EMG-Based Human Machine Interface System

Senior Project Final Report

By: Jonathan Morón, Thomas DiProva, and John Reaser Cochrane

Advisors: Drs. Yufeng Lu and In Soo Ahn

Department of Electrical and Computer Engineering
Bradley University
05/02/2017

ABSTRACT

Electromyogram (EMG) signals are generated by muscles when activated by the nervous system, which generates electric potential. These signals then cause contraction of the muscle which results in body movement. These EMG signals can be read through the use of surface EMG electrodes (sEMG) to assess the muscle to read the signals from the skin above the desired muscle. EMG signals can be useful for developing systems that can help the disabled. This assistance can be performed through the development of prosthetics. The medical field has benefited from intensive research in EMG signal analysis in the past couple decades, which has improved the quality of life for those with psychomotor skill disabilities.

The project aimed to design and develop a wearable human machine interface device for an in home assistance service robot. The system consists of three main subsystems a EMG signal extraction and classification subsystem, a robot pwm motor controlled service robot, and a live video feedback for user control. The EMG control subsystem utilized a MyoWare muscle sensor board, to amplify the EMG signal, and a Particle Photon microcontroller to implement a artificial neural network (ANN) to classify the EMG signals. The Particle Photon transmitted the commands through WiFi to the Raspberry Pi which then generated PWM specific commands for motor control. The Raspberry Pi 3 also hosted a website which was used for robot navigation and user instructions.

ACKNOWLEDGEMENTS

We would like to thank our advisors, Drs. Yufeng Lu and In Soo Ahn, for their guidance and advice throughout the senior capstone project. Without their extensive knowledge in communication and signal analysis this project would not be possible. We would like to express our sincere gratitude and appreciation for the ECE department faculty on their countless hours of help and guidance throughout our undergraduate careers. Finally we would like to thank our family and friends for their support in the project's entirety.

TABLE OF CONTENTS

LIST OF FIGURES

1. BACKGROUND

1.1. Related Work

In 2015, Harvard designed an assisted motion glove to give grip strength and lifting support to humans with hand impairments. [1] The soft glove design is made to be wearable and lightweight for the user. The glove would detect weak EMG specific signal locations to move the gears on the robot in a specific way to model a real human grip. sEMG sensors were implemented to better suit each individual user.

An EMG Human-Machine interface system was designed by researchers to control a robot by reading and analyzing EMG signals produced by eye movement. Feature detections where applied by analyzing the voltage threshold of the signal. The EMG system implemented a path planning algorithm which classifies certain eye movements with approximately a 95.71% accuracy. This system was able to use eye movements to move the robot platform.

Most biomedical field applications obtain and read these signals through the use of placing surface adhesive pads on the skin of the patient. The alternative to this method, is reading the signals intravenously by the injecting a needle into the muscle of the body to read these signals directly. These applications can benefit those with muscle and nerve damage.

1.2 EMG Signals

Electromyogram (EMG) signals are electric potential generated by muscle cells after being activated by the nervous system. The more force we apply to the muscle the greater the action potential, generating a larger electric potential from the muscle cells. In theory, anywhere from 10 Hz to 500 Hz is the typical range of any EMG signal that will be produced by the human body. A raw EMG signal will have a range of -5 to +5 mV before amplification. This value may seem small in nature, but after amplification of the signal can produce very valuable data to researchers.

In most medical and engineering applications, the detection of these signals will be done through surface EMG sensors. They are called surface EMG signals because the material that will be used to detect these signals are placed by an adhesive electrode on the surface of the skin. They obtain useful data on the time and intensity of muscle activation with advantages of being quick and noninvasive to apply. Disadvantages to using a surface emg sensor is the amount of error accumulated from skin, fat, and cross-talk between muscles. The use of medical equipment and human machine interface have been revolutionized by EMG signals, allowing researchers even further understanding of the human body. Even though these signals are hard to classify due to the almost random nature of the action potential, patterns or probability classification can be

implemented to classify the signals. Most medical field applications are called Myoelectric control systems and are predominately used with upper-limb prostheses and electric-powered wheelchairs.

1.3 Neural Networks

Artificial Neural Networks are a computational model that is based off human brain thought process. Neural Networks utilize artificial neurons that link with many other neurons to activate or inhibit the specific output neurons of a system. This system is closely related to a simplified version of how the human brain's axons and synapses work together. An example of an single neuron artificial network diagram is shown in Figure 1.

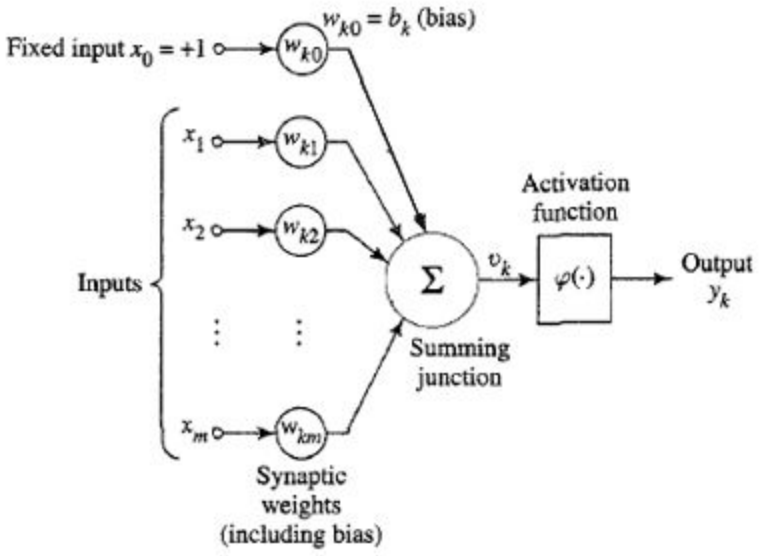

Figure 1. Single Neuron System

Most artificial neural networks include multiple layers consisting of one input layer, single or multiple hidden layers, and one output layer. The input layer consists of the data inputs into the neural network. The hidden layers contain a summation function that takes into account the past layer's output and the weight for the new layer. This is shown in figure 1 as summation (1) over all inputs of (X_m) multiplied by the weights (W_{km}) corresponding to the respective input. These weights and bias are typically found through system learning using training sets of data, which the desired output is already known. When the optimal weights and bias are determined, the neuron pre-activated outputs are processed through an activation function to determine the activated output.

$$v_m = \sum^{k} w_{km} z_k + b_m \tag{1}$$

2. PROJECT DESCRIPTION

The system includes an EMG control, wheeled service robot and monitor subsystems. A MyoWare muscle sensor board was used to read EMG signals from the user that were then transmitted to the Particle Photon's analog to digital converter for advanced signal processing. The Photon then calibrated the system for the user which classified four different commands through the implementation of an artificial neural network (ANN). Upon completion of the classification and calibration of signals the user was then able to send one of four commands to move the robot through WiFi. The four hand and finger commands that the user had to calibrate were for forward, stop, left and right motor control commands.

The wheeled service robot would receive one of four commands through a TCP/IP protocol that was established by the Particle. The Raspberry Pi would then read the command and generate specific pulse width modulation (PWM) signals. Simultaneously, while the Raspberry Pi moved the robot, it also established a web server that displayed the four commands that the user used along with a live video feedback of the robot's path on a computer monitor.

3. SYSTEM DESIGN AND IMPLEMENTATION

System Block Diagram

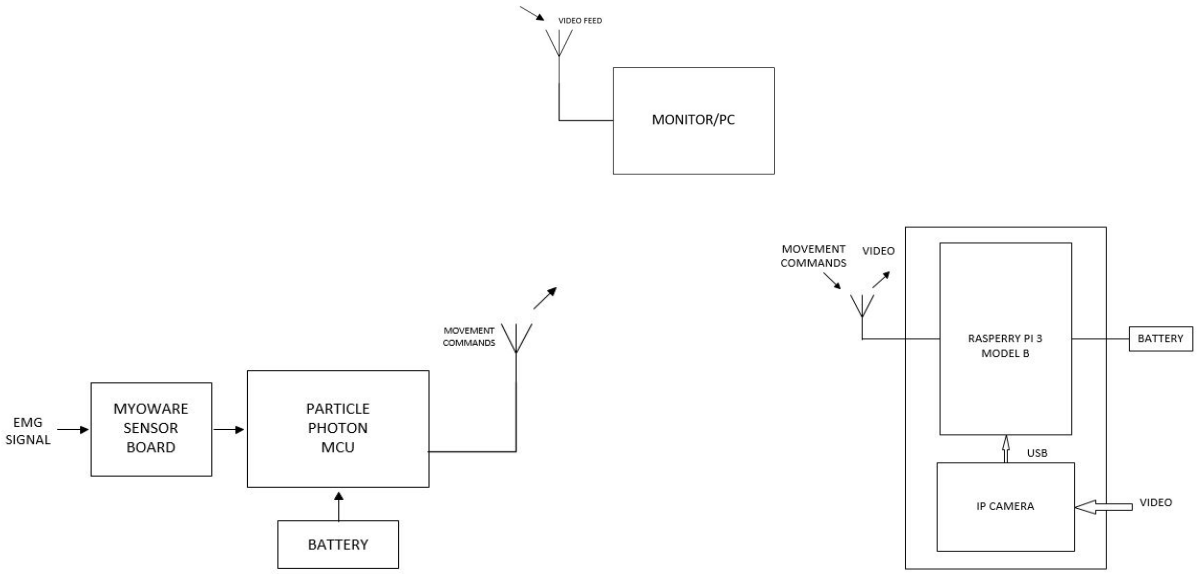

Figure 2. System Block Diagram

3.1 EMG Control Subsystem

The EMG control subsystem utilized a MyoWare muscle sensor board and Particle Photon microcontroller for signal analysis, classification, and communication. The MyoWare muscle sensor board shown in Figure 4 used electrodes placed over the skin on the brachioradialis muscle to detect electric potential of muscle contractions. The sensor board amplified, rectified, and integrated the signal to provide a clean signal for the particle photon's analog-to-digital converter. Figure 4 shows the how the MyoWare muscle sensor board processed the signal for the analog to digital converter.

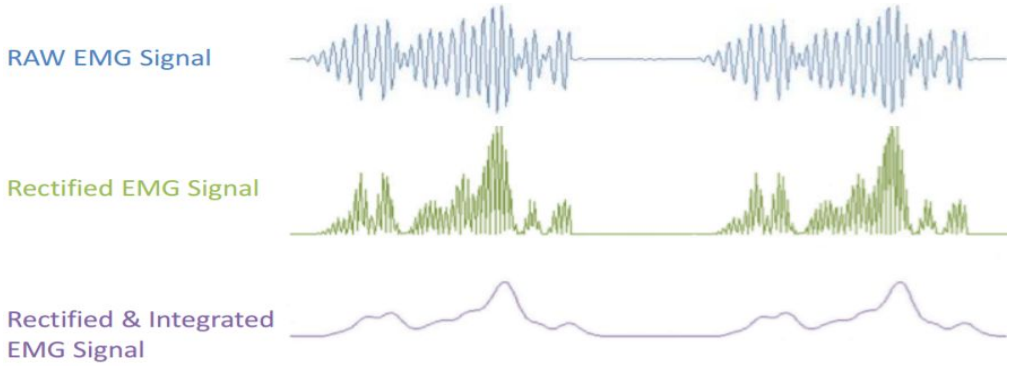

Figure 3. MyoWare Signal Processing

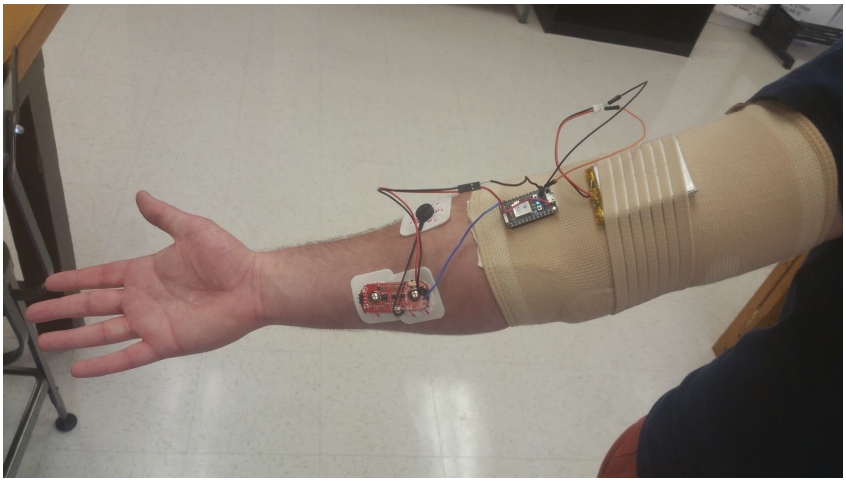

Figure 4. EMG Control Subsystem

Figure 5. MyoWare Muscle Sensor Board

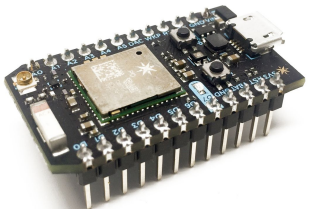

Figure 6. Particle Photon Microcontroller

The particle photon shown in Figure 5 then converted the signal using a analog to digital conversion with 12 bit resolution and a sampling rate of 10,000 Hz. The photon providing a reference 3.3v translated a .8mV per level conversion. Allowing for a translation of analog signal to the corresponding digital number between 0 and 4095. The photon then used event detection to begin sampling of the MyoWare board's incoming data. The voltage threshold to begin sampling was set at 400 levels or the corresponding voltage of 320 mV. The voltage threshold was used to filter out random movement noise and provide a more accurate segment extraction. A maximum sample threshold set at 5000 samples or the corresponding to 500 ms was used to limit the length of a segment and provide accurate movement segments. A minimum sample threshold of 1000 was used to also filter out unwanted random signals. Thresholds were determined experimentally through measurements of the signals on an oscilloscope. Based on the thresholds, once sampling began the equations shown below were updated with new values. Features were extracted from the segments using the continuous updating were root mean squared (2), wavelength (3) and mean signal value (4).

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{2}$$

$$Wavelength = \sum_{i=2}^{n} |x_i - x_{i-1}| \tag{3}$$

$$Mean = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4}$$

Once the features were extracted from the segments depending on the current system mode, calibration or operational, the segment was passed to a artificial neural network to be used in network learning or motion classification. In calibration mode the current operator of the system is asked to repeat a specific motion five times, saving the extracted features to be used in the ANN, shown in Figure 7.

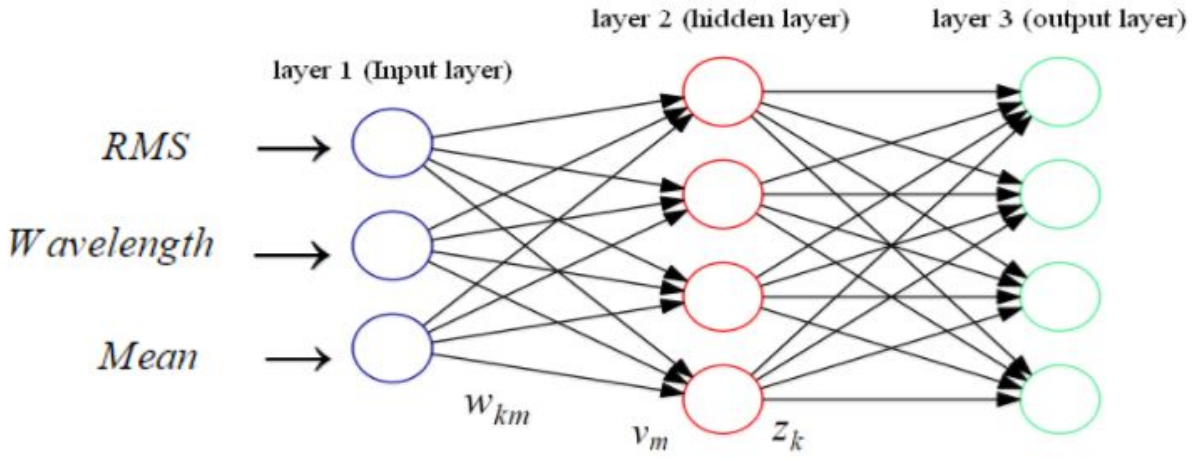

Figure 7. Artificial Neural Network

The artificial neural network was a 3 layer feedforward network consisting of 3 inputs, 4 hidden nodes, and 4 outputs shown in Figure 7. For this project, the sigmoid activation function (5) was chosen for the hidden layer, because the sigmoid function is ideal for implementing with multiple layer neural networks and allows for small changes in the output when there are small changes in the error allowing for smooth learning.

$$z_k = \frac{1}{1+e^{-v_k}} \tag{5}$$

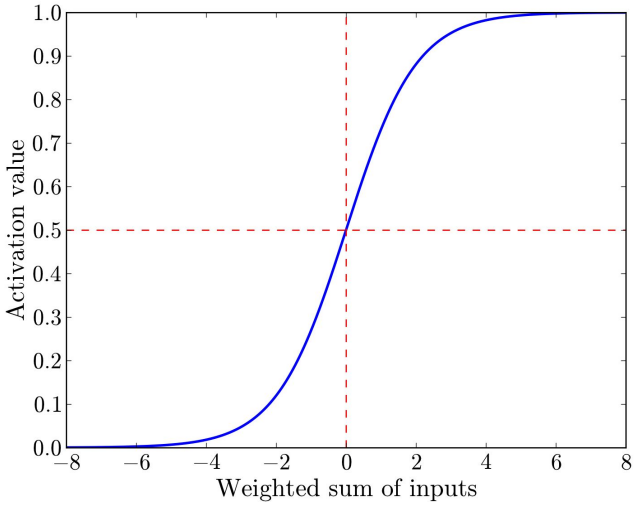

Figure 8. Sigmoid Function
Graph displaying sigmoid activation function squashing the pre-activation output to a value between 0 and 1.

Instead of a sigmoid activation function, the output layer used a softmax function (6) output function. This is because the output layer consisted of multiple output neurons, and softmax allowed the ANN to output a categorical distribution of all the output neurons. This is

important for the system, because it allows for ease of determining the correct output classification.

$$z_k = \frac{e^{v_k}}{\sum_{j=1}^{m} e^{v_k}} \qquad m = 1 \ldots \#\ of\ output\ neurons \tag{6}$$

Using the softmax function in output layer allows our network to learn by minimizing the cost function (7) of the neural network.

$$C = -\frac{1}{n}\sum^{n}[y_k * ln(z_k^L) + (1-y_k) * ln(1-z_k^L)]\ [4] \tag{7}$$

Equation 7 describes the cross entropy cost function, where y is the desired output and z is the actual output. The cost function is essential to the system to allow for incremental changes in the weights and bias to produce the optimal numbers. To minimize the cost function, or to learn, the neural network implements back propagation with gradient descent to find the errors within each layer. Backpropagation is the computation of error, the gradient, backward through the system. Error of the output was computed by using equation 8, which is the difference between the desired output and the actual output.

$$\delta_m^L = z_m - y_m\ [4] \tag{8}$$

Using the output layer error, the hidden layer error is then computed by using equation 9. The layers' errors were then used in the gradient descent to find the new weights and biases.

$$\delta_k^l = \sum (w_{km} * \delta_m^L) * z'(v_k)\ [4] \tag{9}$$

A graphical representation of gradient descent is shown in figure 9, displaying how the weight and bias will decrease until reaching a global cost function minimum. Utilizing the equations 10 and 11, we calculate the new layer's weights and biases respectively based on this theory.

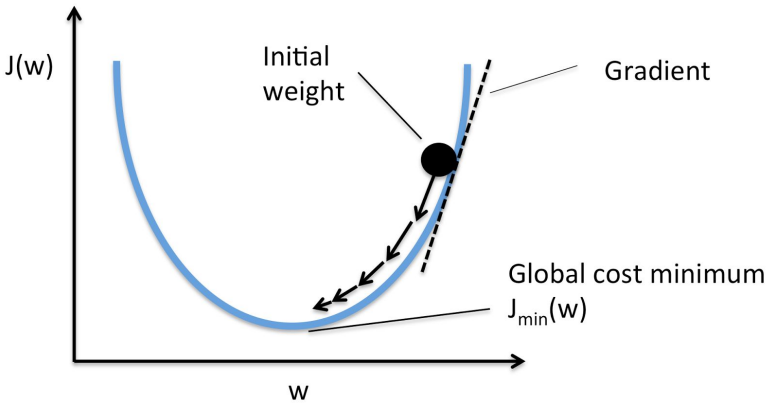

Figure 9. Gradient Descent Graphical Representation

$$w_{km} \rightarrow w_{km} - \frac{\eta}{m} \sum \delta_k * v_m \text{ [4]} \quad (10)$$

$$b_k \rightarrow b_k - \frac{\eta}{m} \sum \delta_k \text{ [4]} \quad (11)$$

Once the gradient descent has reached the global cost minimum the system will stop learning and the system will enter operational mode. In operational mode, the neural network classifier determined the motion classification using the weights and biases produced during calibration. The classified output from the neural network is a movement command which was transmitted to the Raspberry Pi using TCP/IP IEEE 802.11 protocol.

3.2 Wheeled Service Robot and Monitor Subsystem

The wheeled service robot is powered by 12V center tapped battery and equipped with dual differential drive motors. External hardware implemented on the robot was a Raspberry Pi 3 Model B along with a Logitech C260 webcam. The Raspberry Pi was powered by 5 V USB power bank to ensure the free movement of the robot without having to be plugged into a wall socket for power. The operating system (OS) Raspbian Jessie Lite was chosen due to open source access and its command line features along with extensive previous experience with it.

Figure 10. Raspberry Pi 3 Model B

Figure 11. Logitech C260

The specific version of Raspbian Jessie Lite was distributed on 11-17-2016. The lite version was installed because the GUI was not necessary since the Raspberry Pi would be headless once it was fully implemented on the robot. Headless for the Raspberry Pi means physically independent of other subsystems and runs the specified programs without user interaction. Once the SD card with the OS was mounted, a GPIO interface library, WiringPi, was installed in order to use the GPIO pins for PWM signal generation. The Raspberry Pi 3 has two PWM channels, four physical pins, on its forty GPIO header that may be used. Physical pins 12 and 33 were used on PWM channel zero and one respectively. Two different channels were used to ensure that the

two pins generated different PWM signals allowing the robot's left and right turns. If one channel is used both pins would have produced the same PWM signal not allowing the robot to produce a left or right turn. The code to implement this configuration can be found in Appendix A.

The motor control circuit implemented was a power transistor circuit to ensure that the PWM signals would have enough current to turn on the motors. The transistor circuit acts as a gate, which is a amplifier circuit, to allow voltage to pass through corresponding motor control pin generated by the Raspberry Pi. The circuit consisted of two TIP 31A power transistors, two 1k resistors for current limiting and two transistor base to ground 100k resistors used to guarantee the transistor turns off completely. Two diodes were added in parallel of each motor to protect from inverted voltage polarity.

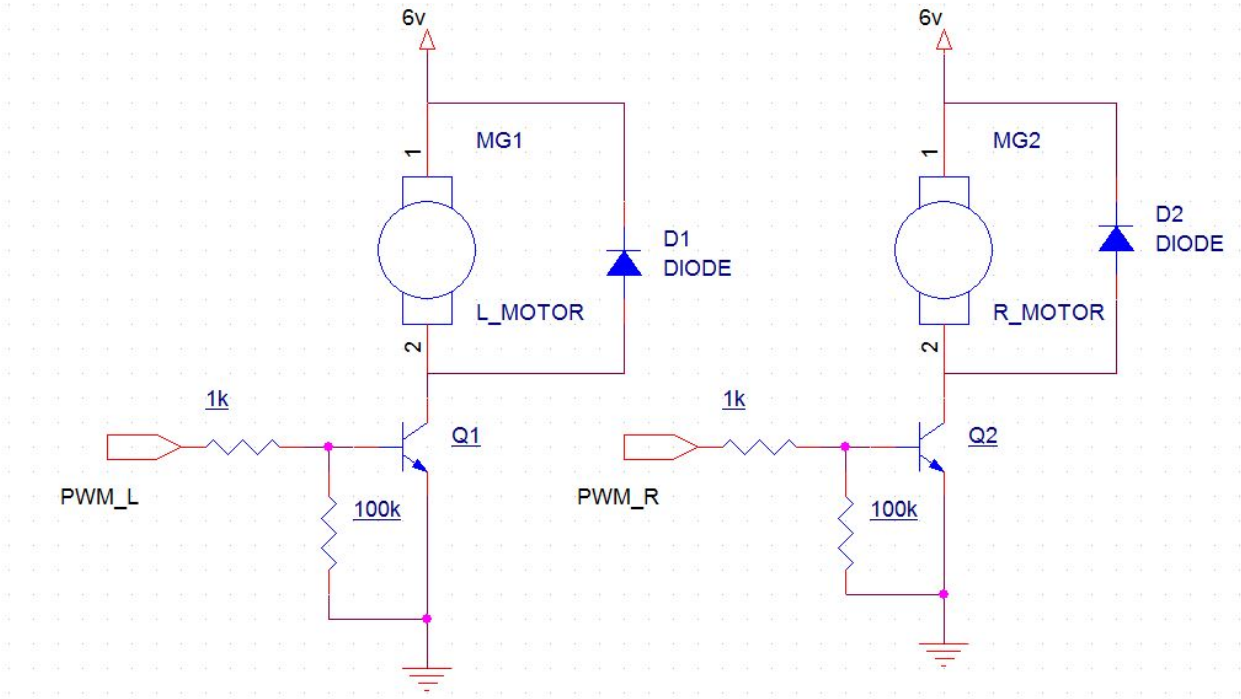

Figure 12. Motor Control Transistor Circuit

Each PWM signal was fed through the base of the transistor to turn on or off the transistor allowing the 6 V robot battery to power the motor control pins L_MOT1 and R_MOT1. This configuration can be seen in Figure 13 and 14.

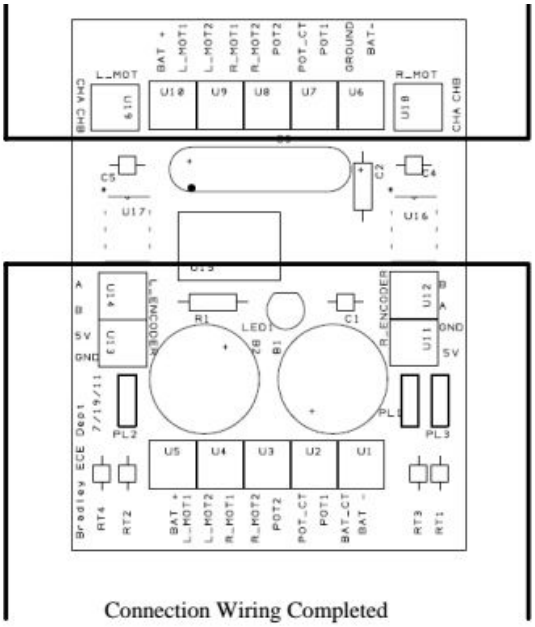

Figure 13. Robot Circuit Pin Layout

Figure 14. GPIO configuration

As stated earlier a TCP/IP IEEE 802.11 protocol was used for communication between both subsystems. The Raspberry Pi runs the server.c file, found in Appendix B, then waits to receive one of four commands to move the robot. Upon receiving a command the Raspberry Pi generated the specified PWM signals to the two hardware PWM pins. For example if the received command for the robot is to turn left, R_MOT1 pin's input will generate a PWM signal while L_MOT1 would remain shut off. The duty cycle of each command, besides the stop command, had a 97% duty cycle. A 100 % duty cycle can be implemented by setting the PWM count in the wiringPi specific function to 1024.

Simultaneously the Raspberry Pi would feed the video to a self-hosted website server. The website was implemented by the use of a Trivial Network Protocol (TNP) to ensure that the correct HTML code would be generated. The video monitoring subsystem was designed so the user had a live video feedback of where the robot was located in their home and always have access to user instructions and commands. The USB camera was attached to the Raspberry Pi that fed the video through Motion, an open source library, to generate a live video feed on the Raspberry Pi's network IP address. Motion's configuration file was set to have a resolution of 680 x 480, 30 frames per second (FPS) which is standard for a video to have little latency.

This video was embedded into a website to display the live video for navigation and instructions for the user on their home monitor. The user's personal website was accessed by searching the url http://'pi_address'.com which the Raspberry Pi generates upon booting up. The video feedback was designed so the user had private live video feed under their own network.

3.3 Stinger Robot

The Stinger Robot was used for system validation. The purpose of the robot was to verify PWM signal generation, communication between all subsystems and the hosting of a web server. Another reason why the this particular robot was chosen was because of previous experience with this particular platform and provided in house by Bradley University Electrical and Computer Engineering department.

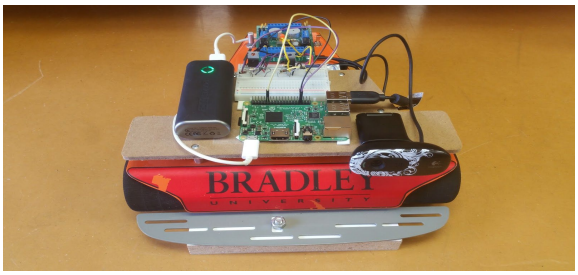

Figure 15. Service Robot: Front View

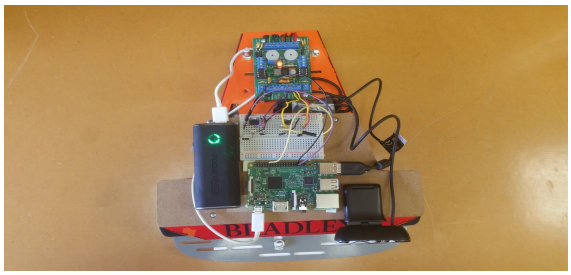

Figure 16. Service Robot: Top View

4. RESULTS

The variation of voltages within the movements are based on where the sensor board is placed on the forearm muscle. The targeted muscles of the sensor board were the brachioradialis and flexor carpi radialis. It was crucial to place the board in this configuration in order to ensure that each finger and hand movement can generate a feasible voltage signal. This configuration was successful, because the system generated four different unique voltage signals above the sample threshold to be used for the ANN.

4.1 Test Subject 1

The four oscilloscope readings represent each of the four commands the user will implement, shown in Figures 17-20. The muscle sensor board was capable of uniquely read and analyze different finger and hand movements. Figure 14 presents the flexion of the wrist which the user did three consecutive times and had a peak voltage of 3.22 V. In figure 14 the user made a fist three times and produced a peak voltage of 2.10 V. The user then exerted his ring finger three times which produced a peak voltage of 2.38 V. Finally the user exerted his middle finger three times with the peak voltage being 1.5 V.

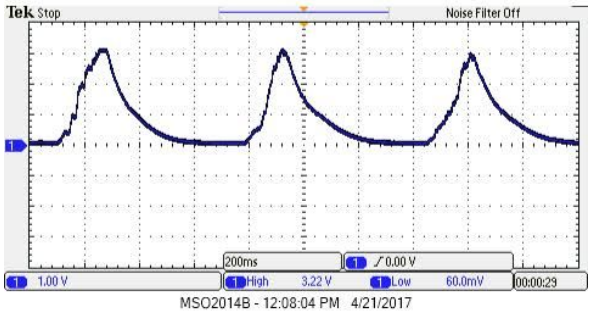

Figure 17. Wrist Flexion Movement

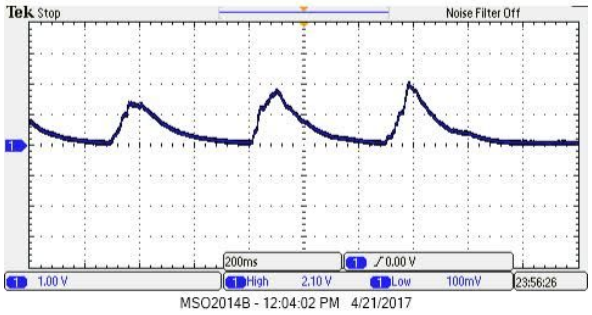

Figure 18. Fist Movement

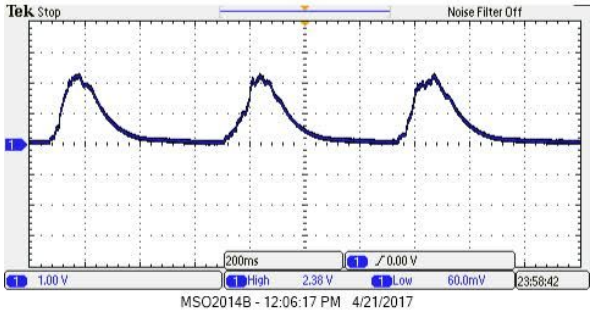

Figure 19. Ring Finger Flexion

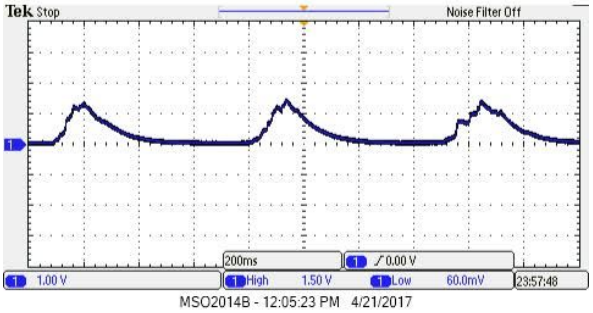

Figure 20. Middle Finger Flexion

4.2 Test Subject 2

Subject two also did the exact four commands and are displayed in figures 17 through 20. Figure 17 presents the flexion of the wrist which the user did three consecutive times and had a peak voltage of 2.66 V. In figure 18 the user made a fist three times and produced a peak voltage of 2.82 V. In figure 19 the user then exerted his ring finger three times which produced a peak voltage of 1.70 V. Finally the user exerted his middle finger three times with the peak voltage being 2.5 V in figure 20. Subject two had substantially different readings then subject one because of a natural tremor they have. The system was still able to classify and categorizes these signals to move the robot.

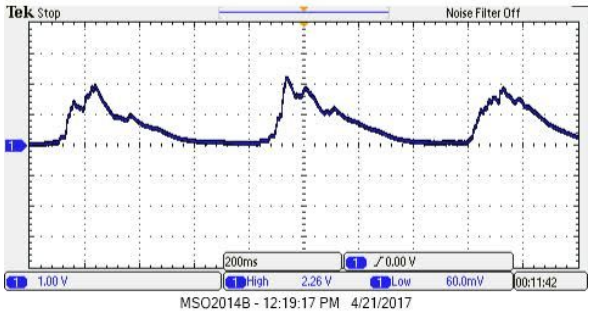

Figure 21. Wrist Flexion Movement

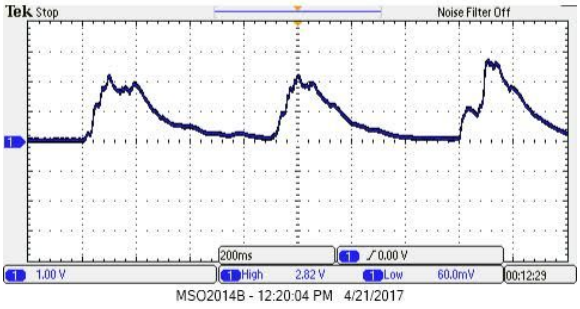

Figure 22. Fist Movement

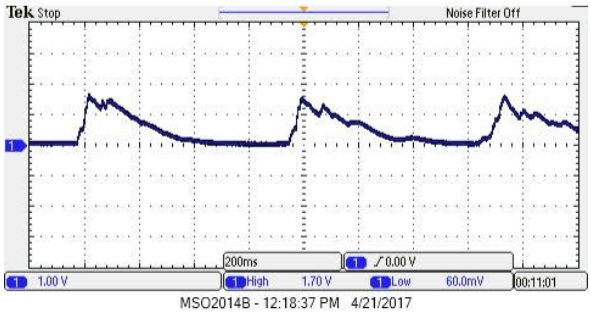

Figure 23. Ring Finger Flexion

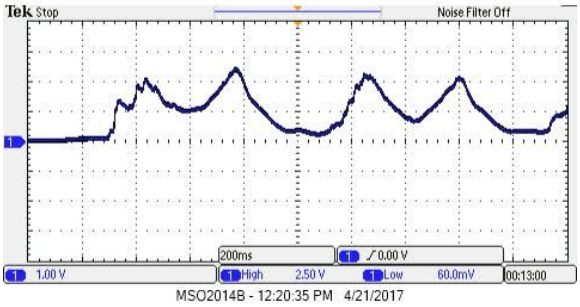

Figure 24. Middle Finger Flexion

4.3 PWM Output

The duty cycle of all generated PWM signals were set at 97% duty cycle. Each motor pin terminal had a oscilloscope channel attached to verify the duty cycle set GPIO pins of the Raspberry Pi 3.

When the motor goes forward both pins should be able to generate a PWM signal. Also both pins should simultaneously generate PWM with little latency at the exact same time to ensure that the robot would move forward. This was verified as both pins had a duty cycle of 97% and turned the motor on at same time. Figure 20 represents both pins on meaning the robot is in a straight path.

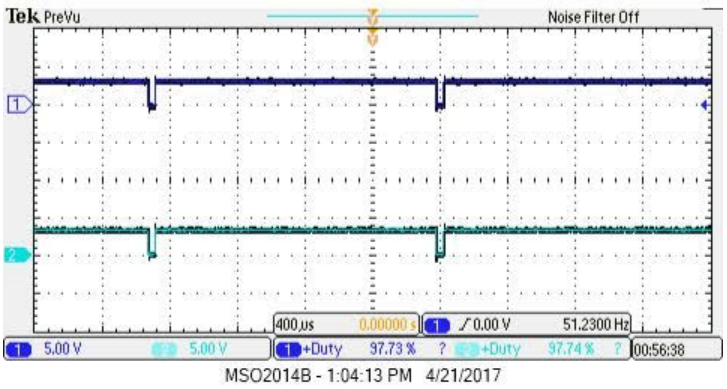

Figure 25. PWM Output - Straight Command

When a turn command is given only one GPIO pin should generate a PWM signal. The other GPIO pin would be set low so the motor one not turn on for that respective turn. Figure 22 represents left turn. In order for the robot to turn left the right motor would turn on while the left motor is off. Figure 23 is the opposite and it shows that the other pin is now off and the other pin is set to high to a 97% duty cycle. This configuration is for a right turn.

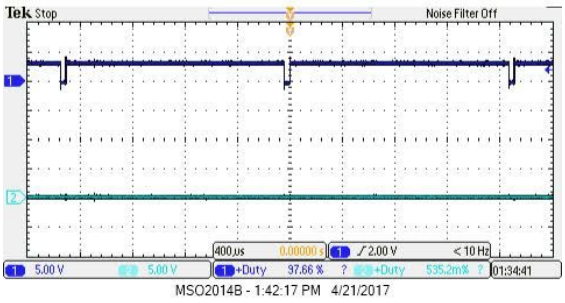

Figure 26. PWM Output - Left Turn

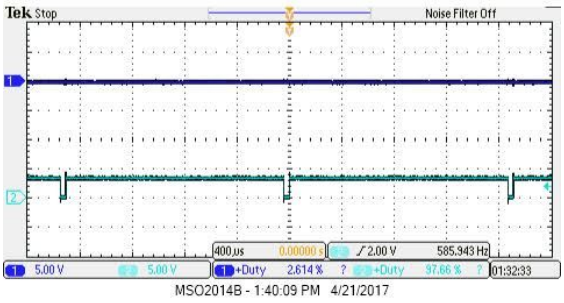

Figure 27. PWM Output - Right Turn

4.4 Website

The Raspberry Pi 3 was able to host the website for robot navigation and user instructions. Figure 26 is the website that the user of this system will be able to locate on the IP address that is assigned to the Raspberry Pi 3.

Figure 28. User Website

5. DISCUSSION

This project proposed a wearable EMG-based human machine interface system for in home assistance. The system consisted of three main subsystems, EMG control, robot service, and video monitoring subsystem. The EMG control was successfully implemented on the Particle Photon, and has achieved partial accuracy with the artificial neural network classification system. Considering the results, the number of sensors used to detect the EMG signals were found to be the limiting factor in this classification system.

The only consideration that was taken into account for the robot subsystem was that the Raspberry Pi generate PWM signals to move the wheeled robot. A larger platform would be needed for “real” applications.

6. CONCLUSION

Upon completion a wearable EMG human machine interface system has been developed. The system has successfully acquired EMG signals and extracted from the MyoWare muscle sensor board. The system can calibrate and classify EMG signals from different movements performed by the user of the system. The communication system was successful in transmitting commands from EMG control subsystem to be received accurately by the service robot system. A web server was implemented to provide live video feedback and instructions to the user.

7. FUTURE WORK

Future work in this project for the EMG control subsystem would include expanding to a larger mcu control board for a greater capacity of EMG sensors for a better accuracy of motion classifications. Other possible changes for better accuracy can include various other classification techniques, including probability classification, or investigation into other possible segment features including wavelet transform. If a proper amplifier chip can be implemented to reduce noise then the raw signal of the EMG may be used for signal analysis, which may improve accuracy of the classification.

The Pioneer P3-DX can replace the current robot to demo in a non-stationary environment, or robot with a servo motors can solve dual differential drive issue. There are many python gpio libraries to implement the code algorithm for the servo motors. The reason why the current robot can not drive straight with dual differential drive motor and caster ball wheel move separately based on separate PWM signals. In order to fix this problem some sort of feedback will have to be implemented so the robot can self correct. A more compatible usb can fix the issue regarding the timeout of the live feed. In the Motion documentation they have a table of all other compatible usb cameras that has been tested by others. A proper lithium battery is better suited for the wearable subsystem along with Pi being powered on by a higher current output around 2A.

REFERENCES

[1] Walsh, Connor J. Herman, Maxwell. Sanah, Siddharth. Galloway, Kevin C. Polygerinos, Panagiotis. “EMG Controlled Soft Robotic Glove For Assistance During Activities of Daily Living.” Harvard Biodesign Laboratory, August 2015.

[2] Keating, Jennifer. "Relating Forearm Muscle Electrical Activity To Finger Forces." Worcester Polytechnic Institute, May 2014.

[3] Stergiou, Christos. "Neural Networks." Department of Computing. Imperial College London, September 2011.

[4] Nielsen, Michael A. "Neural Networks and Deep Learning." *Neural Networks and Deep Learning*. Determination Press, 01 Jan. 1970. Web. 10 May 2017.

[5] "What Is EMG (Electromyography) and How Does It Work?" *IMotions*. N.p., 23 Sept. 2016. Web. 19 Apr. 2017.

[6] Wu, Yunfen, María Ángeles Martínez Martínez, and Pedro OrizaolaBalaguer. "Overview of the Application of EMG Recording in the Diagnosis and Approach of Neurological Disorders."*Overview of the Application of EMG Recording in the Diagnosis and Approach of Neurological Disorders | InTechOpen*. InTech, 22 May 2013. Web. 21 Apr. 2017.

[7] Gil, Jamie Gomez, Israel San-Jose-Gonzalez, Luis Fernando Nicolas-Alonso, and Sergio Alonso Garcia. *Steering a Tractor by Means of an EMG-Based Human-Machine Interface*. Tech. Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid,, 11 July 2011. Web. 24 Apr. 2017

[8] Ferreira, Andre, Wanderley C. Celeste, Fernando A. Cheein, Teodiano F. Bastos-Filho, Mario Sarcinelli-Filho, and Ricardo Carelli. "Human-machine Interfaces Based on EMG and EEG Applied to Robotic Systems." *Journal of NeuroEngineering and Rehabilitation*. BioMed Central, 26 Mar. 2008. Web. 10 May 2017.

APPENDIX A

```
#include <stdio.h>    // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!
#include "pwm.h"

// Pin number declarations. We're using the Broadcom chip pin numbers.
const int l_mot1_pwmPin = 18;      // PWM - Broadcom pin 18, Physical pin 12 PWM channel  0
const int r_mot1_pwmPin = 13;       //PWM - Broadcom pin 12, Physical pin 33  PWM channel 1

//to use another channel of pwm besides pwm0 use pwm1 on bcm 13, pin 33
void pwm_set();

int main()
{
   wiringPiSetupGpio() ;                    //using broadcom pins, use sudo ./"name of build" since it needs root priveleges
   pinMode(l_mot1_pwmPin, PWM_OUTPUT);          //Setting PWM pin 12 as output
   pinMode(r_mot1_pwmPin,PWM_OUTPUT);          //Setting PWM pin 32 as output

   pwmSetMode(PWM_MODE_MS);    //MARK:SPACE

   while(1)
   {
      server();

   }
}
//Duty Cycle  per pin will be at 30% duty cycle
void pwm_set()
{

   /* straight on robot */
   if(command == 1 )
   {
      pwmWrite(l_mot1_pwmPin,1000);
      pwmWrite(r_mot1_pwmPin,1000);
   }
   /* left turn on robot*/
   if(command == 2)
   {
      pwmWrite(l_mot1_pwmPin,0);
      pwmWrite(r_mot1_pwmPin,1000);
   }

   /*right turn on robot*/
   if(command == 3)
   {
```

```
      pwmWrite(l_mot1_pwmPin,1000);
      pwmWrite(r_mot1_pwmPin,0);
   }

   /*stop on robot*/
  if(command== 4)
   {
      pwmWrite(l_mot1_pwmPin,0);
      pwmWrite(r_mot1_pwmPin,0);            //physical pin 12

   }
}
```

APPENDIX B

```
int command=0;

void error( char *msg ) {
   perror(  msg );
   exit(1);
}

void store_data(int x)
{

   printf("\nthis is the new file %d\n",x);
}

void sendData( int sockfd, int x ) {
   int n;

   char buffer[32];
   sprintf( buffer, "%d\n", x );
   if ( (n = write( sockfd, buffer, strlen(buffer) ) ) < 0 )
      error( const_cast<char *>( "ERROR writing to socket") );
   buffer[n] = '\0';
}

int getData( int sockfd ) {
   char buffer[32];
   int n;

   if ( (n = read(sockfd,buffer,31) ) < 0 )
      error( const_cast<char *>( "ERROR reading from socket") );
   buffer[n] = '\0';
   return atoi( buffer );
}

int server(int argc, char *argv[]) {
   int sockfd, newsockfd, portno = 51717, clilen;
   char buffer[256];
   struct sockaddr_in serv_addr, cli_addr;
   int n;
```

```
    int data;

    printf( "using port #%d\n", portno );

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
       error( const_cast<char *>("ERROR opening socket") );
    bzero((char *) &serv_addr, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons( portno );
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0)
       error( const_cast<char *>( "ERROR on binding" ) );
    listen(sockfd,5);
    clilen = sizeof(cli_addr);

    //--- infinite wait on a connection ---
    while ( 1 ) {
       printf( "waiting for new client...\n" );
       if ( ( newsockfd = accept( sockfd, (struct sockaddr *) &cli_addr, (socklen_t*) &clilen) ) < 0 )
          error( const_cast<char *>("ERROR on accept") );
       printf( "opened new communication with client\n" );
       while ( 1 ) {
          //---- wait for a number from client ---
          data = getData( newsockfd );
          printf( "got %d\n", data );
          command = store_data(data);


          if ( data < 0 )
             break;

          //--- send new data back ---
          printf( "sending back %d\n", data );
          sendData( newsockfd, data );
       }
       close( newsockfd );

       //--- if -2 sent by client, we can quit ---
       if ( data == -2 )
          break;
    }
    return 0;
}
```

APPENDIX C

```
int  SocketLibStart() {
#ifdef WIN32
    WSADATA wsaData;
    return( WSAStartup(0x0101, &wsaData) );     /* register with winsock.dll */
```

```
#else
   return(0);
#endif
}

void SocketLibEnd() {
#ifdef WIN32
   WSACleanup(); /* release of winsock.dll */
#endif
}

SOCKET TCPStartServer(const int port, const int queue, const int reuse) {
   struct  protoent *ptrp;  /* pointer to a protocol table entry   */
   struct  sockaddr_in sad; /* structure to hold server's address  */
   SOCKET  sd;              /* socket descriptors                  */

   memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
   sad.sin_family = AF_INET;          /* set family to Internet   */
   sad.sin_addr.s_addr = INADDR_ANY;   /* set the local IP address */

   sad.sin_port = htons((u_short)port);   /* use given port number */

   /* Map TCP transport protocol name to protocol number */
   ptrp = getprotobyname("tcp");
   if ((char*)(ptrp)==0) {
      cerr << "cannot map \"tcp\" to protocol number" << endl;
      return(-1);
   }

   /* Create a socket */
   sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
   if (int(sd)<0) {
      cerr << "socket creation failed" << endl;
      return(-1);
   }

   if (reuse!=0 && setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,(char*)&reuse,sizeof(reuse)) <0 ) {
      cerr << "reuse address option failed" << endl;
      return(-1);
   }

   /* Bind a local address to the socket */
   if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
      cerr << "bind failed" << endl;
      return(-1);
   }

   /* Specify size of request queue */
   if (listen(sd, queue) < 0) {
      cerr << "listen failed" << endl;
      return(-1);
   }

   return(sd);
```

```
}

SOCKET TCPWaitForConnection(SOCKET sd) {
   struct  sockaddr_in cad; /* structure to hold ClientTCP's address  */
   socklen_t alen;         /* length of address                */
   alen = sizeof(cad);
   return( accept(sd, (struct sockaddr *)&cad, &alen) );
   /* Note: we could get the ClientTCP's location from cad structure */
}

int TCPStopServerTCP(SOCKET sd) {
   return( closesocket(sd) );
}

SOCKET TCPStartClient(const char *host, const char *port) {
   struct  addrinfo addr_req;   /* default address parameters (hints) */
   struct  addrinfo *addr_res;  /* ptr to the  address for connection */
   int     sd;                  /* socket descriptor                  */

   /* Convert host name and port name and address hints to the address */
   memset(&addr_req, 0, sizeof(addr_req));
   addr_req.ai_socktype = SOCK_STREAM;
   addr_req.ai_family = AF_INET; // Use: AF_INET6 or AF_INET or AF_UNSPEC
   if (0 != getaddrinfo(host, port, &addr_req, &addr_res)) {
      fprintf(stderr, "cannot set up the destination address\n");
      return(-1);
   }

   /* Create a socket. */
   sd = (int)socket(addr_res->ai_family, addr_res->ai_socktype, addr_res->ai_protocol);
   if (sd < 0) {
      cerr << "socket creation failed" << endl;
      freeaddrinfo(addr_res);
      return(-1);
   }

   /* Connect the socket to the specified server. */
   if (connect(sd, addr_res->ai_addr, addr_res->ai_addrlen) < 0) {
      cerr << "connection failed" << endl;
      freeaddrinfo(addr_res);
      return(-1);
   }
   freeaddrinfo(addr_res);

   return(sd);
}

int TCPStopClient(SOCKET sd) {
   return( closesocket(sd) );
}

int TCPSetTTL(SOCKET sd, int ttl) {
   return( setsockopt(sd,IPPROTO_IP,IP_TTL,(const char*)&ttl,sizeof(ttl)) );
}
```

```
int TCPSetTimeout(SOCKET sd, int tout) {
   return( setsockopt(sd,SOL_SOCKET,SO_RCVTIMEO,(char*)&tout,sizeof(tout)) );
}

int TCPSetFragment(SOCKET sd, int isOk) {
   return(0);
}

int TCPSetSocketReuse(SOCKET sd, int isOk) {
   return(setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char*)&isOk, sizeof(isOk)));
}

int TCPSetNoDelay(SOCKET sd, int isOk) {
#ifdef TCP_NODELAY
   return( setsockopt(sd,IPPROTO_TCP,TCP_NODELAY,(char*)&isOk,sizeof(isOk)) );
#else
   cerr << "TCP_NODELAY not supported by this platform - option ignored" << endl;
   return(0);
#endif
}

int TCPPrepClose(SOCKET sd) {
   return( shutdown(sd,SD_BOTH) );
}

int TCPRecvAny(SOCKET sd, char *buffer, const int maxsize) {
   return( recv(sd,buffer,maxsize,0) );
}

int TCPRecvLine(SOCKET sd, char *line, const int maxsize) {
   int     status;
   int     len=0;
   char    bch;
   while (1) {
      status = recv(sd,&bch,1,0);
      if (status<0) return(status);
      if (len>=maxsize) return(-1); /* someone overruns buffer, disconnect! */
      if (bch=='\n')
      {
         line[len]='\0';
         return(len);
      }
      else if (bch=='\r')
      {
         /* ignore \r */
      }
      else
      {
         line[len]=bch;
         len++;
      }
   }
}
```

```
int TCPRecvDumpLine(SOCKET sd) {
   int     status;
   int     len=0;
   char    bch;
   while (1) {
      status = recv(sd,&bch,1,0);
      if (status<0) return(status);
      if (bch=='\n') return(len);
      if (bch!='\r') len++;
   }
}

int TCPSendAny(SOCKET sd, const char *buffer, int size) {
   return( send(sd,buffer,size,0) );
}

int TCPSendLine(SOCKET sd, const char *line) {
   return( send(sd,line,int(strlen(line)),0) );
}
```