Analytical Analysis: Software and Hardware

Active Prosthesis

Felicity Escarzaga

Team 18F-12 Active Prosthetic

ME 476C-5 Mechanical Engineering Design

November 9, 2018

1 Introduction

This report covers the hardware and software that will be used for the Active Prosthesis project. For a prosthesis to be active, it must use some electrical components actuate. In this project, these components must allow the prosthesis to be able to sense objects, activate when desired, and provide haptic feedback. Therefore, an analytical analysis of the available hardware and coding for the software will need to be performed. Since the final inputs and outputs are still to be determined, the code will take distance sensor readings and use those readings to actuate a motor. The readings will represent the inputs and the motor will represent the outputs for the final design.

2 Hardware

The hardware used is important to determining the capabilities of the prosthetic. If the hardware cannot perform the function to meet the customer requirements well, then better hardware and further prototyping is needed. To reduce unnecessary spending on hardware prototyping, a comparison of the hardware can be used to determine the best equipment properties and compatibility. Compatibility is one of the important aspect of this analysis. The system will not operate if the equipment is not compatible with all hardware included, or if some hardware requires the same pins.

2.1 Distance Sensor

Three distance sensors were compared from three different companies Sparkfun, Amazon, and Polulu [1-3]. All sensors collected to meet a distance range from 0.5 in to 6 in, which is approximately the distance needed to determine if the prosthetic is reaching to grab an object. **Table 2.1** compares the important properties of each sensor which includes range, voltage needed, type, pins needed, and cost.

Table 2.1- Distance Sensors

Distance Sensor	Range	Voltage	Type	Pins Needed	Cost
ZX Distance and Gesture Sensor	0 - 12 in	3.3V - 5V	Laser	5	24.95
Elegoo HC-SR04	0.78 - 157 in	5 V	Sound	4	9.78
Pololu Carrier with Sharp GP2Y0D815Z0F Digital Distance Sensor 15cm	0.2 - 6 in	5 V	Laser	6	9.75

From this table it appears the best sensor to choose is the Elegoo sensor because it has the largest range and the lowest price. However, the Elegoo has an uncertainty of ±3 in. The

uncertainty for the other two sensors were not listed but can be assumed to be lower due to their smaller range and the type of sensor used. The Elegoo uses a sonic type sensor while the other sensors use optical laser sensors. Polulu is the cheapest option but requires six pins. The ZX sensor is the most expensive but requires only five pins.
It was determined that the ZX sensor would be the best sensor option since it has a range including zero. At zero range the prosthetic could receive information the it is not touch the object. Furthermore, since it is also a guestor sense it could be used in the future to determine user intention.

2.2 Motor

The three motors listed in **Table 2.2** are from the same companies listed in section 2.1. Each motor was evaluated using the properties: input voltage, current required, speed, shaft size, and cost. All properties are important, however, shaft size is not important until further in the design when the attachment is determined.

Table 2.2 - Motor

Motor	Input Voltage	Amps	Speed	Shaft	Torque	Cost
URBEST	12V	0.6 A	300 RPM	3 mm /0.118"	7 oz-in	11.99
131:1 Metal Gearmotor 37Dx57L mm	12V / 6V	300 mA	80 RPM	6 mm	250 oz-in	24.95
Stepper Motor	3.2 V	2.0 A	200 SPR	6.35 mm	125 oz-in	30.95

Voltage required is similar for the first two motors but drops for the stepper motor. The stepper motor provides the best torque output for the lowest voltage but it is the most expensive and the voltage drop is balanced by a current increase. The stepper would allow for control of fine movements at 200 steps per revolution, but will increase the overall cost for the motor and the motor driver with the current increase. The URBEST is the least expensive but has the lowest torque and would not produce the required force. The 131:1 Metal Gearmotor is therefore the best option for the prosthesis.

2.3 Motor Driver

In **Table 2.3**, the motor drivers are compared by the number of motors they can operate at the same time, the current that can be ran per channel, whether an additional power supply (other than the microcontroller vin) can be added, the shield compatibility, and the cost. An additional company's board was considered from adafruit.

Table 2.3 - Motor Drivers

Motor Driver	Number of Motors	Amps/Channel	Additional Power Supply	Shield Compatible	Cost
SparkFun Ardumoto	2	2 A	no	R3	20.95
SparkFun Wireless Motor Driver Shield	2	1.2 A	yes	R3, Xbee	26.95
Pololu Dual VNH5019 Motor Driver Shield for Arduino	2	12 A	yes	R3	49.95
Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit - v2.3	4	1.2 A	yes	R3	19.95

It can be seen from the table that the best driver is the Adafruit motor shield. It is compatible with all R3 arduino microcontrollers, produces enough current per channel to run four of the motors selected, has additional power supply input available to power four motors, and it has the lowest cost.

2.4 Arduino

Since Arduino is open source and available around the world, these microcontrollers were chosen and compared amongst each other in **Table 2.4**. The controllers would need to meet the previous hardware requirements from the components selected above and be able to accommodate possibly multiple sensors.

Table 2.4 - Arduino Boards

Microcontroller	Attach-Interrupt Pins	Operating/ Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	Serial Read Pins	Shield Compatible	Cost
Mega 2560	2, 3, 18, 19, 20, 21	5 V / 7-12 V	16 MHz	16/0	54/15	3	R3	38.5
Micro	0, 1, 2, 3, 7	5 V / 7-12 V	16 MHz	12/0	20/7	1	N/A	19.8
Uno	2, 3	5 V / 7-12 V	16 MHz	6/0	14/6	1	R3	22
Zero	all digital pins, except 4	3.3 V / 7-12 V	48 MHz	6/1	14/10	1	R3	42.9
Due	all digital pins	3.3 V / 7-12 V	84 MHz	12/2	54/12	3	R3	35.5

SparkFun RedBoard	2, 3	3.3 V / 7-15 V	16 MHz	6/0	14/6	1	R3	19.95

The Due was selected as the optimal microcontroller due to the number of digital, analog, and attach-interrupt pins required to run multiple sensors and multiple motors. It also has extra serial read pins that can be used to communicate wirelessly to other modules if needed in future design. Finally, it is the cheapest option that meets all requirements and has processing speed allowing for quicker response time from the prosthesis.

3 Code

Using the hardware chosen the software and code can be written. Most arduino compatible sensors and shields have libraries that are downloadable to the arduino IDE. These libraries allow for easier coding of the components used and have some demo codes to show the functions available from the library. Using these libraries and functions available, a validated code can be generated run a motor using the sensor selected. The entire code can be found in **Appendix 6.1**.

4 Results

The hardware chosen from this analysis is ZX sensor, Polulu motor, adafruit motor shield, and arduino due. All the hardware is compatible with the microcontroller chosen and allows for wireless connectivity, additional motors, extra sensors, and other future design modifications. The code available in **appendix 6.1** does run the motor as expected, however the distance sensor could not be tested since it was not available before the deadline for the analysis.

5 References

[1] "Pololu Robotics & Electronics," Pololu, 2018. [Online]. Available: https://www.pololu.com/. [Accessed 3 11 2018].

[2] "SparkFun," Sparkfun, 2018. [Online]. Available: https://www.sparkfun.com/. [Accessed 31 10 2018].

[3] "Amazon," Amazon, 2018. [Online]. Available: https://www.amazon.com/. [Accessed 3 11 2018].

[4] "Adafruit," Adafruit, 2018. [Online]. Available: https://www.adafruit.com/. [Accessed 14 11 2018].

6 Appendix

6.1 Code

```
//Library
#include <Wire.h>
#include <ZX_Sensor.h>
#include <Adafruit_MotorShield.h>

// Create motor shield object
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Select motor
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);

// Constants
const int ZX_ADDR = 0x10;  // ZX Sensor I2C address

// Global Variables
ZX_Sensor zx_sensor = ZX_Sensor(ZX_ADDR);
uint8_t x_pos;
uint8_t z_pos;
uint8_t z_posnew = 0;
uint8_t z_posold = 0 ;
uint8_t dz_pos = 0;

void setup() {

  uint8_t ver;

  // Initialize Serial port
  Serial.begin(9600);

  // Initialize ZX Sensor (configure I2C and read model ID)
  if ( zx_sensor.init() ) {
    Serial.println("ZX Sensor initialization complete");
  } else {
    Serial.println("ZX Sensor initialization incomplete!");
  }

  // Read the model version number and ensure the library will work
  ver = zx_sensor.getModelVersion();
  if ( ver == ZX_ERROR ) {
```

```
    Serial.println("Error reading model version number");
  } else {
    Serial.print("Model version: ");
    Serial.println(ver);
  }
  if ( ver != ZX_MODEL_VER ) {
    Serial.print("Model version needs to be ");
    Serial.print(ZX_MODEL_VER);
    Serial.print(" to work with this library. Stopping.");
    while (1);
  }

  // Read the register map version and ensure the library will work
  ver = zx_sensor.getRegMapVersion();
  if ( ver == ZX_ERROR ) {
    Serial.println("Error reading register map version number");
  } else {
    Serial.print("Register Map Version: ");
    Serial.println(ver);
  }
  if ( ver != ZX_REG_MAP_VER ) {
    Serial.print("Register map version needs to be ");
    Serial.print(ZX_REG_MAP_VER);
    Serial.print(" to work with this library. Stopping.");
    while (1);
  }
}

void loop() {
  // If there is position data available, read and print it
  if ( zx_sensor.positionAvailable() ) {
    z_posnew = zx_sensor.readZ();
    dz_pos = z_posold - z_posnew;
  }
  uint8_t i;
  Serial.print("tick");
  if (abs(dz_pos) > 0) {
    if (dz_pos > 10) {
      myMotor->run(FORWARD);
      myMotor->setSpeed(150);
      delay(10);
    }
    if (dz_pos < -10) {
      myMotor->run(BACKWARD);
```

```
      myMotor->setSpeed(150);
      delay(10);
    }
  }
  if ( zx_sensor.positionAvailable()  ) {
    z_pos = zx_sensor.readZ();
    if ( z_pos != ZX_ERROR ) {
      Serial.print(" Z: ");
      Serial.println(z_pos);
    }
  }
  Serial.print("tock");

  z_posold  = z_posnew;
  Serial.print("tech");
  myMotor->run(RELEASE);
  delay(1000);
}
```