Analytical Analysis: PID Control

Active Prosthesis

NORTHERN ARIZONA UNIVERSITY

College of Engineering, Forestry & Natural Sciences

Felicity Escarzaga
Team 12 Active Prosthetic
ME 476C-5 Mechanical Engineering Design
March 1, 2019

1 Introduction

This analytical analysis looks at running a PID to control a servo motor with a force-sensitive resistor (force resistance sensor) using an Arduino microcontroller. One of the main goals of the Active Prosthesis project is to allow the user to actively control the finger movements. To do this, the user must be able to send signals to the device to tell the device which fingers to move and how much to move them. It was decided that to make this as easy as possible for the user, the device would control motors connected to the fingers using a force sensor. To relate the sensors input to the motor output, a PID was determined to be the best means of control. This would allow the sensor to use an analog input to produce a precise position of the motor relative to the force applied. Below discusses the hardware, code, and results in detail.

2 Hardware

The hardware consists of an Arduino UNO microcontroller, a MG 996R Tower Pro servo motor, a force-sensitive resistor (sensor), three 3.3K Ω resistors, and a breadboard. All components are labeled in *Figure 2.1*.

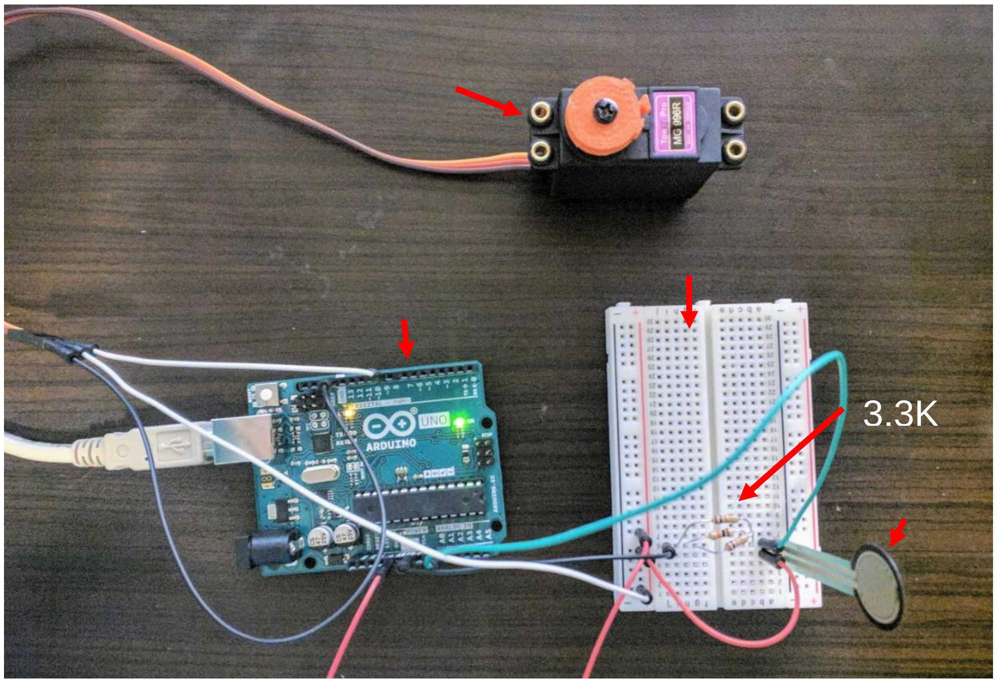


Figure 2.1: Hardware connected.

The force sensor was plug directly into two lines of the breadboard. One line of the pressure sensor is connected to the A0 pin on the Arduino. The three resistors are set in parallel from the same line to the ground pin of the Arduino. Power from the 5V pin is supplied directly to the second line of the sensor and power line of the motor. The motors ground is connected directly to the Arduino ground pin and does not need resistors between them. Finally, the motor's inner potentiometer is connected to the PWM 9 pin on the Arduino. Each component and their function is briefly described below.

2.1 Arduino UNO

The Arduino UNO is an open source microcontroller platform. User can program the arduino using their own code with the assistance of hundreds of available libraries. Libraries are created by the Arduino company, companies that make arduino related products, and by other hobbyists and users. The Arduino UNO was chosen for this analysis because it is the microcontroller the prosthetic will be using in the future, is easy to use, and has many available resources for novice coders. The Arduino's function in this analysis will be to run and process the code, motor, and sensors. It will also display the results and data using the integrated development environment (IDE).

2.1.1 IDE

The IDE is the software component of Arduino. This is where the libraries and code can be created, found, used, and uploaded to the Arduino boards. All coding for this analysis will take place in the Arduino IDE software.

2.2 Servo Motor

Servo motors are DC motors with built in potentiometers that track the position of the motor. The MG 996R Tower Pro servo has can rotate 270° and provides a torque of 13 lb-inches. With this torque and degree of rotation the device should produce 1.6 lb of force at the tip of the finger. Though this is only half the 3 lb force desired, this motors are relatively affordable, light, and small compared to other higher torque motors.

2.3 Force-Sensitive Resistor

The force-sensitive resistor is also known as a force sensor, force resistance sensor, or pressure sensor. This sensor increases resistance when a pressure or force is applied over the area of the sensor. An arduino can read the resistance supplied by the sensor as an analog input and correlate that input to the motor's potentiometer. This sensor was chosen for its small design and easy of use. These sensors can be easily connected to an insole on a shoe and will impose little to no discomfort.

2.3.1 Resistors

Resistance from the force-sensitive resistor cannot be directly connected to the ground pin on the Arduino because the voltage is too high from the 5V volt pin. The high voltage and low resistance from the sensor causes the change in current to be too small to read from the Arduino. To decrease the current and increase the change in the current, a 10K Ω is needed. In this Analysis 3K Ω resistors are used in parallel to equal a 10K Ω resistance.

3 Code

While the hardware is needed to receive results, this analysis is about using PID to control the hardware. The PID is programmed through the code run by the Arduino IDE software. This section focuses on each component of the code and what is does for that section of the total function of the code.

3.1 Libraries

Libraries are preprogrammed functions installed into the Arduino IDE. The libraries used in this code are `PID_v1` and `Servo`; these libraries were created by Brett Beauregard a hobbyist and Michael Margolis an Arduino associate respectively.

```
#include <PID_v1.h>
```

The PID library includes future functions used such as `.SetMode`, `.SetOutputLimits`, `.SetSampleTime`, and `.Compute`. All functions are predefined and do not need to be edited.

```
#include <Servo.h>
```

The Servo library includes future functions used such as `.read` and `.write`. Like the PID library it does not need to be edited.

3.2 Variables

Variables are place holders that must be initialized and defined. There are many ways to define a variable such as double, float, int, long, and so on. The variables used here are defined as double, long, and int.

```
double Setpoint, Input, Output;
double Kp = 2, Ki = 5, Kd = 0.5;
```

A `double` holds 4 bytes of information and allows 6-7 digits of precision. The setpoint, input, and output values must be precise since these relate to the force on the sensor, position on the motor and the difference between the two. Kp, Ki, and Kd which are the proportional, integral, and derivative multipliers, must also be a double since they are used in computing the output. Since the output is defined as a double, it is easier to define all variables as doubles that will be used in the output instead of converting them during computing.
The difference between the sensor input range is 0 to 1023 however the motor potentiometer uses a range of 0 to 255. To relate the sensor input to motor position it was found that the quickest and most precise way to get a ratio between the motor position and sensor force was with a proportional multiplier of 2 an integral multiplier of 5 and a derivative multiplier of 0.5 to the setpoint.

```
long oldPos = 0;
```

A `long` variable is also 4 bytes and is used to compute integers. Since the difference between the new position of the motor and the old position of the motor will only ever be an integer and the positions will not be used to calculate in the PID. It makes the most sense to define the oldPos and newPos as long instead of a double.

3.2.1 Objects

```
Servo servo;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

Objects are much like variables in that they are placeholders but for functions instead of point data. Objects are defined by the library they are calling. The object servo calls the servo library and is used before a function is called such as `servo.read`. The `myPID` object calls the PID library and defines the variables that will be used inside the function. The variables are `Input`, `Output`, `Setpoint`, `Kp`, `Ki`, and `kd`. The `DIRECT` command tells the PID that if the output is above the setpoint then it should be decreased to match the setpoint and that the reverse would also be true until the output and setpoint are the same.

3.4 Setup

```
void setup() {
```

There are many parts in the setup but each part only runs once when the code is uploaded or when the arduino is reset.

```
 Serial.begin(9600);
```

Serial is built in function of the Arduino IDE and calls the serial library which does not have to be installed or defined as included. The serial is set to a baud rate that tells the serial monitor and Arduino to talk to each other when the arduino is connected. It also start the data collection process.

```
 pinMode(A0, INPUT);
 pinMode(9, OUTPUT);
```

The pinMode defines the type of pin as either an input or an output. Here `A0` is defined as an input because the Arduino will be receiving signals from the force sensor through the analog zero pin. Pin `9` is a digital Pulse Width Modulation (PWM) pin. This means it can only receive high (1) or low (0) signals but the PWM allows it to oscillate at a frequency between high and low to create the equivalent analog signal desired. Pin `9` has been defined as an output because it is connected to the motors potentiometer and will be used to control the desired position of the motor.

```
 Input = analogRead(A0);
 Setpoint = 0;
```

The variable `Input` is reading the analog pin `A0` from the sensor. The `Setpoint` starts at zero because when the motor is turned on it should start at its zero position.

```
 myPID.SetMode(AUTOMATIC);
 myPID.SetOutputLimits(0, 1023);
 myPID.SetSampleTime(60);
```

The rest of the setup is defining the PID functions. `.SetMode(AUTOMATIC)` is telling the PID to run automatically when called. `.SetOutputLimits(0, 1023)` the range of the output signal is set between 0 and 1023. `.SetSampleTime(60)` is used to start the PID clock function which is used to calculate the integral and derivative multipliers.

3.5 Loop

```
void loop() {
```

Loop functions run continuously while the Arduino is powered. Here, any functions that must be run repeatedly are programmed or called.

```
 long newPos = analogRead(A0);
```

`newPos` reads the sensor input and is defined locally because it is not used in any other function. There are 32256 bytes available for local storage and only 2048 bytes available for global; so it is best to define locally as much as possible.

```
 if (newPos != oldPos) {
```

`if` functions only run if parameters are met. Here, the function will only run if the new force of the sensor does not match the old force. This makes the Arduino only tell the motor to move if the sensor input has changed.

```
   oldPos = newPos;
```

During this time the new force now becomes the old force.

```
   Input = analogRead(A0);
```

Now the `Input` also reads the sensor. This is done as a separate variable because it will be used in the PID calculation, must be defined as a double, and should not be changed during calculation.

```
   myPID.Compute();
```

Here the PID is finally computed so that the output can be found.

```
   Serial.print(Input);
   Serial.println(Output);
```

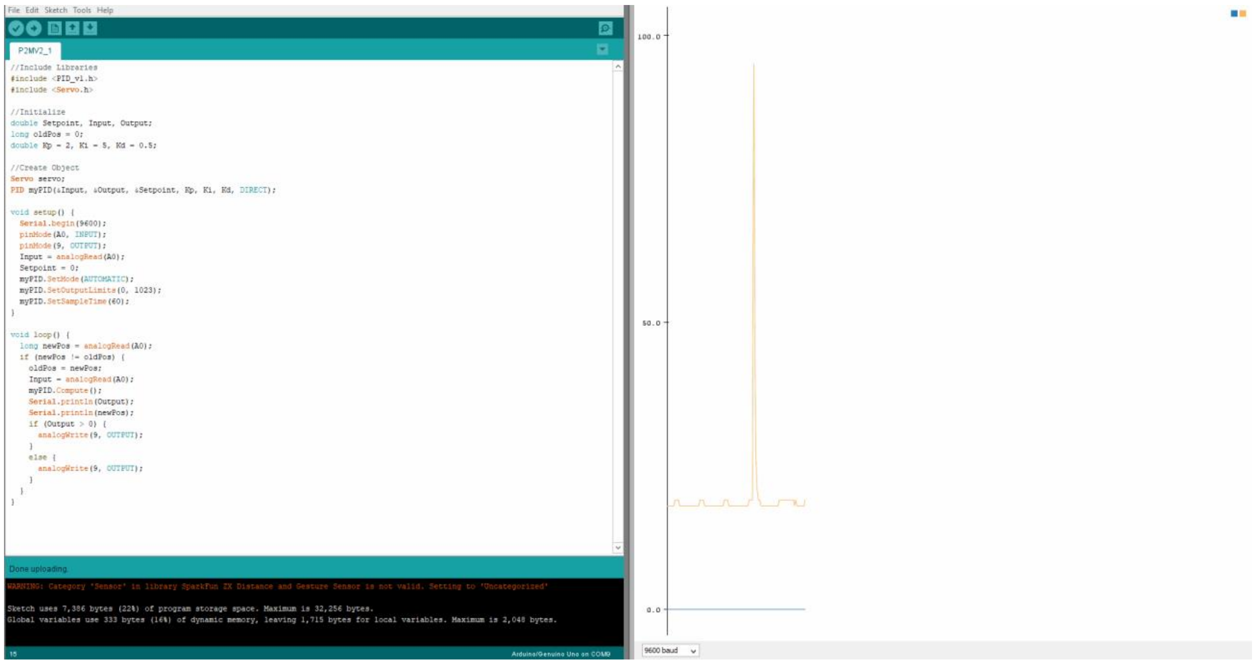
Both the input and outputs are recorded and displayed in the serial monitor as shown in *Figure 4.1* using `Serial.print`. The ln at the end of the second serial print moves the data to a new line once it runs.

```
     digitalWrite(9, OUTPUT);
```

Finally the motor is set to the output value using a digital write function since it is a digital pin and the code is repeated in the loop.

4 Results

The results showed that the motor quickly and precisely followed the sensor input. When the force applied to the sensor increased the motor's position increased. When the sensor was held constant the motor held its position. This can be shown in *Figure 4.1*.



```
//Include Libraries
#include <PID_v1.h>
#include <Servo.h>

//Initialize
double Setpoint, Input, Output;
long oldPos = 0;
double Kp = 2, Ki = 5, Kd = 0.5;

//Create Object
Servo servo;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
  Serial.begin(9600);
  pinMode(A0, INPUT);
  pinMode(9, OUTPUT);
  Input = analogRead(A0);
  Setpoint = 0;
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(0, 1023);
  myPID.SetSampleTime(60);
}

void loop() {
  long newPos = analogRead(A0);
  if (newPos != oldPos) {
    oldPos = newPos;
    Input = analogRead(A0);
    myPID.Compute();
    Serial.println(Output);
    Serial.println(newPos);
    if (Output > 0) {
      analogWrite(9, OUTPUT);
    }
    else {
      analogWrite(9, OUTPUT);
    }
  }
}
```

```
Done uploading.
WARNING: Category 'Sensor' in library SparkFun ZX Distance and Gesture Sensor is not valid. Setting to 'Uncategorized'

Sketch uses 7,386 bytes (22%) of program storage space. Maximum is 32,256 bytes.
Global variables use 333 bytes (16%) of dynamic memory, leaving 1,715 bytes for local variables. Maximum is 2,048 bytes.
15                                                            Arduino/Genuino Uno on COM9
```

Figure 4.1: Code running. (Gif will be uploaded with file. It may not run in document.)

The sensor was more sensitive than originally thought. Even the slightest release of pressure and the sensor reading dropped, in turn quickly changing the position of the motor. This may make the grip of the prosthetic unsteady unlessed used at full force. The graph in *Figure 4.2* shows the data of the sensor and the motor in blue and red respectively and it is clear the shakiness of the motor dependent on the sensor.

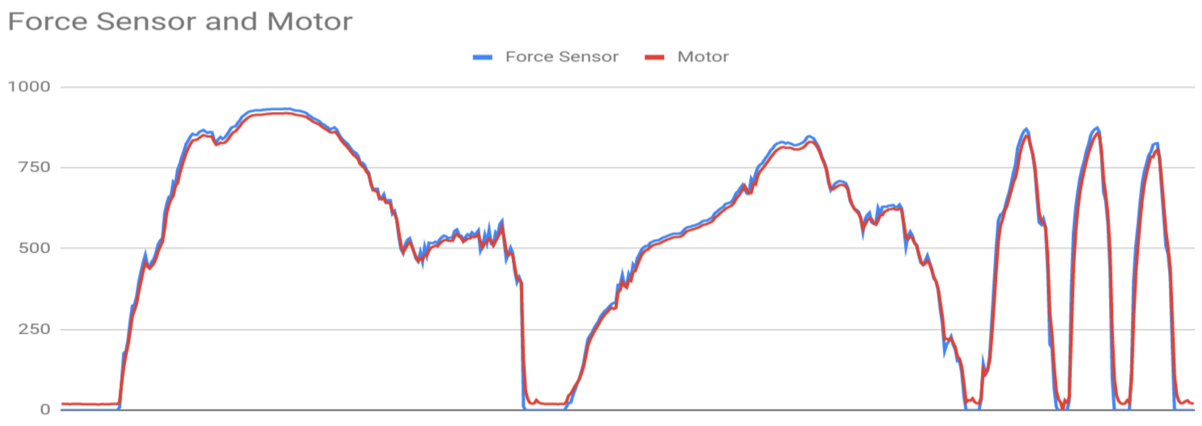


Figure 4.2: Data - Motor follows Sensor

Though the sensor and motor may be very sensitivity, with practice this could be a good thing. It does not appear that the shaking is due to interference but instead due to unsteady pressure. If the client has some practice they may be able to very precisely control the position of the fingers and will have instant responsiveness from the motors.

5 References

"Arduino," *Arduino*. [Online]. Available: https://www.arduino.cc/. [Accessed: 28-Feb-2019].

6 Appendix

6.1 Code

```
//Include Libraries
#include <PID_v1.h>
#include <Servo.h>

//Initialize
double Setpoint, Input, Output;
long oldPos = 0;
double Kp = 2, Ki = 5, Kd = 0.5;

//Create Object
Servo servo;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
 Serial.begin(9600);
 pinMode(A0, INPUT);
 pinMode(9, OUTPUT);
 Input = analogRead(A0);
 Setpoint = 0;
 myPID.SetMode(AUTOMATIC);
 myPID.SetOutputLimits(0, 1023);
 myPID.SetSampleTime(60);
}

void loop() {
 long newPos = analogRead(A0);
 if (newPos != oldPos) {
   oldPos = newPos;
   Input = analogRead(A0);
   myPID.Compute();
   Serial.print(Input);
   Serial.println(Output);
   digitalWrite(9, OUTPUT);
 }
}
```