supply, GND connected microcontroller GND
R_EN and L_EN shorted and connected to 5V level, the drive to work.
L_PWM, input PWM signal or high motor forward
R_PWM, input PWM signal or high motor reversal

Premium Optical Sensor Feedback Actuator position control via Arduino and Motor driver

Arduino Code:

```
// OS series Actuator Position Control with potentiometer
// Disclaimer: Support for Arduino coding is limited and is
// provided "as is" without warranties of any kind.
// Sections of code samples from 3rd party sources is acknowledged

#include <elapsedMillis.h>
```

```
#include <Bounce2.h>
#define debounceDelay 50 // the debounce time in ms; decrease if quick button presses are
                  //ignored, increase if you get noise (multiple button clicks detected
                                          //by the code when you only pressed it once)

#define falsepulseDelay 2 // noise pulse time  , if too high, ISR will miss pulses.

#define ShftUpCMND 3    // This is momentary switch Input3
#define ShftDWnCMND 4          // This is momentary switch Input3
#define ManAutoCMND 7 // This is momentary switch Input3
#define Zero 10000
elapsedMillis timeelapsed ;
unsigned int  waitInterval ;
Bounce debouncerUP = Bounce(); // Instantiate a Bounce object for UP switch
Bounce debouncerDWn = Bounce(); // Instantiate another Bounce object for Down switch
Bounce debouncerManAuto = Bounce(); // Instantiate a Bounce object for ManAuto switch


unsigned long now;
unsigned long lastPulsecame;
volatile long lastDebounceTime = 0;   // the last time the interrupt was triggered
volatile unsigned int Pulse_Counter = Zero;   // counter for the number of pulses
volatile unsigned int previousPulse_Counter = 0;   // counter for the number of pulses
volatile unsigned int Counter_diff = 0;   // counter for the number of pulses difference
volatile unsigned int lastPulse;
unsigned int ValueA0;

// define motor drive pins Extend and Retract for actuator
volatile char FwdRev = 'S' ;
int RPWM_Output1 = 9;   // Arduino PWM output pin 5; connect to IBT-2 pin 1 (RPWM)
int LPWM_Output1 = 10; // Arduino PWM output pin 6; connect to IBT-2 pin 2 (LPWM)
int act_PWM =255;                  // define speed value, 255 is full speed
int act_MidPWM=200; // speed when in midband
int act_LoPWM =150; // speed when in low band

unsigned long  start, finished;
float elapsed, act_sp;

const int  OPTICAL_Pin = 2;    // the pin that the optical sensor o/p
const int set_potA0=A0; // Position set pot input pin
int set_potValueA0=0; // variable to hold pot value
int pulse_timeout=500;
int strokeCount;
```

```
boolean  this_buttonState; // variable for reading the pushbutton status
boolean timed;
boolean state =LOW;
boolean printonce=HIGH;
boolean quitLoop =LOW;
boolean timeelapsedFired;

int samplesize=5;
int sampleInterval=10; // time between samples in ms.



//  ----------------------------------------Routine to get average value of input pot

float getSensorAverageValue ( int sensorPin,   int numberOfSamples,  long
timeGap)
{
  static int currentSample; // current sensor sample.
  static float currentValue;  // current sensor value.

  // current value works as a sum counter.
  currentValue = 0;

  // get sensor samples with delay and calculate the sum.
  for (int i = 0; i < numberOfSamples; i++) {
    // get sensor sample.
    currentSample = analogRead(sensorPin);

    // add sample to the sum counter.
    currentValue += currentSample;

    // delay some time for the next sample.
    delay(timeGap);
  }

  // get the average sensor value (ignore the fraction).
  return (currentValue / numberOfSamples);
}



// ---------------------------------ISR ----------------------------------------------------
--
void count_ISR() {
  // This function is called by the interrupt
      if ((millis() - lastDebounceTime) > falsepulseDelay) { //if current time minus
the last trigger time is greater than the delay (pulse) time, pulse is valid.
         lastDebounceTime = millis();
         //received valid pulse
         if ( FwdRev=='E'){
            Pulse_Counter++;
```

```
        }
        if ( FwdRev=='R'){
          Pulse_Counter--;
         }
    }
  } // End of ISR

// ------------------------------time delay----------------
void  timeDelay( unsigned int waitInterval) {



   while (timeelapsed < (waitInterval) ) {

        //Serial.println("looping in delay");
        //Serial.println(timeelapsed);
        //Serial.println(waitInterval);
        //timeelapsed = 0;          // reset waiting back to zero
      }
   } // End of time delay


void limit_check() {
   quitLoop=HIGH;
   //Pulse_Counter = 0;
  while (quitLoop==HIGH) { // check if actuator has reached fully extend or
retract, ie no pulses for 500ms
      previousPulse_Counter = Pulse_Counter;
      timeelapsed = 0;
      timeDelay(pulse_timeout);
      if ( FwdRev=='E'){
          Counter_diff= Pulse_Counter - previousPulse_Counter;
        }else{
            Counter_diff= previousPulse_Counter - Pulse_Counter ;
         }

      if (Counter_diff<=0) {
        driveActuator( 1 , 'S',act_PWM ); // Stop  actuator
        quitLoop=LOW;
       }
  }
  Serial.print(" \n Reached End Limit");

}

//------------routine to drive actuator. Inputs are Actuator number and mode :
S:off, E:Extend, R:Retract

// Sets one relay ON and the other OFF to extend or retract the actuator, If both
relays OFF, actuator Stops

void driveActuator( int chosenActuator , char mode , int speed_PWM ) {
```

```
  int relayFWD=9;
  int relayREV=10;
  int fdrive=0;
  int rdrive=0;

  switch (chosenActuator) {
     case 1:
       relayFWD=9;
       relayREV=10;
     break;

     case 2:
        // for future use
        //relayFWD=5;
        //relayREV=6;
     break;


  }
  switch (mode) {
     case 'S':
     fdrive=0;
     rdrive=0;
     break;

     case 'E':
     fdrive=0;
     rdrive=speed_PWM;
     break;

     case 'R':
     fdrive=speed_PWM;
     rdrive=0;
     break;

  }
    analogWrite(relayFWD, fdrive);
    analogWrite(relayREV, rdrive);



}
// ---- EO Actuator Drive

void setup() {
  Serial.begin(9600);
  // connect AREF to 3.3V and use that as VCC, less noisy!
  analogReference(EXTERNAL);

  pinMode(RPWM_Output1, OUTPUT);  //define motor drive ouput pins
  pinMode(LPWM_Output1, OUTPUT);   //define motor drive ouput pins
  pinMode(OPTICAL_Pin, INPUT_PULLUP); // initialize the Optical Sensor pin as a
input:
```

```
  attachInterrupt(0, count_ISR, RISING); // optical sensor output to Arduino pin2
interrupt
  pinMode(ShftUpCMND, INPUT_PULLUP);  //define inputs for control and attach
to Bounce routine
  debouncerUP.attach( ShftUpCMND );
  debouncerUP.interval(debounceDelay); // interval in ms
  pinMode(ShftDWnCMND, INPUT_PULLUP); //define inputs for control and attach
to Bounce routine
  debouncerDWn.attach( ShftDWnCMND );
  debouncerDWn.interval(debounceDelay); // interval in ms
  pinMode(ManAutoCMND, INPUT_PULLUP); //define inputs for control and attach
to Bounce routine
  debouncerManAuto.attach( ManAutoCMND );
  debouncerManAuto.interval(debounceDelay); // interval in ms

  pinMode(set_potA0, INPUT);

   // Retract fully  actuator
   Pulse_Counter = Zero;
    FwdRev = 'R';
    driveActuator( 1 , FwdRev ,act_PWM );
  Serial.print("\n First retract  & delay");
  limit_check(); // if reached end-limit send stop to actuator

  // Extend fully  actuator
  Pulse_Counter = Zero;
  FwdRev ='E';
  driveActuator( 1 ,FwdRev,act_PWM );
  Serial.print("\n Started Extend ");
  limit_check();
  Serial.print(" \n Extend Pulse Counter = ");
  Serial.print(Pulse_Counter);
  strokeCount=Pulse_Counter;

 // Retract fully  actuator

   FwdRev ='R';
  driveActuator( 1 ,FwdRev,act_PWM );
  Serial.print("\n Second retract  & wait ");
  limit_check();
  Serial.print(" \n Retract Pulse Counter = ");
  Serial.print(Pulse_Counter);

  timeDelay(5000); // wait 5 seonds


} // End of setup


void loop() {
//-------------------------------------main Routine ----------------------------

  debouncerUP.update();
```

```
  debouncerDWn.update();
  debouncerManAuto.update();
  if (printonce==HIGH) {
  Serial.print("\n looping in main");
  Serial.print(" \n Pulse Counter = ");
    Serial.print(Pulse_Counter); /// Pulse counter equals the retract counter
number
  printonce =LOW;

  }
  Serial.println(" loop ..... ");

  this_buttonState= debouncerUP.read() ; // check if UP is pressed: i.e. has gone
LOW
  while (this_buttonState ==LOW  ) {  // Drive actuator to extend
     //Serial.print(" \n Up Button pressed ");
     //Pulse_Counter = 0;
     FwdRev ='E';
     driveActuator( 1 ,FwdRev,act_PWM );
     //Serial.print("\n Started Extend ");
     //Serial.print(" \n Extend Pulse Counter = ");
     Serial.println(Pulse_Counter);
     debouncerUP.update();
     this_buttonState= debouncerUP.read() ; // check if UP is pressed: i.e. has
gone LOW
     }



  this_buttonState= debouncerDWn.read() ; // check if DWn is pressed: i.e. has
gone LOW
  while (this_buttonState ==LOW  ) {  // Drive actuator to extend
     //Serial.print(" \n Down Button pressed ");

     //Pulse_Counter = 0;
     FwdRev ='R';
     driveActuator( 1 ,FwdRev,act_PWM );
     //Serial.print("\n Started Extend ");
     //Serial.print(" \n Extend Pulse Counter = ");
     Serial.println(Pulse_Counter);
     debouncerDWn.update();
     this_buttonState= debouncerDWn.read() ; // check if Down is pressed: i.e.
has gone LOW
     }



     //limit_check();

   // Read potentiometer to get set position
// sensor value is in the range 0 to 1023
```

```
  this_buttonState= debouncerManAuto.read() ; // check if ManAuto is pressed:
i.e. has gone LOW, i.e Auto
   while (this_buttonState ==LOW  ) {  // Set position based on pot
 //do {

  set_potValueA0=
getSensorAverageValue(set_potA0,samplesize,sampleInterval);
  ValueA0 = map (set_potValueA0, 0, 1024, Zero, strokeCount);


if (Pulse_Counter < (ValueA0-10)) {
   act_PWM=255;
   FwdRev ='E';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}
else if(Pulse_Counter < (ValueA0-5)) {
   act_PWM=act_MidPWM;
   FwdRev ='E';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}
else if (Pulse_Counter < (ValueA0-2)) {
   act_PWM=act_LoPWM;
   FwdRev ='E';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}
else if (Pulse_Counter < (ValueA0-1)) {
   act_PWM=act_LoPWM;
   FwdRev ='S';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}


if (Pulse_Counter > (ValueA0+10)) {
   act_PWM=255;
   FwdRev ='R';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}
else if(Pulse_Counter > (ValueA0+5)) {
   act_PWM=act_MidPWM;
   FwdRev ='R';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}
else if (Pulse_Counter > (ValueA0+2)) {
   act_PWM=act_LoPWM;
   FwdRev ='R';
   driveActuator( 1 ,FwdRev,act_PWM );
```

```
   timeDelay(100);
}
else if (Pulse_Counter > (ValueA0+1)) {
   act_PWM=act_LoPWM;
   FwdRev ='S';
   driveActuator( 1 ,FwdRev,act_PWM );
   timeDelay(100);
}

 Serial.print( "\n Desired  Position= ");
 Serial.print(ValueA0);
 Serial.print( "\n Current Position= ");
 Serial.print(Pulse_Counter);
 debouncerManAuto.update();
 this_buttonState= debouncerManAuto.read() ; // check if ManAuto is pressed:
i.e. has gone LOW, i.e Auto
 }

  FwdRev ='S';
  act_PWM=255;
  driveActuator( 1 ,FwdRev,act_PWM );
  timeDelay(100);
} //end Void Loop
```

****