Human Interface Device for Rehabilitative Action on the Radiocarpal Joint

Capstone Final Report
Spring 2014

Jennifer Schror
John Moffitt

Advisor: Professor Burdea

Department of Electrical and Computer Engineering
Rutgers University
Table of Contents

I. Abstract

II. Overview

III. Hardware
   a. Rotary Actuator & Power Supply
   b. Microcontroller
   c. Half Bridge Force Sensors & Analog Circuit
   d. Forearm & Hand Platform, Servo Mount Block

IV. Software
   a. Python Code
   b. Arduino Code

V. Sustainability Analysis

VI. Conclusion

VII. References

VIII. Appendices
   a. Arduino Code
   b. Python Code
      1. Exercise Test
      2. Plotting
   c. Materials List
I. Abstract

Injury and illness presents both an emotional and economic burden on an individual. Technological developments have made robotic intervention increasingly affordable and commonplace. Patients can achieve desired range of motion, strength, and ultimately increased function despite injury. Individuals requiring physical therapy or advanced ergonomic movement may benefit from the creation of a human interface device and companion computer software.

This system is designed and constructed as a novel HID for rehabilitation directly aimed at injuries of the wrist during flexion and extension. The device rests on a flat surface and has a single movable platform that rotates with the motion of the user's wrist. This utilizes a force sensor setup to measure applied palm pressure. A software solution is also integrated with the goal of aiming and guiding the patient in recovery. This interfaces the microcontroller with a computer to prompt the user to select angle of motion and direct to apply force. The software also tracks the history of the force applied over time.

One of the biggest hurdles with the development of the system was mastering serial communication. Designing the sensor circuit also presented an issue. This did not disqualify the feasibility of such a project, however. Within this growing field of computer-assisted physical therapy, the robotic rehabilitative device has much potential for use.

Keywords - Rehabilitation, servomotor, bidirectional serial communication, wrist, capstone.

II. Overview

The development of this system realizes the primary objective of designing and constructing a novel HID for rehabilitation directly aimed at injuries of the wrist during flexion and extension. This is done through the implementation of a microcontroller-based system for the control and movement of a motorized platform. This utilizes a force sensor setup to measure increasingly applied palm pressure during recovery. In addition, a software solution is created with the goal of aiming and guiding the patient. This is accomplished through the implementation of an interface between the microcontroller and a computer to prompt the user to engage the platform's angle of motion and direct him or her to apply force.

The system works by having the user place his or her wrist on the wooden forearm and hand platform. After calibration of the load sensors, the user is then asked to apply pressure in order to accomplish flexion of the wrist and orient the platform at a horizontal angle. The user is next prompted to relax the wrist and hand while the
platform returns to the initial position. The user is finally asked to apply increasing force in order to slowly guide the platform into a fully lowered position. The prompting is accomplished through a graphical interface with arrow guides. The level and intensity of the pressure is measured and this data is used to determine whether the correct rehabilitative action has been achieved.

III. Hardware

a. Rotary Actuator & Power Supply

The motor chosen is a 180° maximum rotation pulse width modulated servomotor. This device allows for 700 to 2300 μs range of motion, and a regulated DC power supply at 4.8 V allows for 0.95sec/60° of motion. This provides for the possible 1375 ounce-inches of torque. An LED setup indicates direction of motion and power.

This particular model was chosen for a number of reasons. The primary requirement is the static and dynamic torque that the servo can supply. Maximum torque rating on servos is typically for static torque, since dynamic torque is lower. A human hand can apply approximately 10 pounds of weight at the fingertips, while the forearm is held flat against a surface. A human hand is typically around 7 inches. Assuming all the force is applied at the fingertips, the torque of a human hand is approximately (10 lbs.)*(7 in.) or 1120 oz.-in. This estimate is generous, since much of the force will be applied through the palm in this application, which is less than 7 in. from the wrist.

Because a servo can only apply a relatively low amount of torque, a gearbox was used. For example, the servo without a gearbox can only apply a maximum torque of 343 oz.-in. A gearbox linearly increases the torque that a servo can apply, while linearly decreasing the speed of rotation. A 5:1 ratio gearbox was used, which scales the torque from 343 to 343*5 or 1715 oz.-in. This satisfies the primary requirement of the servo.
It needs to be able to resist force applied by a hand and be able to hold its position at the same time. It also needs to be able to apply enough torque to push a hand that is resisting movement.

The gearbox chosen also includes a feedback potentiometer, which helps maintain an accurate position, even if pushed hard enough to overcome the static torque of the servo. The feedback loop allows the servo to detect that it is in the incorrect position, and adjust immediately.

The second most important requirement for the servo is the range of motion. The chosen gearbox allows for 180° rotation, as the servo used is modified by the supplier to allow for a wider range of motion. Normally the servo allows for 180° of motion, and the gearbox would divide that by 5. The feedback modification done by the supplier also includes the increased range of motion. This allows for a pulse signal between 700 to 2300 μs to correspond to the 180° range.

The last mandatory requirements are the power and signal needs of the servo. Since we chose to use an Arduino microcontroller, which operates at 5V and can send a pulse width modulated signal at 5 V, the servo has to be able to use a 5V signal. Initially, the servo was powered through the Arduino, so a servo that can operate at 5V power was chosen. However, a separate power supply was later added to prevent the servo from drawing too much current from the Arduino's power circuit.

The speed of the servomotor is also an important factor. If the servo is too slow, it can become tedious to do range of motion exercises, as the user will have to slow down for the servo to keep up. The 5:1 ratio gearbox divides the servo's speed (in °/sec) by 5. The servo's maximum speed is 0.14 sec/60°. This leaves the servo-gearbox combination with a speed of 0.7 sec/60°. Since the device will only require 180° of motion, it takes the device about 2 seconds to follow the full range of motion (2). While ideally, it would be faster, tradeoffs had to be made to maintain a reasonable price, and speed is not the primary requirement.

Less important requirements include weight, size, and possible attachments for mounting it to the moving platform. The device was designed around these specifications, instead of the other way around, because there are a limited number of available gearboxes and servos that can supply the torque required.
Also of note is the attachment used to join the servomotor with hand platform. It is a single direction 3 in. aluminum arm. The arm mounts directly on the gearbox and is strong enough to move the hand platform without breaking under the torque.

b. Microcontroller

The microcontroller selected was a 5 V/16 MHz Arduino with ATmega32U4 processor and built-in USB connection. The device consists of several analog I/O pins, of which 2 are used for reading the load sensor values. In addition, pulse width modulation capable pins allowed for control of the servomotor.

Fig. 3: The layout of the Arduino
Pro Micro - 5V/16 MHz

The USB transceiver allows for serial communication, which is accomplished through the computer. This micro-B USB connection connects to the computer through a standard USB port. The USB port allows for the Arduino to be powered at 5 V however the microcontroller is capable of being powered with up to 12 V due to the onboard voltage regulator. Ease of use and portability requirements mandated a choice of USB power for this system.

This micro-B USB port also allows for power to be drawn from the Arduino. The VCC pin supplies 5 V in order to power the servomotor direction controlled LEDs and the 2 INA125P amplifiers. Because of potential fluctuations in current and voltage drawn, it was not feasible to power the servomotor itself with the Arduino.

The analog I/O pins consist of analog to digital convertors that allow for the input of analog signals. The analog to digital convertors are 10-bit and assist in accurately reading the load sensor values. The pins with PWM functionality allow for analog writing. This enables accurate and fast bidirectional communication with the servomotor control for reading and writing the necessary angles of motion.

A switch circuit is designed to interface between the Arduino's RST and GND pins. This allows for the performance of quick resets on the device. This became an
important feature as the serial communication between the Arduino and computer has a
tendency to become garbled and fail to allow reading sporadically. This is because the
Arduino buffers and sends the data from the analog I/O pins and the PWM pins at a rapid
rate.

The Arduino also enables 3 LEDs in series with 220 ohm resistors to be powered
from an analog I/O pin. This circuit illustrates the servomotor's direction of motion and
indicates such to the user.

c. Half Bridge Force Sensors & Analog Circuit

In order to accurately measure the patient's applied force during recovery of wrist
motion, 4 50 kg load sensors were utilized. Each load sensor constitutes one half bridge
of a Wheatstone bridge. Therefore, it was first necessary to complete the bridge circuit in
order to accurately obtain force sensor readings and reduce risk of temperature variation.
This was accomplished by pairing the load sensors with one another in groups of 2 to
complete the Wheatstone bridge, rather than building resistor circuits. This had the added
advantage of discounting issues related to differing resistor tolerances.

Temperature sensitivity is an important issue when it comes to making specific
measurements with this type of load sensor. The sensor itself is rated at an output
sensitivity of 0.05% of the full scale of the output per 10° C. The operable temperature
range is also given as between 0 and 50° C\(^3\). These facts are important to keep in mind
because of potential issues regarding the patient's hand heating up the device or because
of heat generated from the servomotor and microcontroller circuit. It is also important to
remember that the general output is only within the small millivolt range.

![Fig. 4: Size of a single load sensor](image)

The load sensor complete bridge circuits allowed for less than 10 V for excitation.
This made it possible to power the sensor circuits with the 5 V VCC pin on the Arduino.
This power circuit is joined with the amplification circuit in order to create usable load
sensor output values. This is because the output sensitivity of a single load sensor is 1.0
+/- 0.1 mV/V.
Portability and convenience necessitated choosing an amplifier that could both accurately create gain on such small input signals and not need an additional power supply to create bias voltage. This is possible with the use of the INA125P instrumentation amplifier. Rapid and accurate adjustment of the load sensor gain is possible with the use of 2 potentiometers as variable resistors. The gain is given as $G = 4 + 60 \text{kilo ohms} / R_g$.

The 2 amplifier circuits are simple, depending only on the potentiometers and power supplied from the Arduino. The INA125P provides high accuracy on low power. There is low offset drift and low input bias current $^{(1)}$.

d. **Forearm & Hand Platform, Servo Mount Block**

The forearm platform was chosen to be 12 in. long and 3-1/2 in. wide to accommodate any arm, large or small. Originally a 2x4 (dimensions are 1-1/2 by 3-1/2) cut to 12 in. was used, but the surface was not smooth, so a 1/2 in. thick piece of pinewood was screwed on top of the 2x4. This also proved to be a more reliable surface to attach the hinges onto. The hinges are standard cabinet hinges, but they were chosen based on a few factors: how smoothly they rotate, if they cover the required range of motion, and most importantly, they must be able to be mounted a few inches hanging off the edge of the forearm platform, which allows the lower part of the hand platform to hang downward without touching the forearm platform.

The lower edge of the hand platform was also sanded to ensure no contact is made which could prevent a full range of motion on the device. The hand platform was designed to be 7 in. by 3-1/2 in. so that it would line up with the forearm platform evenly, and to fit all but the largest of hands. Since the load sensors need to be able to detect force regardless of the orientation and motion of the hand platform, it had to be designed as two separate pieces of wood. The base piece connects to the rest of the device (hinges and servo arm), and the floating piece is only connected to the base piece by the load sensors.
The floating piece is made up of poplar board. This lightweight material is mounted at 3 points to the base piece. This design decision was made as a precaution against warping. If the boards of wood warp, and the floating piece was mounted by 4 points of contact, one of the points of contact could potentially become detached and prevent accurate force detection. With 3 points of contact, the board can warp a reasonable amount, and the 3 points of contact will remain in contact. 2 of the points of contact are the load sensors, and the third is a small piece of metal. Metal was used because it does not compress very much and as a result will not absorb any of the force on the hand platform.

Unfortunately, wood itself is capable of absorbing loads to some extent, compared to metal or plastic, so the hand platform design is not perfect. Wood was the optimal choice for this design because it is not a conductor of heat, while metal is. The load sensors are very sensitive to heat, and a human hand generates enough heat to completely throw off any accurate load sensor reading. Since a human hand is very close the load sensors in this device, metal cannot be used for the hand platform without some strong heat insulator between it and the load sensors. So while wood causes a small drop in load sensor accuracy because it compresses slightly under a load, it simplifies the design and prevents temperature from affecting the load sensors.

Pine was used for the base piece of the hand platform because it is strong and thick enough to reliably attach the servo arm. Poplar wood was used for the floating platform because it is light and thin, so it won't add much extra force to the load sensors and servo.

Fig. 6: Servo mount block is pictured along with forearm and hand platforms

Also of interest is the servo mount block. Since the servo and gearbox used have a specific way of mounting to surrounding objects, a servo mount block had to be created to mount the servo to the rest of the device. The primary goal of the mount block is to firmly hold the servo and gearbox in the correct position, relative to the forearm and hand platforms. The center of the primary hub gear needs to be on the same axis as the hinge's axis of rotation on the rest of the device. The axis cannot become misaligned, so the block must hold firmly. Otherwise, the rotation of the hand platform will become difficult for the servo, and could break the servo or the hinges.

The block is designed as 4 small pieces of pinewood screwed together. They were
cut to size and drilled to fit perfectly and to tightly hold the servo and gearbox in place. The mount block is screwed directly into the side of the forearm platform.

IV. Software

a. Python Code

Python was used for the PC software because it allows for rapid iteration and there are a host of libraries available for use. Development of the user interface itself made use of the wxPython library. The plot program was completed with Python's matplotlib and math libraries. All Python development was done with the pySerial library.

The pySerial library allows for quick and easy bidirectional serial port reading and writing. Through the use of this and the ability to poll the Arduino serial port between fixed delays to read load sensor values, the user is able to read and send accurate data.

The GUI interface was created with the use of wxPython. The functions for updating and displaying images allow the user to see on screen the wrist movements in real time. This type of feedback assists the user in following the on screen prompts for motion.

![Fig. 7: The "Exercise Test"
program prompts the user for upward and downward movement](image)

The plotting portion of the Python code accomplishes buffered real-time graphing of the load sensor values. The processed rate at which the data changes, or baud rate, is set to 115200. Accuracy and rate of data input is set to the Arduino maximum.
b. **Arduino Code**

The bulk of the coding was done in the C/C++ based Arduino IDE. The provided Servo library makes reading from and sending data to the servomotor a simple matter of interfacing with the Python code. Checks are in place to insure that the servomotor never moves beyond its range of motion. To aid in this, delays are put into place.

V. **Sustainability Analysis**

Alternative devices designed to aid the user and speed up recovery time through the use of virtual reality or game setups have been implemented. However, these types of devices are so far limited to University settings and research driven hospitals. As such, the prices are generally very high. Because practically there are so few available options for patients seeking novel methods of rehabilitation, the demand may be high enough to sustain a profitable business.

The manufacture of this type of product could be feasible. This device consists of components costing under $250. The design can also be modified to use cheaper mass produced parts and less expensive materials. The servo gearbox, specifically, could be replaced by a much cheaper solution if the parts are manufactured in bulk, as opposed to ordered from a hobby website. The Arduino and amplifier circuits could be replaced by a simpler microcontroller and cheaper instrumentation amplifiers, especially if ordered in bulk. Lastly, assembly is very simple to begin with, and would only get simpler with a manufactured design. The general idea could also be applied to other joints and a higher range of motion (rotation and side-to-side bending) to create a whole suite of rehabilitation products.
VI. Conclusion

In the case of this device, initially, designing the hand platform to allow for the load sensors to accurately detect force, regardless of the motion or position of the servo was the most difficult challenge. Soon after, additional problems were discovered. The load sensors had to be set up to prevent temperature changes from affecting their accuracy. Mounting the hand platform to the servo proved to be a difficult task, since the hinges screw into the top of the board instead of the center. Properly amplifying the load sensor output was a challenge. After trying a variety of opamps, it was found that this type of amplification could not be done properly without an instrumentation amp. Getting reliable serial communication between PC and Arduino with Python took some time, but after a lot of trial and error, we figured it out.

Robotic rehabilitative devices have been an intensive field of research since the 1980s. The design, building, and coding of these devices is not without issues. Fortunately, in this case, each issue was overcome resulting in a finished product that matches closely to what was set out to be done.
References

# Appendices

## a. Arduino Code

```c
#include <Servo.h>

Servo myservo;
//int loadSensorValue = 0;
int loadSensorValue1 = 0;
int loadSensorValue2 = 0;
int newServoPosition = 1450;
int servoPosition = 1450; //platform is flat
const int ledPinUp = 8;
const int ledPinStop = 9;
const int ledPinDown = 7;

void setup() {
  Serial.begin(115200);
  myservo.attach(10);
  pinMode(ledPinUp, OUTPUT);
  pinMode(ledPinStop, OUTPUT);
  pinMode(ledPinDown, OUTPUT);
  moveArm();
}

void loop() {
  readLoadSensors();
  if(Serial.available() == 4) {
    receiveSerialInput();
  }
  else {
    sendSerialOutput();
  }
  //delay(100);
}

void readLoadSensors() {
  //int tempSensVal1 = analogRead(A0);
  //delay(100);
  int tempSensVal2 = analogRead(A2);
  delay(10);
}  
```
//loadSensorValue1 = (tempSensVal1 + tempSensVal2)/2;
//loadSensorValue1 = tempSensVal1;
loadSensorValue2 = tempSensVal2;
}

void receiveSerialInput()
{
    unsigned long serialValue = readLongFromBytes();
    if(serialValue <= 2150 & serialValue >= 950) { //max and min, reject all outside values to
        //prevent damage
        newServoPosition = (int)serialValue;
        moveArm();
    }
    //Serial.print(serialValue, DEC); //for testing
    //Serial.print('
');
}

void sendSerialOutput()
{
    //Serial.print(loadSensorValue1, DEC);
    //Serial.print(' ');
    Serial.print(loadSensorValue2, DEC);
    Serial.print(' ');
    Serial.print(servoPosition, DEC);
    Serial.print('
'); //NEEDED FOR READLINE TO WORK IN PYTHON, WILL HANG OTHERWISE
}

unsigned long readLongFromBytes() {
    union u_tag {
        byte b[4];
        unsigned long lval;
    } u;
    Serial.flush();
    u.b[0] = Serial.read();
    u.b[1] = Serial.read();
    u.b[2] = Serial.read();
    u.b[3] = Serial.read();
    return u.lval;
}

void moveArm()
{
    int servoDelay = 1; //approximately how long it will take to finish the movement
    ///.00135 seconds/microsecond input, use 1.5 just to be sure it finishes motion and to
    convert to ms
    if(newServoPosition > servoPosition) {
        newServoPosition = servoPosition;
    }
}
servoDelay = (newServoPosition - servoPosition) * 1.5;
digitalWrite(ledPinUp, HIGH);
digitalWrite(ledPinStop, LOW);
}
else if(newServoPosition < servoPosition) {
    servoDelay = (servoPosition - newServoPosition) * 1.5;
    digitalWrite(ledPinDown, HIGH);
    digitalWrite(ledPinStop, LOW);
}
servoPosition = newServoPosition;
myservo.writeMicroseconds(servoPosition);
delay(servoDelay);
//delay(1);
digitalWrite(ledPinUp, LOW);
digitalWrite(ledPinStop, HIGH);
digitalWrite(ledPinDown, LOW);
}
b. Python Code

1. Exercise Test

# contains code for debugging

import time
import serial
import struct
import wx
import math

class windowClassInner(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Exercise Test', size=(400, 400))

        panel = wx.Panel(self, -1)
        self.panel = panel

        # add pic
        self.imageFile = '/Users/m/Desktop/good code/untitled folder/handstop.png'
        image = wx.Image(self.imageFile, wx.BITMAP_TYPE_ANY)
        image = wx.Image.Rotate(image, math.radians(-78), wx.Point(0,0))
        x = (image.GetWidth() - 190)/2
        imageBitmap = wx.StaticBitmap(panel, -1, wx.EmptyBitmap(190,190))
        image = wx.Image.Resize(image, size=(190,190), pos=(-x,-x))
        imageBitmap.SetBitmap(wx.BitmapFromImage(image))
        self.imageBitmap = imageBitmap

        sizer = wx.BoxSizer(wx.VERTICAL)

        instructions = wx.StaticText(panel, -1, 'Press button to start exercise test')

        self.Bind(wx.EVT_CLOSE, self.closewindow)

        button1 = wx.Button(panel, label="Begin", pos=(270, 150), size=(60, 60))
        self.Bind(wx.EVT_BUTTON, self.onbutton, button1)

        text = wx.TextCtrl(panel, -1, style=wx.TE_MULTILINE|wx.TE_READONLY)
        self.text = text
sizer.Add(instructions, 0, wx.ALIGN_CENTER_HORIZONTAL)
sizeritemimage = sizer.Add(imageBitmap, 0, wx.ALIGN_CENTER)
sizer.Add(button1, 0, wx.ALIGN_CENTER_HORIZONTAL)
sizer.Add(text, 1, wx.EXPAND|wx.ALL, 10)

panel.SetSizer(sizer)

def updateimage(self, angle, state):
    if (state == 1):
        self.imageFile = "/[\Users/m/Desktop/good code/untitled folder/handstop.png"
    elif (state == 2):
        self.imageFile = "/[\Users/m/Desktop/good code/untitled folder/handup.png"
    else:
        self.imageFile = "/[\Users/m/Desktop/good code/untitled folder/handdown.png"

    image = wx.Image( self.imageFile, wx.BITMAP_TYPE_ANY)
    image = wx.Image.Rotate(image, math.radians(angle), wx.Point(0,0))
    x = (image.GetWidth() - 190)/2
    image = wx.Image.Resize(image, size=(190,190), pos=(-x,-x))
    self.imageBitmap.SetBitmap(wx.BitmapFromImage(image))
    self.sizeritemimage.SetFlag(wx.ALIGN_CENTER)
    self.Refresh()
    self.Update()

def textoutput(self, text):
    self.text.Clear()
    self.text.AppendText(text)

def onbutton(self, event):
    ser.flush()
    self.textoutput('Take your hand off the device, and allow it to start in the upward position.

    currentangle = 2130
    ser.write(struct.pack(chr(1), int(currentangle)))
    wx.Yield()

    endtime = time.time() + 0.5
    while(time.time() < endtime):
ser.readline()

self.textoutput('Calibrating... Do NOT touch the device!')
wx.Yield()

dertime = time.time() + 0.5
while(time.time() < endtime):
    ser.readline()

dertime = time.time() + 5
rollingavg = 0
valuescount = 0
uselesscount = 0
degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
print degreesangle
self.updateimage(degreesangle, 1)
# ser.flush()
while(time.time() < endtime):
    # True
    ser.readline()

    if(uselesscount > 20):
        rollingavg = rollingavg + line
        valuescount = valuescount + 1
        uselesscount = uselesscount + 1

    rollingavg = rollingavg/valuescount
    ##print rollingavg
    wx.Yield()

self.textoutput('Calibration complete. You may now use the device. Press gently and slowly increase pressure until the device reacts. Bring the platform to the horizontal point.')
wx.Yield()

# ser.flush()
# tempvar = 2140
while(currentangle > 1450):
    
    # break

    degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
    print degreesangle
    self.updateimage(degreesangle, 3)

    # endtime = time.time() + testdelaytime
    # while(time.time() < endtime):
    #     # True

    # tempvar = tempvar - 1
    # line = "12 " + str(tempvar)
    line = ser.readline()
    data = [float(val) for val in line.split()]
    line = data[0]
    currentangle = data[1]
    #line = 23
    print line, currentangle
    if(line > rollingavg + 20):
        # True
        ser.write(struct.pack('I', int(currentangle-5)))
        #print 'SENDING ', currentangle-20, ' TO THE DEVICE'
    wx.Yield()

endtime = time.time() + 0.5
while(time.time() < endtime):
    # True
    ser.readline()

self.textoutput('Good! Now relieve pressure on the device, but keep your hand on it. It will slowly return to it\'s starting position."
wx.Yield()
# ser.flush()

# tempvar = currentangle

while(currentangle < 2140):
# break

degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
print degreesangle
self.updateimage(degreesangle, 2)

# endtime = time.time() + testdelaytime
# while(time.time() < endtime):
#   # True

# tempvar = tempvar + 1
# line = "12 " + str(tempvar)
line = ser.readline()
data = [float(val) for val in line.split()]
line = data[0]
currentangle = data[1]
#line = 16
print line, currentangle
if(line < rollingavg+15):
  # True
  ser.write(struct.pack('I', int(currentangle+5)))
  #print 'SENDING ', currentangle+20, ' TO THE DEVICE'
wx.Yield()

endtime = time.time() + 0.5
while(time.time() < endtime):
  # True
  ser.readline()

self.textoutput('Good! Now apply more pressure and bring it all the way down. Full flexion!')
wx.Yield()

# ser.flush()

#tempvar = currentangle

while(currentangle > 960):

degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
print degreesangle
self.updateimage(degreesangle, 3)
# endtime = time.time() + testdelaytime
# while(time.time() < endtime):
#   # True

# tempvar = tempvar - 1
# line = "12 " + str(tempvar)
line = ser.readline()
data = [float(val) for val in line.split()]
line = data[0]
currentangle = data[1]
#line = 23
print line, currentangle
if(line > rollingavg + 20):
    # True
    ser.write(struct.pack('I', int(currentangle-5)))
    #print 'SENDING ', currentangle-20, ' TO THE DEVICE'
wx.Yield()

endtime = time.time() + 0.5
while(time.time() < endtime):
    # True
    ser.readline()

self.textoutput('Good! Now relieve pressure on the device, but keep your hand on it. It will slowly return to it\'s starting position.\')
wx.Yield()
# ser.flush()
while(currentangle < 2140):

    degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
    print degreesangle
    self.updateimage(degreesangle, 2)

# endtime = time.time() + testdelaytime
# while(time.time() < endtime):
#   # True

# tempvar = tempvar + 1
# line = "12 " + str(tempvar)
line = ser.readline()
data = [float(val) for val in line.split()]
line = data[0]
currentangle = data[1]
#line = 16
print line, currentangle
if(line < rollingavg+15):
    # True
    ser.write(struct.pack('I', int(currentangle+5)))
    #print 'SENDING ', currentangle+20, ' TO THE DEVICE'
wx.Yield()

eendtime = time.time() + 0.5
while(time.time() < endtime):
    # True
    ser.readline()

self.textoutput('Good job! The exercise test is complete!')

degreesangle = 180-(currentangle - 700 + 50) / (80.0 / 9) - 90
print degreesangle
self.updateimage(degreesangle, 1)
wx.Yield()

def closebutton(self, event):
    ser.close()
    self.Close(True)

def closewindow(self, event):
    self.Destroy()

class windowClass(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Welcome')
        panel=wx.Panel(self)
        sizerA = wx.BoxSizer(wx.VERTICAL)
        welcomeText = wx.StaticText(panel, -1, 'Select from the options below:')
        sizerA.Add(welcomeText, wx.EXPAND, wx.ALIGN_CENTER_HORIZONTAL)
        self.Bind(wx.EVT_CLOSE, self.closewindow)
        button1 = wx.Button(panel, label="Play")
        self.Bind(wx.EVT_BUTTON, self.playbutton, button1)
sizerA.Add(button1, wx.EXPAND, wx.ALIGN_CENTER_HORIZONTAL)
button=wx.Button(panel, label="Exit")
self.Bind(wx.EVT_BUTTON, self.closebutton, button)
sizerA.Add(button, wx.EXPAND, wx.ALIGN_CENTER_HORIZONTAL)
panel.SetSizer(sizerA)

def closebutton(self, event):
    self.Close(True)

def closewindow(self, event):
    self.Destroy()

def playbutton(self, event):
    frame = windowClassInner(parent=None, id=-1)
    frame.Show()

#########################
ser = serial.Serial('/dev/cu.usbmodem1421', 115200, timeout=1)
ser.readline()
# testdelaytime = 0.005

def main():
    app = wx.App()
    frame = windowClass(parent=None, id=-1)
    frame.Show()
    app.MainLoop()

if __name__ == "__main__":
main()


2. Plotting

import serial
import struct
import sys
import numpy as np
from time import sleep
from collections import deque
from matplotlib import pyplot as plt

ser = serial.Serial('/dev/cu.usbmodem1421', 115200, timeout=1)

class AnalogData:
    # constr
    def __init__(self, maxLen):
        #self.ax = deque([0.0]*maxLen)
        self.ay = deque([0.0]*maxLen)
        self.maxLen = maxLen

        # ring buffer
        def addToBuf(self, buf, val):
            if len(buf) < self.maxLen:
                buf.append(val)
            else:
                buf.pop()
                buf.appendleft(val)

        # add data
        def add(self, data):
            assert(len(data) == 2)
            #self.addToBuf(self.ax, data[0])
            self.addToBuf(self.ay, data[0])

    # plot class
    class AnalogPlot:
        # constr
        def __init__(self, analogData):
            # set plot to animated
            plt.ion()
            self.axline, = plt.plot(analogData.ax)
            self.ayline, = plt.plot(analogData.ay)
plt.ylim([0, 1023])

# update plot
def update(self, analogData):
    #self.axline.set_ydata(analogData.ax)
    self.ayline.set_ydata(analogData.ay)
    plt.draw()

analogData = AnalogData(100)
analogPlot = AnalogPlot(analogData)

def plotter(ser):
    # expects 1 arg - serial port string
    #if(len(sys.argv) != 2):
    #    print 'Example usage: python showdata.py "/dev/tty.usbmodem411"
    #    exit(1)

    #strPort = '/dev/cu.usbmodem1411'
    #strPort = sys.argv[1]

    # plot parameters
    #analogData = AnalogData(100)
    #analogPlot = AnalogPlot(analogData)

    print 'Plotting data...\nPress CTRL-C to pause'

    # open serial port
    #ser = serial.Serial(strPort, 9600)
    while True:
        try:
            line = ser.readline()
            #print line
            data = [float(val) for val in line.split()]
            #print data
            if(len(data) == 2):
                analogData.add(data)
                analogPlot.update(analogData)
        except KeyboardInterrupt:
            print ' Pausing the Plot'
            break
    # close serial
    ser.flush()
#plt.close()
#ser.close()

def changeAngle():
    val = int(input('Enter Angle in Degrees. Flat is 0, range is -56 to 78: '))
    if(val > 78 or val < -56):
        print 'Angle is out of range! Enter a new angle.'
    else:
        #convert angle to pulse width value and send to arduino
        val = (val + 90)*(80.0/9)+700-50
        #print val
        ser.write(struct.pack('I', int(val)))

def readData():
    while(True):
        line = ser.readline()
        print line

while(True):
    mode = input('Type 1 to change the angle, type 2 to read data, type 3 to leave the program: ')
    if(mode == 1):
        changeAngle()
    elif(mode == 2):
        plotter(ser)
    elif(mode == 3):
        exit()
        ser.flush()
        ser.close()
### Materials List

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
<th>Total Price ($)</th>
<th>Count Used</th>
<th>Used Price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1/4&quot; wood screws</td>
<td>10</td>
<td>1.18</td>
<td>4</td>
<td>0.47</td>
</tr>
<tr>
<td>1-1/2&quot; wood screws</td>
<td>10</td>
<td>1.18</td>
<td>8</td>
<td>0.94</td>
</tr>
<tr>
<td>3/8&quot; machine screws</td>
<td>14</td>
<td>1.18</td>
<td>4</td>
<td>0.34</td>
</tr>
<tr>
<td>Flat washers</td>
<td>36</td>
<td>1.18</td>
<td>18</td>
<td>0.59</td>
</tr>
<tr>
<td>2-1/2&quot; wood screws</td>
<td>8</td>
<td>1.18</td>
<td>2</td>
<td>0.30</td>
</tr>
<tr>
<td>3/4&quot; wood screws</td>
<td>18</td>
<td>1.18</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Hinges (with screws)</td>
<td>2</td>
<td>2.57</td>
<td>2</td>
<td>2.57</td>
</tr>
<tr>
<td>2&quot;x4&quot; lumber</td>
<td>96 inches</td>
<td>2.74</td>
<td>12 inches</td>
<td>0.34</td>
</tr>
<tr>
<td>1/4&quot;x4&quot; poplar board</td>
<td>24 inches</td>
<td>1.94</td>
<td>7 inches</td>
<td>0.58</td>
</tr>
<tr>
<td>1/2&quot;x4&quot; pine board</td>
<td>24 inches</td>
<td>2.56</td>
<td>24 inches</td>
<td>2.56</td>
</tr>
<tr>
<td>Servo and gearbox combo</td>
<td>1</td>
<td>149.98</td>
<td>1</td>
<td>149.98</td>
</tr>
<tr>
<td>3&quot; single aluminum servo arm</td>
<td>1</td>
<td>10.99</td>
<td>1</td>
<td>10.99</td>
</tr>
<tr>
<td>5V voltage regulator</td>
<td>1</td>
<td>1.25</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Arduino Pro Micro</td>
<td>1</td>
<td>19.95</td>
<td>1</td>
<td>19.95</td>
</tr>
<tr>
<td>50kg load sensors</td>
<td>4</td>
<td>39.80</td>
<td>4</td>
<td>39.80</td>
</tr>
<tr>
<td>INA125P opamps</td>
<td>4</td>
<td>25.76</td>
<td>2</td>
<td>12.88</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>264.62</strong></td>
<td></td>
<td><strong>242.29</strong></td>
</tr>
</tbody>
</table>