Team TELEKINESIS

Final Report

EE Senior Design –Spring 2012

University of Notre Dame

Andrew Fons
Adrian Moreno
Adebayo Omoyeni
Brian Rockwell
James Simonse

Table of Contents

1. Introduction

Conducting and analyzing electroencephalograms (EEGs, henceforth) historically have required a hospital setting with expensive machinery. Due to increases in technology and the “open-source” community’s ingenuity, consumer level EEG devices are now affordable, attainable, and intuitive to use. By working with these devices, we open up a new spectrum of research tools and opportunities, especially in the field of rehabilitation for stroke victims and patients with Attention Deficit Disorder (ADD). These tools will give doctors and lab assistants the ability to monitor concentration and relaxation levels in their patients while they try to conduct a task to better be able to help them in their rehabilitation. In light of this, we are proposing a non-invasive system that can be implemented at a low cost to assist with the rehabilitation process of such victims.

1.1 Problem Statement and Proposed Solution

Cognitive therapy has always been administered by a specialized doctor to those who have suffered from a stroke or for those who struggle with ADD. This therapy is found to be very useful in a patient's recovery but limited by the amount of time a doctor can allocate to a patient. Patients may only get an hour a week of therapy even they would benefit with more therapy time. Thus, we are proposing a Cognitive Therapy System (CTS), a system that incorporates a consumer-affordable EEG (electroencephalogram) in the form of a headset, which is no bulkier than a standard wrap-around headphones, and series of add-ons such as a robotic arm and an LED array. The headset will read signals directly from the user without them having to physically manipulate the system, send those signals to a computer to be processed, and pass the correct commands to a microcontroller which will control the robotic arm or LED arrays. This solution returns functionality to a user, has a relatively low cost, and can be implemented efficiently as the system can be customized to a user and can be mass produced. The headset provides a noninvasive way to monitor a user’s brain waves and can be removed easily at the end of the day with no discomfort to the user. This will then be used in conjunction with current therapy methods to increase the amount of therapy given to patients and monitor their progress.

2. System Requirements

2.1 Overall System:

The goal of this project is to create a cognitive therapy instrument to be used on those who have suffered from a stroke or others who may need cognitive therapy. This will be accomplished by using an electroencephalograph, or EEG, to measure the concentration levels of a patient and giving visual feedback. The visual feedback will be one of two things depending on what the nurse or doctor (user) sets up with the patient. The first feedback will be an array of 16 Red/Blue/Green LEDs. The second form of feedback will be a robotic arm. The data from the EEG will be analyzed and displayed on a computer and commands will be sent via USB to a board which will control the LEDs and robotic arm. The computer will allow the user to program the feedback to customize therapy and monitor the patient's progress.

The system requires a control test prior to beginning therapy. This will give a baseline profile and test that the EEG is correctly connected. Once the control test is finished, the user will configure the system for high reward and low effort. As the patient's cognitive capabilities improve, the effort required for visual feedback will be raised.

The system will come in three parts: the EEG headset, a box containing the LED array and main board, and the robotic arm. The EEG headset will be powered by a built-in rechargeable battery. It can be recharged with a USB connection and a charge should last at least 8 hours. The main board and the LED array will be powered by individual power regulated 3.3V from the USB connections from the computer. The robotic arm will be powered by 5V regulated wall wart. The use of USB power and wall wart will reduce the need for batteries.

2.2 Subsystem Requirements

2.2.1 Human/headset/computer

The EEG headset needs to send data signals to the computer through its wireless protocol. The computer will analyze the data and determine if the patient is concentrating. If the patient is concentrating, the computer needs to output an assigned command.

2.2.2 Computer/Microcontroller

The main board will have a pre-programmed microcontroller. When it receives a command via USB, it will output either to the motor drivers on the main board.

2.2.3 Microcontroller/LED Array

The LED array should comprise of functional multicolor LEDs. The microcontroller of the LED array must be able to dynamically control the LEDs as desired. The LEDs must be able to light up to reflect the present concentration level.

2.2.4 Microcontroller/Robotic Arm

The Robotic arm will have five motors controlled by the motor drivers on the main board. The motor drivers will be controlled via binary logic and direct all the motors on the arm. The robotic arm should be able to move in a manner that reflects the present concentration level.

2.3 Future Enhancements Requirements

In the interest of appealing to wider range of customers, the system should be integrated to both Windows and Mac operating systems. Although Linux works great and is free, there are more people who use Windows and Apple computers.

Making acquisitions more continuous and observing relaxation are additional goals. This would make for better data and more variables to work with so as to improve the quality of therapy given.

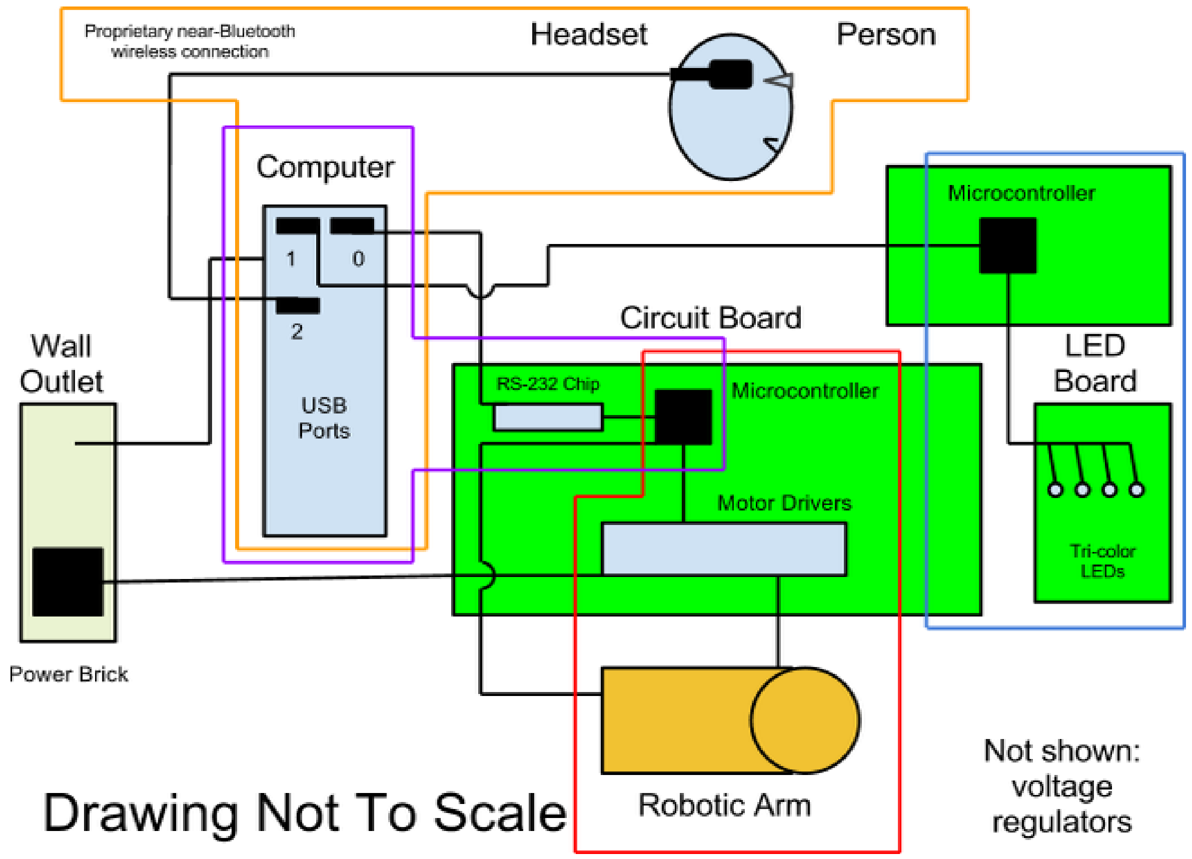
Finally, making a graphical user interface with our system will make it more user-friendly; a command line interface is likely unfamiliar to our target users.

3. Detailed Project Description

3.1 System Theory of operation

The process begins by establishing a baseline relaxed mental state. After this control is acquired, the EEG headset reads the patient's concentration level and transmits the collected information to the computer which analyzes the data. Using the analytic tool Octave to perform Welch's method, our code determines if the user was concentrating and the duration and intensity of concentration. Based on this information, the computer sends digital signals to the microcontroller which in turn controls the action of the robotic arm or the LEDs, depending on which rewards system is selected. Specifically, the LED system is controlled through a dedicated microcontroller using binary signals. The motor drivers that control the arm are also controlled in a similar manner through binary signals. The headset is powered by rechargeable lithium-ion batteries, the microcontrollers by USB, and the arm is powered by a regulated wall power source.

3.2 System Block Diagram

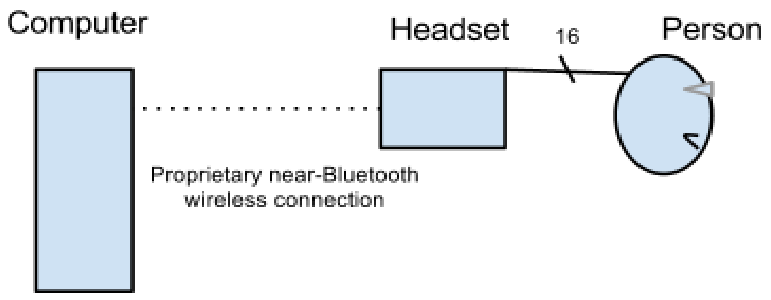


_________orange- Human–headset–computer subsystem
_________purple- computer–microcontroller subsystem
_________red- Microcontroller–Arm subsystem
_________blue- Microcontroller–LED subsystem

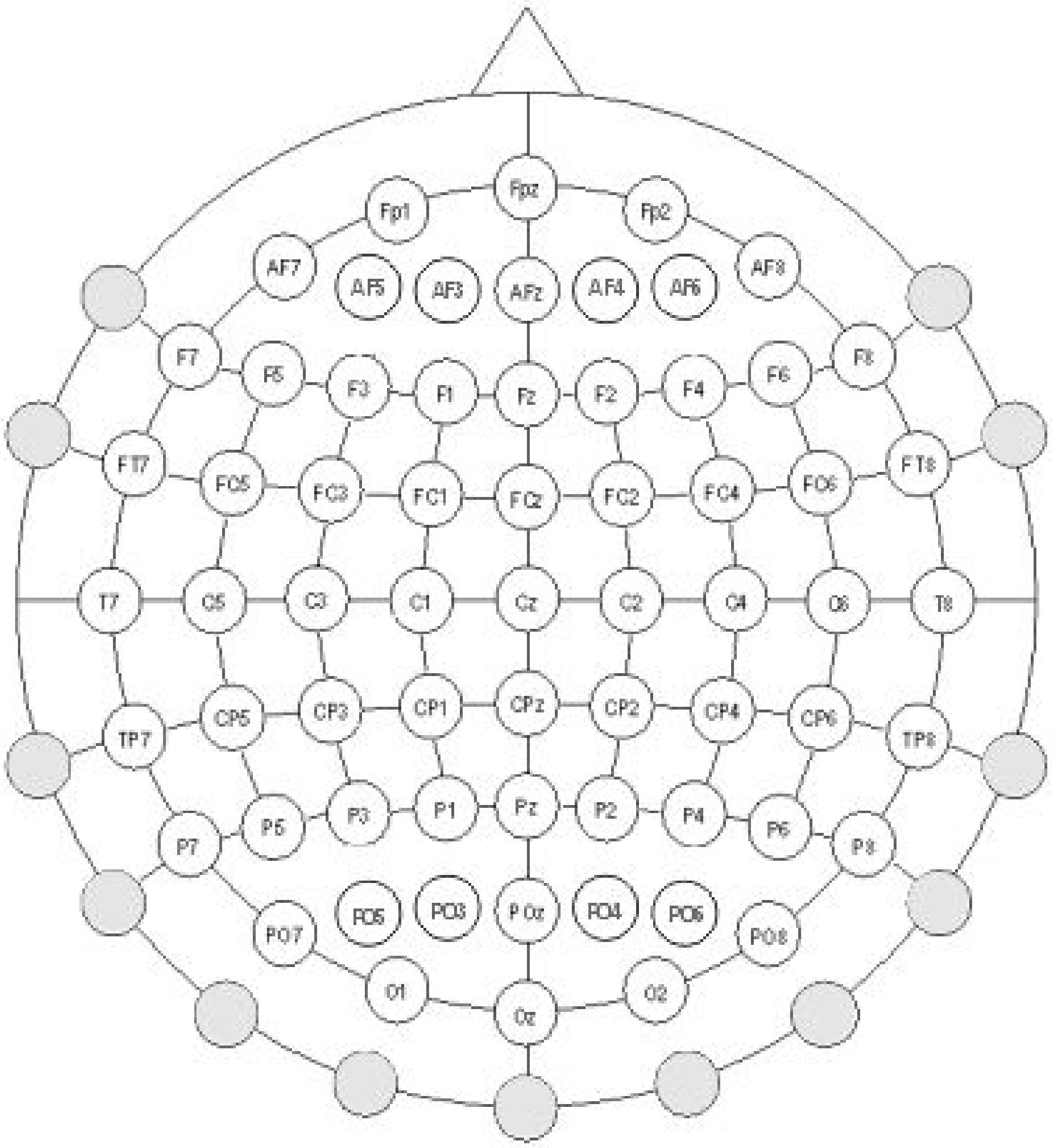
3.3 Human/Headset/Computer description

Subsystem Requirements
This system must be able to extract EEG data from the user and analyze it to find out whether or not someone is concentrating in pseudo real-time. The computer also monitors degree and duration of concentration.

Human-Headset-Computer Interface



Our headset is an Emotiv Epoc, which has sixteen sensors spread out over the person's head. Two of those are used as reference sensors, while the fourteen non-reference sensors are F3, F4, F7, F8, FC5, FC6, AF3, AF4, O1, O2, P7, P8, T7, T8. The sensor positions may be seen below

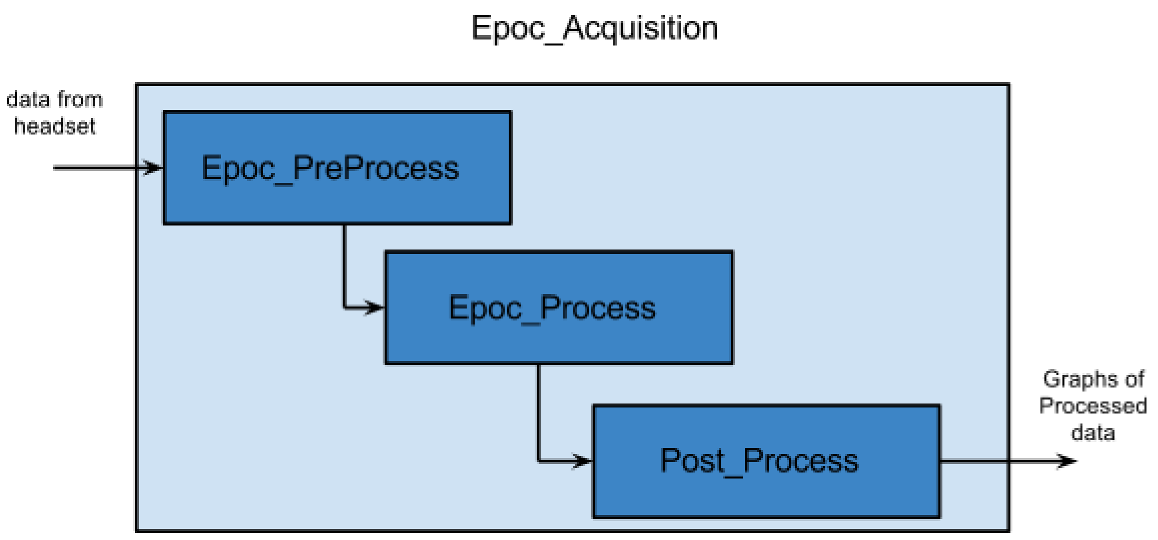


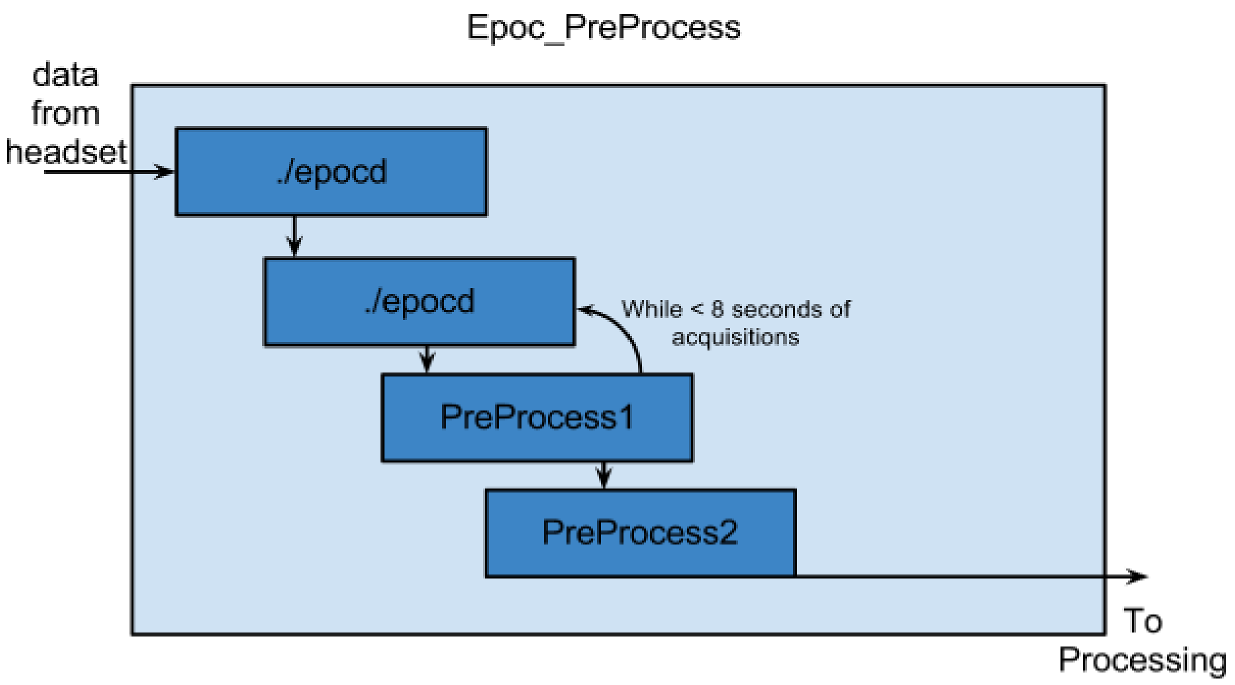
As a person thinks, neurons fire in the person's brain. These small movements of charged particles create time-varying voltages across the scalp. The EEG sensors measure these voltages and report the information to the headset.

The headset takes the fourteen sensors' data, adds gyroscope information, and transmits those sixteen channels of information to the computer. When our program receives the information, it discards the two channels of gyroscope data and processes the rest. Unless otherwise specified, all code is in Octave. The software is organized as follows:

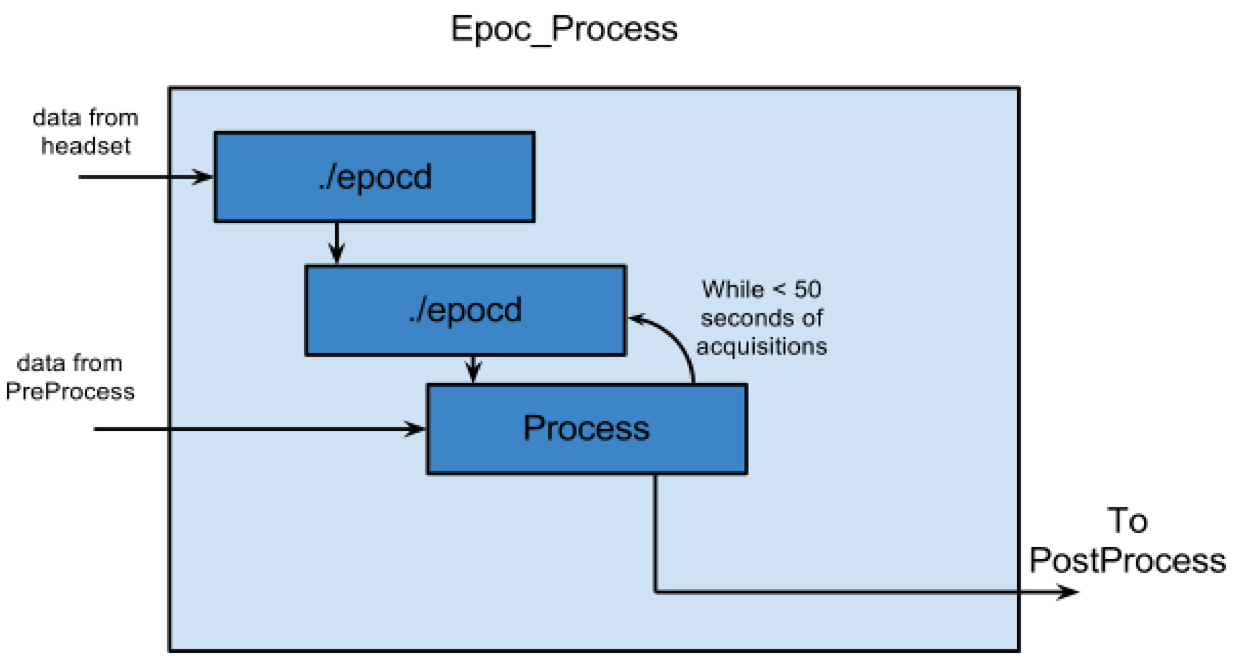
Epoc_Acquisition

- Epoc_PreProcess
  - LOOP [./epocd.c (C code)
    - PreProcess1
      - splitdata
      - bandpassData
        - bandpassFiltFilt
      - avgPSD_Data
        - avgPSD
          - pwelch
        - writes average]
  - PreProcess2
    - PreProcessData
      - Reference Data
    - writes references
- Epoc_Process
  - LOOP [./epocd.c (C code)
    - Process
      - split data
      - bandpassData
        - bandpassFiltFilt
      - Referenced Average
        - refAvg
          - avgPSD
        - ./connect w (C code)
          - tty USB0 signal
      - writes data]
- PostProcess
  - PSD_Reference

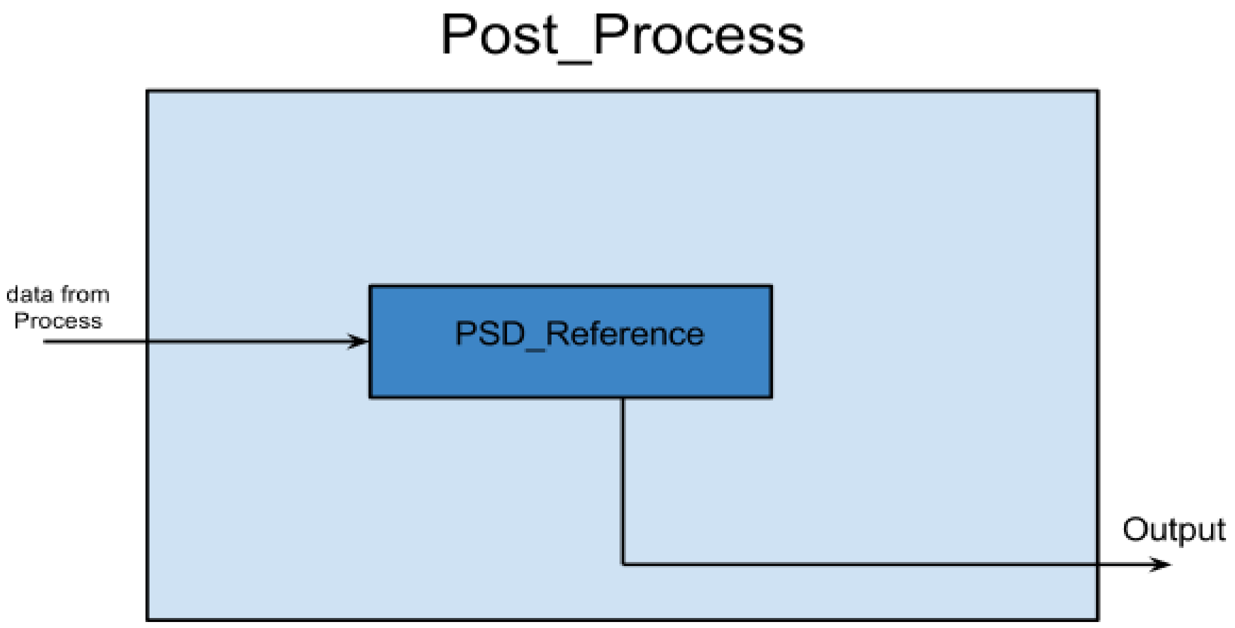




epocd is an acquisition program. When running Epoc_PreProcess, epocd first runs (N-1) acquisitions, where N is the number of acquisitions in one second, then enters a loop in which the software runs one acquisition and runs PreProcess 1 on the previous second of acquisitions. PreProcess 1 separates the data into its fourteen component channels, bandpasses each channel to extract data between 12 Hz and 32 Hz (we actually start at 10.8 Hz and end at 35.2 Hz to avoid the infinite steepness of an ideal filter), and averages the power spectrum density over each one second interval. PreProcess2 finds the average, standard deviation, and variance of all of the one second trials for each channel.



In Epoc_Process: epocd will run (N-1) acquisitions, then enter the loop until fifty seconds of acquisitions are taken. In the Process block, each 1 second acquisition has the average of the eight second preprocessed data subtracted. If (data - 8 second average) < 0, set the result equal to zero. If the result is greater than the standard deviation of the preprocessed data, that is considered "concentrating".



PSD_Reference takes the data from process and graphs power spectrum densities for each of the 14 data channels. However, we are only interested in AF3 and AF4 to check for concentration levels.

3.4 Computer/ Microcontroller description

This subsystem consists of the computer and its communication with the microcontroller.

Subsystem Requirements

After our program has identified the patient's concentration level, the computer must communicate that information to the microcontroller. The RS-232 chip converts USB voltage levels to TTL voltage levels so the microcontroller can handle it. It allows RS-232 communication between the computer and microcontroller.

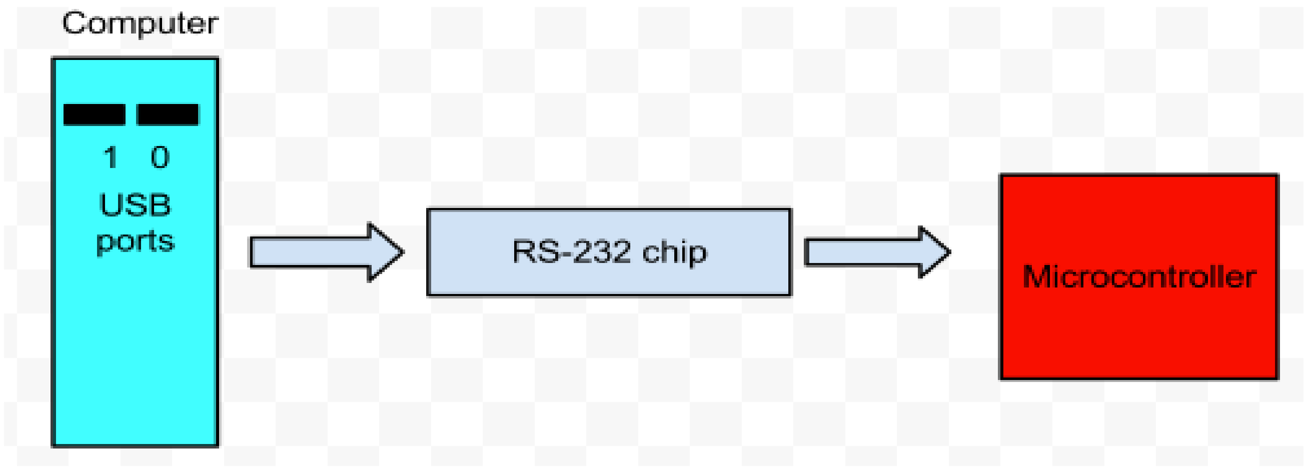


Diagram of computer– microcontroller subsystem

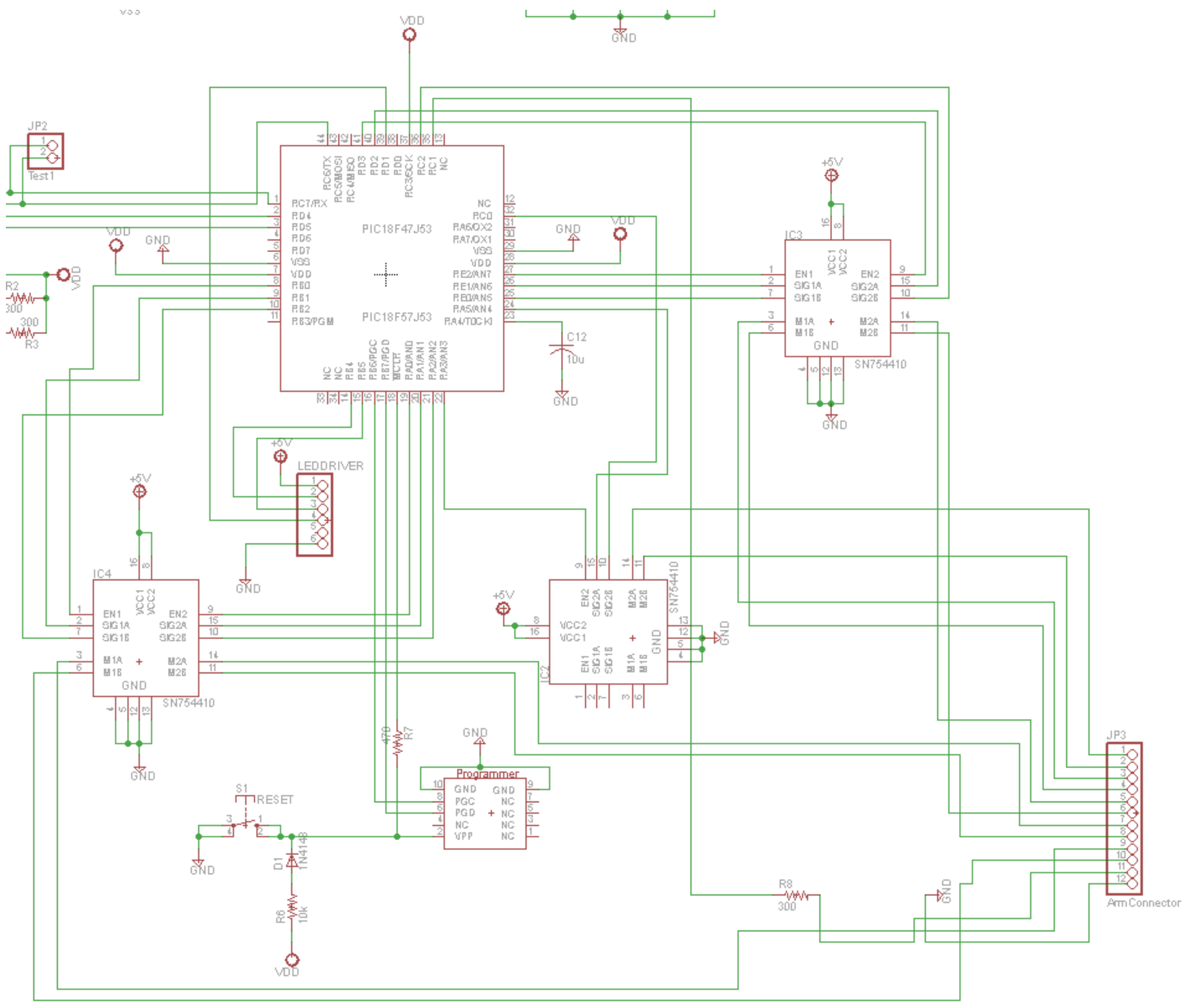
The diagram above illustrates the path of information from the computer to the microcontroller.

We chose FT232RL for our RS-232 chip because we were already familiar with it from Senior Design 1. Also, the chip's presence on the Senior Design kit boards allowed us to prototype early, long before any orders were made. An advantage that proved invaluable.

3.5 Microcontroller/ Arm description

Subsystem Requirements

The microcontroller must be able to talk to motor drivers and successfully manipulate the arm. Specifically, by using three motor drivers, we should be able to control each motor in the arm independently. Also, there is a direct connection from the microcontroller to the arm to turn the arm's LED on or off.



The three motor drivers are part number SN754410. The microcontroller currently communicates with the motor drivers by sending them bits. However, they do have the capability to respond to trits (-1, 0, or 1). We chose to use these particular motor drivers because they can source enough current to drive our motors and work at a low voltage.

3.6 Microcontroller/LED array description

Subsystem Requirements

Our EEG headset measures the concentration level of patients. We want our LED system to reward the user based on their ability to concentrate. The LED array must be capable of simultaneous operation as directed by the microcontroller. We did not meet this requirement, instead we drove current directly through a microcontroller.

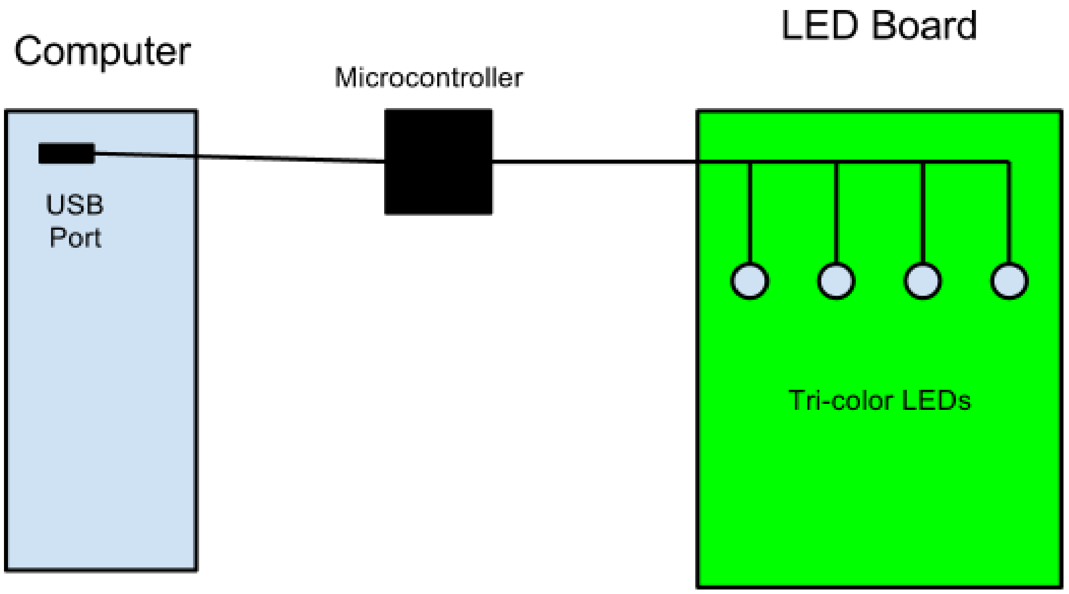


Diagram of LED subsystem

Above, we have a diagram of the LED system. We chose to control the LEDs directly with a microcontroller because the microcontroller can source enough current to power a few LEDs. It is time efficient and good for prototyping.

4. System Integration Testing

4.1 How the integrated set of subsystems was tested

All of the subsystems were created individually before being tested. The system integration test took three major phases to reach completion: 1) colinear development of computer-side and microcontroller-side subsystems, 2) integration of all computer-side subsystems, and 3) step-by-step integration of the microcontroller-side subsystems into the computer-side subsystem. Ideally, step 2 would also have had the all of the microcontroller subsystem integrated together and step 3 would just have been the two sides joining together, but the parts of the microcontroller-side subsystems were not available until much after the computer-side subsystems.

The computer-side subsystems are predominated by EEG data extraction and analysis. For development purposes they were conducted at different times but the final product demands that they are run simultaneously. Extraction involves more than just pure extraction - it also involves the small training program conducted before the real test to find reference values. Thus, the integration of the computer-side subsystems took the form of EEG acquisition and saving the data, analyzing the data, performing both functions in pseudo real-time, then adding the training period to the extraction routine and incorporating the values found there into the processing of the signals.

Once integrated, each member of the team underwent three different trials to test the system. These trials consisted of an 8-second relaxation reference test followed by one of the 100-second extraction tests - Relax, Focus, and Alternate, each requiring the subject to keep their eyes open at all times. Relax demanded the patient clear their mind and not concentrate, Focus demanded the opposite, for the subject to continuously concentrate, and Alternate required the subject to switch between the two previous states every few seconds. In the processing of the data, whenever the system detected the appropriate degree of concentration it would print to the terminal a message relating its findings.

With the acquisition system validated, the next step was for the computer to begin talking to the microcontroller through RS232 communication using a USB port as a COM port. To do this, we added a system command in addition to the terminal display to be executed when the system determined a subject was concentrating. This system command executed a compiled C++ program using TTYUSB0 to send raw serial data to the microcontroller, which would print a message to an LCD display upon receiving this input. The LCD display was used as a temporary stand in while parts for the main board arrives. Upon receiving the parts and soldering together the board, two LEDs on the board would flash whenever it receives RS232 communication. From this point forth, testing and integration of the system used the RS232 communication from the computer to test components.

Next task was to successfully send commands to the motor drivers from the microcontroller upon receiving a character input from the RS232 communication on the USART. The "pincers" and the LED on the arm stood as the barometer for how well this worked, with the goal to be reversing the signal sent upon the reception of a character input. For the LED, this meant turning on and off, while the pincer would alternate between opening and closing. This was chosen for the pincer as it is technically controlled through a trit rather than a bit; it has the states opening, closing, and motionless. Opening and closing were the states chosen for ease of testing and avoiding hardware damage from overextension.

Only one LED driver was used for the testing of the LED subsystem, with the goal to be able to turn all the LEDs in the system on and off at once. Fine control was ignored for this part in favor of proof of a working system. The toggle for the LEDs was once again based upon the computer sending a single character down the RS232 communication line. This was first conducted without the arm subsystem attached, then with it attached.

4.2 How the testing demonstrates that the overall system meets the design requirements

While the bulk of the processing takes place on the computer-side of the system, it is vital for the computer to communicate with the microcontroller to enable feedback/ rewards for the subject. The testing shows that the pilot of the system can begin an acquisition cycle that will be able to acquire EEG signals, process those signals, and determine concentration levels from them. This system, when it determines an appropriate amount of concentration, can send various instructions to the microcontroller, thus allowing the system to relay the length of concentration. By sending simple commands through the microcontroller to the various drivers, we show the capability of our system to execute a series of fine commands. This results in a complex reward system for the subject.

5. User Manual / Installation Manual

5.1Disclaimer:

Our EEG therapeutic system currently only works in a Linux environment and makes use of the open-source scripting language and interpreter Octave with the signal processing toolkit installed. Attempting to install the software on another operating system will cause errors and prevent the system from working. While it is possible to run Octave scripts on Matlab, there is no guarantee that they are universally compatible. However, with slight modification found on the emokit website, the project can be run on a Macintosh machine. Currently there is no support for running the project from a Windows machine. All instructions henceforth assume a linux OS.

5.2 Downloading Linux:

The software was originally created on the Linux Mint OS v12 "Lisa" but should be compatible with other linux distributions. Linux Mint can be downloaded from http://www.linuxmint.com/ and can be either fully installed or installed side-by-side with another OS for dual-booting purposes. Please download the most stable, recent version available.

5.3 Downloading Octave:

Once linux is downloaded and fully installed, the next step is to download and install octave and the octave signal processing toolkit. To do so, open a terminal and type: "sudo apt-get install octave" which will begin the download process. Once everything is acquired, run the command: "sudo apt-get install octave-signal", which will install the signal processing toolkit for octave.

5.4 Installing the required packages:

In the InstallationFiles folder there should be four zipped folders: the emokit, CMake, libusb, and libmcrpyt. Extract and install the emokit last (this makes compilation easier), but the order of installing CMake, libusb, and libmcrypt do not matter. For those three programs, follow the directions found in their respective README files on how to install (usually a case of make install-> make).

Once done, extract the emokit and navigate to the emokit's 'c' directory. Once there, type "cmake ." to create the necessary Makefiles. Then replace the c/examples/epocd/epocd.c file present with the one in the project's NecessaryFiles folder. Then place the files in the NecessaryFiles/AcquisitionFiles folder in the emokit's c/bin directory as well as making a folder named "AcquisitionData" in the c/bin directory. Return to the 'c' directory, then type 'make.' This should compile all of the pieces.

Running the software

Once all of the above steps have been completed, open a terminal, navigate to the emokit's c/bin directory, and type "octave" to bring up an octave terminal. Then type Epoc_Acquisition to begin the process.

To view the graphs of the data, in an octave terminal type "PostProcess".

Troubleshooting

The cord to the microcontroller package is plugged in before the cord to the headset's usb device, otherwise the software won't be able to send the data correctly.

The headset must be fully charged and turned on with the sensors placed correctly for the system to work properly. If you are getting faulty results, check the placement of the sensors.

A saline solution is provided for wetting the sensors. To ensure proper data acquisition, ensure the sensors are sufficiently moist.

Once all of the software is installed, plug the EEG headset's USB dongle into USB Port 2, the microcontroller for the LED board into USB Port 1, and the main microcontroller board into USB Port 0. The arm must be connected to the main board.

To reset the arm's position, type ./connect <blank> where blank is a character from the following set: {q w e r t; a s d f g; z x c v b}. The first five letters STOP motors 1-5 respectively. The second five send those motors forward, while the last five send them in reverse. "./connect 1" will stop all motors.

To set the concentration threshold levels, look in 8.1.19 ReferencedAverage.m for limitLevel1, limitLevel2, and limitLevel3. These define the concentration thresholds from lowest to highest. The numbers you set for these variables are fractions of the standard deviation of your concentration level from the eight-second reference period at the beginning of an acquisition.

6. To-Market Design Changes

We have designed a system that has instant applications in the present theraphy process of ADHD and stroke victims. Due to the limitations placed on us by our budget and allocated time, a lot of room for improvements has been left. To take our product to market we would need to make a couple of modifications. First of all, the controls of the LEDs should be changed. Presently, the LEDs are being controlled by a separate microcontroller. This implementation though adequate, limits the number of LEDs that can be used for the array. This can be adequately addressed by the use of and LED driver.

Another change that would improve the performance and accuracy of our system is a change to the programming environment in which we operate. Currently, we are implementing our code in both interpretive and compiled computer languages. To significantly increase the operating speed, we will need to switch to exclusively compiling coding language. This move would also improve our real-time operation capabilities. Another direction to move towards with regards with modifications is to increase the scope of data that is analyzed. Presently, we are only considering data from two channels, AF3 and AF4 with which we were able to harness the concentration data from our users. By considering the data from the other twelve sensors we can get more data that can be used for measuring other brain activity.

A useful modification would be to enable profiling. This entails setting up individual profiles for each user of our system, saving peculiar information about them including a progress report. This would effectively eliminate the need to train the system and do the relaxation step. To improve aesthetics and make our system easier to use, a GUI should be developed for the LED array.

7.Conclusion

While our project did not successfully meet all of its requirements, we did produce a working feedback system for an EEG headset without using any developer code from the headset manufacturers.

8. Appendices

Code is given in alphabetical order by name of file.

8.1 Computer Code

8.1.1 avg.m

```
function [result] = avg(data)
% finds the average of the power

total = 0;
num = length(data);

for i=1:num
   total = total + data(i)^2;
end

result = (total / num);

% scaled down by large factor
result = result * 1e-6;
end
```

8.1.2 avgData.m

```
function [GYROX GYROY F3 F4 F7 F8 FC5 FC6 AF3 AF4 O1 O2 P7 P8 T7 T8] =
avgData(GYROXband, GYROYband, F3band, F4band, F7band, F8band, FC5band, FC6band,
AF3band, AF4band, O1band, O2band, P7band, P8band, T7band, T8band)
```

```
GYROX = avg(GYROXband(:));
GYROY = avg(GYROYband(:));
F3 = avg(F3band(:));
F4 = avg(F4band(:));
F7 = avg(F7band(:));
F8 = avg(F8band(:));
FC5 = avg(FC5band(:));
FC6 = avg(FC6band(:));
AF3 = avg(AF3band(:));
AF4 = avg(AF4band(:));
O1 = avg(O1band(:));
O2 = avg(O2band(:));
P7 = avg(P7band(:));
P8 = avg(P8band(:));
T7 = avg(T7band(:));
T8 = avg(T8band(:));



end
```

8.1.3 avgPSD.m

```
function [result] = avgPSD(data)
% finds the average of the power density

headsetFrequency = 128;
segmentLength = headsetFrequency / 8;
overlap = .25;

headsetFrequency = 128;

[Pxx, f] = pwelch(data, segmentLength, overlap, length(data),
headsetFrequency, 'onesided', 'plot');

result = mean(Pxx);

end
```

8.1.4 avgPSD_Data.m

```
function [GYROX GYROY F3 F4 F7 F8 FC5 FC6 AF3 AF4 O1 O2 P7 P8 T7 T8] =
avgPSD_Data(GYROXband, GYROYband, F3band, F4band, F7band, F8band, FC5band,
FC6band, AF3band, AF4band, O1band, O2band, P7band, P8band, T7band, T8band)

GYROX = avgPSD(GYROXband(:));
GYROY = avgPSD(GYROYband(:));
F3 = avgPSD(F3band(:));
F4 = avgPSD(F4band(:));
F7 = avgPSD(F7band(:));
F8 = avgPSD(F8band(:));
FC5 = avgPSD(FC5band(:));
FC6 = avgPSD(FC6band(:));
AF3 = avgPSD(AF3band(:));
AF4 = avgPSD(AF4band(:));
O1 = avgPSD(O1band(:));
O2 = avgPSD(O2band(:));
P7 = avgPSD(P7band(:));
P8 = avgPSD(P8band(:));
T7 = avgPSD(T7band(:));
T8 = avgPSD(T8band(:));



end
```

8.1.5 bandpassData.m

```
function [GYROXband GYROYband F3band F4band F7band F8band FC5band FC6band
AF3band AF4band O1band O2band P7band P8band T7band T8band] = bandpassData(Fs,
cutoff1, cutoff2,GYROXdata, GYROYdata, F3data, F4data, F7data, F8data, FC5data, FC6data,
AF3data, AF4data, O1data, O2data, P7data, P8data, T7data, T8data)

GYROXband = bandpassFiltFilt(GYROXdata, Fs, cutoff1, cutoff2);
GYROYband = bandpassFiltFilt(GYROYdata, Fs, cutoff1, cutoff2);
F3band = bandpassFiltFilt(F3data, Fs, cutoff1, cutoff2);
F4band = bandpassFiltFilt(F4data, Fs, cutoff1, cutoff2);
F7band = bandpassFiltFilt(F7data, Fs, cutoff1, cutoff2);
F8band = bandpassFiltFilt(F8data, Fs, cutoff1, cutoff2);
FC5band = bandpassFiltFilt(FC5data, Fs, cutoff1, cutoff2);
FC6band = bandpassFiltFilt(FC6data, Fs, cutoff1, cutoff2);
AF3band = bandpassFiltFilt(AF3data, Fs, cutoff1, cutoff2);
```

```
AF4band = bandpassFiltFilt(AF4data, Fs, cutoff1, cutoff2);
O1band = bandpassFiltFilt(O1data, Fs, cutoff1, cutoff2);
O2band = bandpassFiltFilt(O2data, Fs, cutoff1, cutoff2);
P7band = bandpassFiltFilt(P7data, Fs, cutoff1, cutoff2);
P8band = bandpassFiltFilt(P8data, Fs, cutoff1, cutoff2);
T7band = bandpassFiltFilt(T7data, Fs, cutoff1, cutoff2);
T8band = bandpassFiltFilt(T8data, Fs, cutoff1, cutoff2);

end
```

8.1.6 bandpassFiltFilt.m

```
function [DATA] = bandpassFiltFilt(data, Fs, cutoff1, cutoff2)

%period = 1 / Fs;
%ticks = (1:numel(data))*period;

offset1 = cutoff1 / 10;
offset2 = cutoff2 / 10;

%Declare variables for the six points of F:
P1 = 0 ;
P2 = (cutoff1-offset1)/(Fs/2);
P3 = cutoff1 / (Fs/2);
P4 = cutoff2 / (Fs/2);
P5 = (cutoff2+offset2)/(Fs/2);
P6 = 1;

% design specifications for a band pass filter
F = [P1 P2 P3 P4 P5 P6];
A = [0 0 1 1 0 0];

% twenty point finite impulse response
%flt = fir2(20,F,A);
flt = [0.0023 0.0067 0.0050 -0.0033 0.0074 0.0211 -0.0490 -0.1706 -0.1220 0.1468 0.3145
0.1468 -0.1220 -0.1706 -0.0490 0.0211 0.0074 -0.0033 0.0050 0.0067 0.0023];


DATA = filtfilt(flt,1,data);

%Plot the original and the filtered data
```

```
%subplot(2,1,1), plot(data), subplot(2,1,2), plot(DATA);

end
```

8.1.7 Epoc_Acquisition.m

```
function Epoc_Acquisition

clear all;
clc;
clf;

% constants
extensionType = sprintf('.csv');
destinationFolder = sprintf('AcquisitionData');

computerName = input('Which computer is being used? ', 's');
patient = input('What is the name of the patient? ', 's');
state = input('What is the target mental state? ', 's');
dateName = input('What is the date (MMDDYY)? ', 's');

filename = sprintf('%s/%s_%s_%s_%s', destinationFolder, computerName, patient, state,
dateName);

prompt = sprintf('echo 8 second relaxation acquisition [Press any key to continue]');
system(prompt);
kbhit();
Epoc_PreProcess(filename, extensionType);

repeat = 'y';

while(repeat == 'y')

        prompt = sprintf('echo Full 25 second acquisition [Press any key to continue]');
        system(prompt);
        kbhit();
        Epoc_Process(filename, extensionType);

        % make sure motors are off in the end
        command = sprintf('./connect 1');
        system(command);
```

```
	PostProcess(filename);

	repeat = input('Would you like to repeat the trial? [y/n]\nNote: this will erase the
data from the previous trial');
endwhile

end
```

8.1.8 Epoc_PreProcess.m

```
function Epoc_PreProcess(basename, extensionType)

% remove basename_averages.csv, basename_sums.csv, basename_references.csv,
basename_bases.csv
filename = sprintf('%s_averages%s', basename, extensionType);
command = sprintf('rm %s', filename);
system(command);

filename = sprintf('%s_sums%s', basename, extensionType);
command = sprintf('rm %s', filename);
system(command);

filename = sprintf('%s_references%s', basename, extensionType);
command = sprintf('rm %s', filename);
system(command);

filename = sprintf('%s_bases%s', basename, extensionType);
command = sprintf('rm %s', filename);
system(command);

filename = sprintf('%s_bases', basename);

% epocd doing its Acquire run taking the filename as command-line input
currentFilename = sprintf('%s_%i%s', filename, 1, extensionType);
previousFilename2 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

currentFilename = sprintf('%s_%i%s', filename, 1, extensionType);
```

```
previousFilename1 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

currentFilename = sprintf('%s_%i%s', filename, 2, extensionType);
previousFilename0 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

for i = 1:31
        currentFilename = sprintf('%s_%i%s', filename, i, extensionType);

        command = sprintf('sudo ./epocd %s', currentFilename);
        system(command);

        PreProcess1(currentFilename, previousFilename0, previousFilename1,
previousFilename2, basename);

        previousFilename0 = sprintf('%s', currentFilename);
        previousFilename1 = sprintf('%s', previousFilename0);
        previousFilename2 = sprintf('%s', previousFilename1);
end

PreProcess2(basename);

end
```

8.1.9 Epoc_Process.m

```
function Epoc_Process(basename, extensionType)

concentrationCount = 0;
samples = 100;

% control the direction of the motors
toggle_level0 = 0;
toggle_level1 = 0;
toggle_level2 = 0;
toggle_level3 = 0;
```

```
% epocd doing its Acquire run taking the filename as command-line input
currentFilename = sprintf('%s_%i%s', basename, 0, extensionType);
previousFilename2 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

currentFilename = sprintf('%s_%i%s', basename, 1, extensionType);
previousFilename1 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

currentFilename = sprintf('%s_%i%s', basename, 2, extensionType);
previousFilename0 = sprintf('%s', currentFilename);
command = sprintf('sudo ./epocd %s', currentFilename);
system(command);

for i = 3:(samples-1)

        currentFilename = sprintf('%s_%i%s', basename, i, extensionType);

        command = sprintf('sudo ./epocd %s', currentFilename);
        system(command);

        concentration = Process(currentFilename, previousFilename0, previousFilename1,
previousFilename2, basename);

        previousFilename0 = sprintf('%s', currentFilename);
        previousFilename1 = sprintf('%s', previousFilename0);
        previousFilename2 = sprintf('%s', previousFilename1);

        command = sprintf('./connect 1'); % turn off the motors at the start of every loop
        system(command);

        [toggle_level0 toggle_level1 toggle_level2 toggle_level3] =
SystemControl(concentration, toggle_level0, toggle_level1, toggle_level2, toggle_level3);

end

end
```

8.1.10 epocd.c

```
/* Emotic EPOC daemon that decrypt stream using ECB and RIJNDAEL-128 cipher
 * (well, not yet a daemon...)
 *
 * Usage: epocd (consumer/research) /dev/emotiv/encrypted output_file
 *
 * Make sure to pick the right type of device, as this determines the key
 * */

#include <stdio.h>
#include <string.h>
#include "libepoc.h"

#define headsetFrequency 128       // 2^7 Hertz
#define acquisitionTime .25// (2^2)^-1 seconds

// gathers data for appropriate amount of time, writes to file
void Acquire(char* filename, struct epoc_frame frame, epoc_device* d, char* data)
{
        // headset data
        FILE *input;
        FILE *output;

        // remove any existing file with name [filename] to avoid append errors
        remove(filename);

        // gather appropriate size of data
        int i;
        for(i = 0; i < (headsetFrequency * acquisitionTime); i++)
        {
                if(epoc_read_data(d, data) > 0)
                {
                        fflush(stdout);

                        epoc_get_next_frame(&frame, data);

                        // write out headset data in columns
                        output = fopen(filename, "a+");
        fprintf(output,"%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",
                                frame.gyroX, frame.gyroY,
                                frame.F3, frame.F4, frame.F7, frame.F8,
                                frame.FC5, frame.FC6,
                                frame.AF3, frame.AF4,
                                frame.O1, frame.O2,
                                frame.P7, frame.P8,
                                frame.T7, frame.T8);

                        fclose(output);
                }
        }
}

int main(int argc, char **argv)
{
        // constants
        const int acquisitionSamples = 100;

        char* filename;
        filename = argv[1];

        enum headset_type type;

        char raw_frame[32];
        struct epoc_frame frame;
        epoc_device* d;
        char data[32];

        // initialize headset
        d = epoc_create();

        // check for headset connected
        printf("Current epoc devices connected: %d\n", epoc_get_count(d, EPOC_VID,
EPOC_PID));
        if(epoc_open(d, EPOC_VID, EPOC_PID, 0) != 0)
        {
                printf("CANNOT CONNECT\n");
                return 1;
        }

        Acquire(filename, frame, d, data);
```

```
	// remove headset data
	epoc_close(d);
	epoc_delete(d);
	return 0;
}
```

8.1.11 PostProcess.m

```
function PostProcess(filename)


Average = sprintf('%s_averages.csv', filename);

Reference = sprintf('%s_references.csv', filename);

AverageData = dlmread(Average, ',');

ReferenceData = dlmread(Reference, ',');

figureNum = 1;

%
figureNum = PSD_ReferenceGraph('Data:', filename, AverageData, ReferenceData,
figureNum);


end
```

8.1.12 PreProcess1.m

```
%This script will find the targeted data and store it all together
function PreProcess1(currentFile, previousFile0, previousFile1, previousFile2, baseFile)

% acquisition constants (from epocd.c)
headsetFrequency = 128;
sourceFile = sprintf('%s_bases.csv', baseFile);

% EEG data
% beta waves correlate to concentration; frontal, symmetric
beta_min = 12;
beta_max = 30;
```

```
% 1. Grab data from file
% 2. Bandpass data for beta waves
% 3. Find the sum and average of each channel

% 1]
% data acquired from input file
[GYROXdata GYROYdata F3data F4data F7data F8data FC5data FC6data AF3data AF4data
O1data O2data P7data P8data T7data T8data] = splitData(currentFile,previousFile0,
previousFile1, previousFile2);

% 2]
[GYROXband GYROYband F3band F4band F7band F8band FC5band FC6band
AF3band AF4band O1band O2band P7band P8band T7band T8band] =
bandpassData(headsetFrequency, beta_min, beta_max,GYROXdata, GYROYdata, F3data,
F4data, F7data, F8data, FC5data, FC6data, AF3data, AF4data, O1data, O2data, P7data,
P8data, T7data, T8data);

% 3]
[GYROX GYROY F3 F4 F7 F8 FC5 FC6 AF3 AF4 O1 O2 P7 P8 T7 T8] =
avgPSD_Data(GYROXband, GYROYband, F3band, F4band, F7band, F8band, FC5band,
FC6band, AF3band, AF4band, O1band, O2band, P7band, P8band, T7band, T8band);

writer = fopen(sourceFile, 'a');
fprintf(writer, '%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n', GYROX, GYROY, F3,
F4, F7, F8, FC5, FC6, AF3, AF4, O1, O2, P7, P8, T7, T8);
fclose(writer);



end
```

8.1.13 PreProcess2

```
%This script will find the targeted data and store it all together
function PreProcess2(baseFile)

sourceFile = sprintf('%s_bases.csv', baseFile);
referenceFile = sprintf('%s_references.csv', baseFile);
referenceData = dlmread(sourceFile, ',');
```

```
% split into three part vectors of median, standard deviation, and variance
[GYROXdata GYROYdata F3data F4data F7data F8data FC5data FC6data AF3data AF4data
O1data O2data P7data P8data T7data T8data] = PreProcessData(referenceData);

GYROXdata = [0.0 0.0 0.0];
GYROYdata = [0.0 0.0 0.0];


writer = fopen(referenceFile, 'a');
for i=1:3
   %printf('%i\n', i);
   fprintf(writer, '%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n', GYROXdata(i),
GYROYdata(i), F3data(i), F4data(i), F7data(i), F8data(i), FC5data(i), FC6data(i), AF3data(i),
AF4data(i), O1data(i), O2data(i), P7data(i), P8data(i), T7data(i), T8data(i));

end
fclose(writer);

end
```

8.1.14 PreProcessData.m

```
function[GYROXdata GYROYdata F3data F4data F7data F8data FC5data FC6data AF3data
AF4data O1data O2data P7data P8data T7data T8data] = PreProcessData(referenceData)

% channel locations in the data
GYROX = 1;
GYROY = 2;
F3 = 3;
F4 = 4;
F7 = 5;
F8 = 6;
FC5 = 7;
FC6 = 8;
AF3 = 9;
AF4 = 10;
O1 = 11;
O2 = 12;
P7 = 13;
P8 = 14;
```

```
T7 = 15;
T8 = 16;

% reference the data to mean, std, var
[GYROXdata(1) GYROXdata(2) GYROXdata(3)] = ReferenceData(referenceData(:,GYROX));
[GYROYdata(1) GYROYdata(2) GYROYdata(3)] = ReferenceData(referenceData(:,GYROY));
[F3data(1) F3data(2) F3data(3)] = ReferenceData(referenceData(:,F3));
[F4data(1) F4data(2) F4data(3)] = ReferenceData(referenceData(:,F4));
[F7data(1) F7data(2) F7data(3)] = ReferenceData(referenceData(:,F7));
[F8data(1) F8data(2) F8data(3)] = ReferenceData(referenceData(:,F8));
[FC5data(1) FC5data(2) FC5data(3)] = ReferenceData(referenceData(:,FC5));
[FC6data(1) FC6data(2) FC6data(3)] = ReferenceData(referenceData(:,FC6));
[AF3data(1) AF3data(2) AF3data(3)] = ReferenceData(referenceData(:,AF3));
[AF4data(1) AF4data(2) AF4data(3)] = ReferenceData(referenceData(:,AF4));
[O1data(1) O1data(2) O1data(3)] = ReferenceData(referenceData(:,O1));
[O2data(1) O2data(2) O2data(3)] = ReferenceData(referenceData(:,O2));
[P7data(1) P7data(2) P7data(3)] = ReferenceData(referenceData(:,P7));
[P8data(1) P8data(2) P8data(3)] = ReferenceData(referenceData(:,P8));
[T7data(1) T7data(2) T7data(3)] = ReferenceData(referenceData(:,T7));
[T8data(1) T8data(2) T8data(3)] = ReferenceData(referenceData(:,T8));

end
```

8.1.15 Process.m

```
%This script will find the targeted data and store it all together
function [concentration] = Process(currentFile, previousFile0, previousFile1,
previousFile2, baseFile)

% acquisition constants (from epocd.c)
headsetFrequency = 128;
averageFile = sprintf('%s_averages.csv', baseFile);
referenceFile = sprintf('%s_references.csv', baseFile);

% EEG data
% beta waves correlate to concentration; frontal, symmetric
beta_min = 12;
beta_max = 30;

% 1. Grab data from file
% 2. Bandpass data for beta waves
% 3. Find the referenced average power of each channel & determine if concentrating
% 4. Write data to file
% 5. Determine if concentrating

% 1]
% data acquired from input file
[GYROXdata GYROYdata F3data F4data F7data F8data FC5data FC6data AF3data AF4data
O1data O2data P7data P8data T7data T8data] = splitData(currentFile,previousFile0,
previousFile1, previousFile2);

% 2]
[GYROXband GYROYband F3band F4band F7band F8band FC5band FC6band
AF3band AF4band O1band O2band P7band P8band T7band T8band] =
bandpassData(headsetFrequency, beta_min, beta_max,GYROXdata, GYROYdata, F3data,
F4data, F7data, F8data, FC5data, FC6data, AF3data, AF4data, O1data, O2data, P7data,
P8data, T7data, T8data);

% 3]
[GYROX GYROY F3 F4 F7 F8 FC5 FC6 AF3 AF4 O1 O2 P7 P8 T7 T8 concentration] =
ReferencedAverage(referenceFile,GYROXband, GYROYband, F3band, F4band, F7band,
F8band, FC5band, FC6band, AF3band, AF4band, O1band, O2band, P7band, P8band,
T7band, T8band);

% 4]
writer = fopen(averageFile, 'a');
fprintf(writer, '%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f\n', GYROX, GYROY, F3,
F4, F7, F8, FC5, FC6, AF3, AF4, O1, O2, P7, P8, T7, T8);
fclose(writer);



end
```

8.1.16 PSD_ReferenceGraph.m

```
function figureNum = PSD_ReferenceGraph(state, person, averageData, referenceData,
figureNum);

% channel locations in the data
GYROX = 1;
GYROY = 2;
```

```
F3 = 3;
F4 = 4;
F7 = 5;
F8 = 6;
FC5 = 7;
FC6 = 8;
AF3 = 9;
AF4 = 10;
O1 = 11;
O2 = 12;
P7 = 13;
P8 = 14;
T7 = 15;
T8 = 16;

% AF channels

% AF3
% produce a graph with the limit shown over
% the data gathered

limit = referenceData(2,AF3);
limitData = zeros(rows(averageData));
for i = 1:rows(averageData)
        limitData(i) = limit;
end

figure(figureNum)
figureNum = figureNum  + 1;

hold on;
plot(1:.25:25,averageData(:,AF3), 'k-');
plot(limitData*.5, 'r-');
plot(limitData, 'b-');
plot(limitData*1.5, 'g-');

titlestring = sprintf('AF3 %s - %s', state, person);
title(titlestring);
legend('AF3', 'limit1', 'limit2', 'limit3');
axis([0 25 0 60000])
```

```
xlabel('Time (s)');
ylabel('mean(PSD(bandpassed data))');

% AF4
% produce a graph with the limit shown over
% the data gathered

limit = referenceData(2,AF4);
limitData = zeros(rows(averageData));
for i = 1:rows(averageData)
        limitData(i) = limit;
end


figure(figureNum)
figureNum = figureNum  + 1;

hold on;
plot(1:.25:25,averageData(:,AF4), 'k-');
plot(limitData, 'r-');
plot(limitData*.5, 'r-');
plot(limitData, 'b-');
plot(limitData*1.5, 'g-');

titlestring = sprintf('AF4 %s - %s', state, person);
title(titlestring);
legend('AF4', 'limit1', 'limit2', 'limit3');
axis([0 25 0 60000])
xlabel('Time (s)');
ylabel('mean(PSD(bandpassed data))');
end
```

8.1.17 refAvg.m

```
function [result] = refAvg(data, reference)
% finds the average of the power
result = avgPSD(data);

% based on reference
result = max(result - reference,0);
```

```
end
```

8.1.18 ReferenceData.m

```
function [avg stddev variance] = ReferenceData(data)

avg = mean(data);
stddev = std(data);
variance = stddev * stddev;

end
```

8.1.19 ReferencedAverage.m

```
function [GYROX GYROY F3 F4 F7 F8 FC5 FC6 AF3 AF4 O1 O2 P7 P8 T7 T8 concentration] =
ReferencedAverage(filename,GYROXband, GYROYband, F3band, F4band, F7band, F8band,
FC5band, FC6band, AF3band, AF4band, O1band, O2band, P7band, P8band, T7band,
T8band)

referenceData = dlmread(filename, ',');

GYROX = refAvg(GYROXband(:),referenceData(1,1));
GYROY = refAvg(GYROYband(:),referenceData(1,2));
F3 = refAvg(F3band(:),referenceData(1,3));
F4 = refAvg(F4band(:),referenceData(1,4));
F7 = refAvg(F7band(:),referenceData(1,5));
F8 = refAvg(F8band(:),referenceData(1,6));
FC5 = refAvg(FC5band(:),referenceData(1,7));
FC6 = refAvg(FC6band(:),referenceData(1,8));
AF3 = refAvg(AF3band(:),referenceData(1,9));
AF4 = refAvg(AF4band(:),referenceData(1,10));
O1 = refAvg(O1band(:),referenceData(1,11));
O2 = refAvg(O2band(:),referenceData(1,12));
P7 = refAvg(P7band(:),referenceData(1,13));
P8 = refAvg(P8band(:),referenceData(1,14));
T7 = refAvg(T7band(:),referenceData(1,15));
T8 = refAvg(T8band(:),referenceData(1,16));

% determine concentration
concentration = false;
limitData = 2; % standard deviation
```

```
limitLevel1 = .5;
limitLevel2 = 1.0;
limitLevel3 = 1.5;

% single-channel concentration test
if(AF3 > limitLevel3*referenceData(limitData,9))
        AF3_concentration = .8;
elseif(AF3 > limitLevel2*referenceData(limitData,9))
        AF3_concentration = .6;
elseif(AF3 > limitLevel1*referenceData(limitData,9))
        AF3_concentration = .4;
else
        AF3_concentration = 0;
end

if(AF4 > limitLevel3*referenceData(limitData,10))
        AF4_concentration = .8;
elseif(AF4 > limitLevel2*referenceData(limitData,10))
        AF4_concentration = .6;
elseif(AF4 > limitLevel1*referenceData(limitData,10))
        AF4_concentration = .4;
else
        AF4_concentration = 0;
end

concentration = max(AF3_concentration, AF4_concentration);
%{
if concentration > 0
        yell = sprintf('\n~~~~~~~~~~~~~~~~~~~~~~~~~~\nCONCENTRATING!
\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n');
        printf('%s', yell);
        command = sprintf('./connect w');
        system(command);
end
%}
end
```

8.1.20 splitData.m

```
function [GYROXdata, GYROYdata, F3data, F4data, F7data, F8data, FC5data, FC6data,
```

```
AF3data, AF4data, O1data, O2data, P7data, P8data, T7data, T8data] = splitData(currentFile,
previousFile0, previousFile1, previousFile2)


acquisitionTime = .25;
headsetFrequency = 128;
stop = headsetFrequency*acquisitionTime;


%The script will then split that matrix into its components: the 4 'F'
%   channels, the 2 'FC' channels, the 2 'AF' channels, the 2 'O' channels,
%   the 2 'P' channels, and the 2 'T' channels.

% channel locations in the data
GYROX = 1;
GYROY = 2;
F3 = 3;
F4 = 4;
F7 = 5;
F8 = 6;
FC5 = 7;
FC6 = 8;
AF3 = 9;
AF4 = 10;
O1 = 11;
O2 = 12;
P7 = 13;
P8 = 14;
T7 = 15;
T8 = 16;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%
%  P2 P1 P0 C
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
% P2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

%Get the data into an array
data = dlmread(previousFile2, ',');

%Transpose the matrix
data2 = data';

%Split the matrix data2 into its 6 parts
GYROXdata(1:stop) = data2(GYROX,:);
GYROYdata(1:stop) = data2(GYROY,:);
F3data(1:stop) = data2(F3,:);
F4data(1:stop) = data2(F4,:);
F7data(1:stop) = data2(F7,:);
F8data(1:stop) = data2(F8,:);
FC5data(1:stop) = data2(FC5,:);
FC6data(1:stop) = data2(FC6,:);
AF3data(1:stop) = data2(AF3,:);
AF4data(1:stop) = data2(AF4,:);
O1data(1:stop) = data2(O1,:);
O2data(1:stop) = data2(O2,:);
P7data(1:stop) = data2(P7,:);
P8data(1:stop) = data2(P8,:);
T7data(1:stop) = data2(T7,:);
T8data(1:stop) = data2(T8,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% P1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%Get the data into an array
data = dlmread(previousFile1, ',');

%Transpose the matrix
```

```
data2 = data';

%Split the matrix data2 into its 6 parts
GYROXdata(stop+1:stop*2) = data2(GYROX,:);
GYROYdata(stop+1:stop*2) = data2(GYROY,:);
F3data(stop+1:stop*2) = data2(F3,:);
F4data(stop+1:stop*2) = data2(F4,:);
F7data(stop+1:stop*2) = data2(F7,:);
F8data(stop+1:stop*2) = data2(F8,:);
FC5data(stop+1:stop*2) = data2(FC5,:);
FC6data(stop+1:stop*2) = data2(FC6,:);
AF3data(stop+1:stop*2) = data2(AF3,:);
AF4data(stop+1:stop*2) = data2(AF4,:);
O1data(stop+1:stop*2) = data2(O1,:);
O2data(stop+1:stop*2) = data2(O2,:);
P7data(stop+1:stop*2) = data2(P7,:);
P8data(stop+1:stop*2) = data2(P8,:);
T7data(stop+1:stop*2) = data2(T7,:);
T8data(stop+1:stop*2) = data2(T8,:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
% P0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%

%Get the data into an array
data = dlmread(previousFile0, ',');

%Transpose the matrix
data2 = data';

%Split the matrix data2 into its 6 parts
GYROXdata(1+(stop*2):stop*3) = data2(GYROX,:);
GYROYdata(1+(stop*2):stop*3) = data2(GYROY,:);
F3data(1+(stop*2):stop*3) = data2(F3,:);
F4data(1+(stop*2):stop*3) = data2(F4,:);
F7data(1+(stop*2):stop*3) = data2(F7,:);
F8data(1+(stop*2):stop*3) = data2(F8,:);
```

```
FC5data(1+(stop*2):stop*3) = data2(FC5,:);
FC6data(1+(stop*2):stop*3) = data2(FC6,:);
AF3data(1+(stop*2):stop*3) = data2(AF3,:);
AF4data(1+(stop*2):stop*3) = data2(AF4,:);
O1data(1+(stop*2):stop*3) = data2(O1,:);
O2data(1+(stop*2):stop*3) = data2(O2,:);
P7data(1+(stop*2):stop*3) = data2(P7,:);
P8data(1+(stop*2):stop*3) = data2(P8,:);
T7data(1+(stop*2):stop*3) = data2(T7,:);
T8data(1+(stop*2):stop*3) = data2(T8,:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% C
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%


%Get the data into an array
data = dlmread(currentFile, ',');

%Transpose the matrix
data2 = data';

%Split the matrix data2 into its 6 parts
GYROXdata(1+(stop*3):stop*4) = data2(GYROX,:);
GYROYdata(1+(stop*3):stop*4) = data2(GYROY,:);
F3data(1+(stop*3):stop*4) = data2(F3,:);
F4data(1+(stop*3):stop*4) = data2(F4,:);
F7data(1+(stop*3):stop*4) = data2(F7,:);
F8data(1+(stop*3):stop*4) = data2(F8,:);
FC5data(1+(stop*3):stop*4) = data2(FC5,:);
FC6data(1+(stop*3):stop*4) = data2(FC6,:);
AF3data(1+(stop*3):stop*4) = data2(AF3,:);
AF4data(1+(stop*3):stop*4) = data2(AF4,:);
O1data(1+(stop*3):stop*4) = data2(O1,:);
O2data(1+(stop*3):stop*4) = data2(O2,:);
P7data(1+(stop*3):stop*4) = data2(P7,:);
P8data(1+(stop*3):stop*4) = data2(P8,:);
```

```
T7data(1+(stop*3):stop*4) = data2(T7,:);
T8data(1+(stop*3):stop*4) = data2(T8,:);

end
```

8.1.21 SystemControl.m

```
function [toggle_level0 toggle_level1 toggle_level2 toggle_level3] =
SystemControl(concentrationLevel, toggle_level0, toggle_level1, toggle_level2,
toggle_level3)

% define how active the patient has been concentrating
% by what percent of the time the patient is concentrating
level0 = 0.0;
level1 = 0.25;
level2 = 0.50;
level3 = 0.75;

toggleLimit = 4;
toggleChange = toggleLimit / 2;

% evaluate concentration level and direction of motion
if(concentrationLevel > level3)
        toggle_level3 = toggle_level3 + 1;

        if(toggle_level3 >= toggleLimit)
                toggle_level3 = 0;
        end

        if(toggle_level3 >= toggleChange)
                command = sprintf('./connect d');
        else
                command = sprintf('./connect c');
        end

        system(command);
end
if(concentrationLevel > level2)
        toggle_level2 = toggle_level2 + 1;

        if(toggle_level2 >= toggleLimit)
        toggle_level2 = 0;
    end

    if(toggle_level2 >= toggleChange)
        command = sprintf('./connect f');
    else
        command = sprintf('./connect v');
    end

    system(command);
end
if(concentrationLevel > level1)
    toggle_level1 = toggle_level1 + 1;

    if(toggle_level1 >= toggleLimit)
        toggle_level1 = 0;
    end

    if(toggle_level1 > toggleChange)
        command = sprintf('./connect g');
    else
        command = sprintf('./connect b');
    end

    system(command);
end

command = sprintf('');





end
```

8.1.22 usb.cpp

```
#include <stdio.h>   // standard input / output functions
#include <stdlib.h>  // general standard library functions
#include <unistd.h>  // UNIX standard function definitions
#include <fcntl.h>   // File control definitions
```

```
#include <termios.h> // POSIX terminal control definitions
#include <string.h>


int open_port(int device);
int configure_port(int fd);
int read_lines(int fd, char* buf, int lines);

int main(int argc, char* argv[])
{
  // usbtty0
  int fd0 = open_port(0);
  fd0 = configure_port(fd0);

  // usbtty1
  int fd1 = open_port(1);
  fd1 = configure_port(fd1);

  int position;
  char* input;
  input = argv[1];

  char buffer[255];
  char command[32];
  sprintf(command, "%s", input);
  //printf("sending %s\n", command);

  // signals to ttyusb0
  ////////////////////////////////////////////////////////////////
  write(fd0, command,strlen(command));
  position = read_lines(fd0, buffer, 255); // get next 3 lines

  if (0 >= position)
  {
    printf("\nPort closed unexpectedly after %d characters, exiting.\n",
         -position);
    exit(-1);
  }
  else
  {
```

```
            //printf("GOT: %s\n", buffer);
  }
  close(fd0);
  ///////////////////////////////////////////////


  // signals to ttyusb1
  //////////////////////////////////////////////////////////
  write(fd1, command,strlen(command));
  position = read_lines(fd1, buffer, 255); // get next 3 lines

  if (0 >= position)
  {
    printf("\nPort closed unexpectedly after %d characters, exiting.\n",
         -position);
    exit(-1);
  }
  else
  {
            //printf("GOT: %s\n", buffer);
  }
  close(fd1);
  //////////////////////////////////////////////////////////////

  return 0;
}

/* read a set number of lines of text from the port */
int read_lines(int fd, char* buf, int lines)
{
  int count;
  int line, row = 0;

  for (line = 0; lines < lines; line++)
  {
    for (; buf[row] != '\n'; row++)
    {
      if (0 >= read(fd, (buf + row), 1))  // port closed for some reason
        return -count;
      else
      {
        putchar(buf[row]);
        count++;
      }
    }
  }
  return count;
}

/*Function to open a serial port*/
int open_port(int device)
{
  int fd; // file description for the serial port

  //fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NONBLOCK);
  if(device == 0)
  {
        fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);
  }
  else
  {
        fd = open("/dev/ttyUSB1", O_RDWR | O_NOCTTY | O_NDELAY);
  }


  if(fd == -1) // if open is unsucessful
  {
    perror("open_port: Unable to open /dev/ttyS0 - ");
  }
  else
  {
    //fcntl(fd, F_SETFL, O_NONBLOCK);
    fcntl(fd, F_SETFL, 0);
  }

  return(fd);
}

/*Port configuration - constant BAUD rate B57600*/
int configure_port(int fd)
```

```
{
  struct termios port_settings;      // structure to store the port settings in

  tcgetattr(fd, &port_settings);
  cfsetispeed(&port_settings, B57600);    // set baud rates
  cfsetospeed(&port_settings, B57600);

  port_settings.c_cflag |= (CLOCAL | CREAD);

  port_settings.c_cflag &= ~PARENB;    // set no parity, stop bits, data bits
  port_settings.c_cflag &= ~CSTOPB;
  port_settings.c_cflag &= ~CSIZE;
  port_settings.c_cflag |= CS8;

  //options.c_cflag &= ~CRTSCTS;  // disable hardware flow control
  //port_settings.c_lflag |= (ICANON | ECHO | ECHOE);
  port_settings.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

  tcsetattr(fd, TCSANOW, &port_settings);    // apply the settings to the port

  return(fd);
}
```

8.2 Microcontroller Code

8.2.1 micro_leddriver.c

```
#ifndef _TKIUART_H_
#define _TKIUART_H_

void putc(char value);
char getc(void);
void init_usart(unsigned short rate);
void puts(char *strn);
void putd(unsigned short dat);
unsigned short read_short(void);

#endif // _uart_motordriver_H_
```

8.2.2 micro_leddriver.h

```
#ifndef _microleddriver_H_
#define _microleddriver_H_
```

```
#define KEY_D 100
#define KEY_C 99
#define KEY_F 102
#define KEY_V 118
#define KEY_G 103
#define KEY_B 98

#define KEY_1 49
#define KEY_2 50
#define KEY_3 51

#define STATE_OFF 0
#define STATE_ON1 1
#define STATE_ON2 2
#define STATE_ON3 3

#define LED0 0
#define LED1 1
#define LED2 2
#define LED3 3

#define TOTAL_LEDS 4

void init_LED(void);
void control_LEDs(char input);
void setLED_state(char LED, char state);
void setAllOff(void);

void setLED0(char state);
void setLED1(char state);
void setLED2(char state);
void setLED3(char state);

#endif
```

8.2.3 TKI_Main_LED.c

```
/* TKI
 * main file
 */
```

```
#include <system.h>

#include "TKI_UART.h"
#include "micro_leddriver.h"
////////////////////////////////////////////


                                                                  // Oscillator Configuration see p.
36 PIC18F47J53 Fam
// CONFIG1L
#pragma config WDTEN = OFF      // Watchdog Timer (Disabled - Controlled by SWDTEN
bit)
#pragma config PLLDIV = 5       // PLL Prescaler Selection (Divide by 5 (20 MHz oscillator
input))
#pragma config CFGPLLEN = ON    // PLL Enable Configuration Bit (PLL Enabled)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset (Enabled)
#pragma config XINST = OFF      // Extended Instruction Set (Disabled)

// CONFIG1H
#pragma config CPUDIV = OSC1    // CPU System Clock Postscaler (used to be CPUDIV =
OSC1)     page 41 PIC18F47J53 Fam;
#pragma config CP0 = OFF        // Code Protect (Program memory is not code-protected)

// CONFIG2L
#pragma config OSC = INTOSCPLL  // Oscillator (INTOSCPLL)
//#pragma config SOSCSEL = HIGH   // T1OSC/SOSC Power Selection Bits (High Power
T1OSC/SOSC circuit selected)
#pragma config CLKOEC = OFF     // EC Clock Out Enable Bit  (CLKO output disabled on the
RA6 pin)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor (Enabled)
//#pragma config IESO = ON       // Internal External Oscillator Switch Over Mode
(Enabled)

// CONFIG2H
//#pragma config WDTPS = 32768    // Watchdog Postscaler (1:32768)

// CONFIG3L
//#pragma config DSWDTOSC = INTOSCREF// DSWDT Clock Select (DSWDT uses INTRC)
//#pragma config RTCOSC = T1OSCREF// RTCC Clock Select (RTCC uses T1OSC/T1CKI)
```

```
//#pragma config DSBOREN = ON     // Deep Sleep BOR (Enabled)
//#pragma config DSWDTEN = OFF    // Deep Sleep Watchdog Timer (Disabled)
//#pragma config DSWDTPS = G2     // Deep Sleep Watchdog Postscaler (1:2,147,483,648
(25.7 days))

// CONFIG3H
//#pragma config IOL1WAY = ON     // IOLOCK One-Way Set Enable bit (The IOLOCK bit
(PPSCON<0>) can be set once)
//#pragma config ADCSEL = BIT10   // ADC 10 or 12 Bit Select (10 - Bit ADC Enabled)
//#pragma config MSSP7B_EN = MSK7 // MSSP address masking (7 Bit address masking
mode)

// CONFIG4L
//#pragma config WPFP = PAGE_127  // Write/Erase Protect Page Start/End Location
(Write Protect Program Flash Page 127)
//#pragma config WPCFG = OFF      // Write/Erase Protect Configuration Region
(Configuration Words page not erase/write-protected)

// CONFIG4H
#pragma config WPDIS = OFF      // Write Protect Disable bit (WPFP<6:0>/WPEND region
ignored)
#pragma config WPEND = PAGE_WPFP// Write/Erase Protect Region Select bit (valid
when WPDIS = 0) (Pages WPFP<6:0> through Configuration Words erase/write protected)
#pragma config LS48MHZ = SYS48X8// Low Speed USB mode with 48 MHz system clock
bit (System clock at 48 MHz USB CLKEN divide-by is set to 8)


#pragma CLOCK_FREQ 20000000

void main(void)
{
        init_usart(86);

        init_LED();

  do
  {
    control_LEDs(getc());

  }while(true);
}
```

8.2.4 TKI_Main_Motor.c

```
/* TKI
 * main file
 */

#include <system.h>

#include "TKI_MotorControl.h"
#include "TKI_UART.h"
////////////////////////////////////////////////


                                                                        // Oscillator Configuration see p.
36 PIC18F47J53 Fam
// CONFIG1L
#pragma config WDTEN = OFF      // Watchdog Timer (Disabled - Controlled by SWDTEN
bit)
#pragma config PLLDIV = 5       // PLL Prescaler Selection (Divide by 5 (20 MHz oscillator
input))
#pragma config CFGPLLEN = ON    // PLL Enable Configuration Bit (PLL Enabled)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset (Enabled)
#pragma config XINST = OFF      // Extended Instruction Set (Disabled)

// CONFIG1H
#pragma config CPUDIV = OSC1    // CPU System Clock Postscaler (used to be CPUDIV =
OSC1)     page 41 PIC18F47J53 Fam;
#pragma config CP0 = OFF        // Code Protect (Program memory is not code-protected)

// CONFIG2L
#pragma config OSC = INTOSCPLL  // Oscillator (INTOSCPLL)
//#pragma config SOSCSEL = HIGH   // T1OSC/SOSC Power Selection Bits (High Power
T1OSC/SOSC circuit selected)
#pragma config CLKOEC = OFF     // EC Clock Out Enable Bit  (CLKO output disabled on the
RA6 pin)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor (Enabled)
//#pragma config IESO = ON       // Internal External Oscillator Switch Over Mode
(Enabled)

// CONFIG2H
```

```
//#pragma config WDTPS = 32768    // Watchdog Postscaler (1:32768)

// CONFIG3L
//#pragma config DSWDTOSC = INTOSCREF// DSWDT Clock Select (DSWDT uses INTRC)
//#pragma config RTCOSC = T1OSCREF// RTCC Clock Select (RTCC uses T1OSC/T1CKI)
//#pragma config DSBOREN = ON     // Deep Sleep BOR (Enabled)
//#pragma config DSWDTEN = OFF    // Deep Sleep Watchdog Timer (Disabled)
//#pragma config DSWDTPS = G2     // Deep Sleep Watchdog Postscaler (1:2,147,483,648
(25.7 days))

// CONFIG3H
//#pragma config IOL1WAY = ON     // IOLOCK One-Way Set Enable bit (The IOLOCK bit
(PPSCON<0>) can be set once)
//#pragma config ADCSEL = BIT10   // ADC 10 or 12 Bit Select (10 - Bit ADC Enabled)
//#pragma config MSSP7B_EN = MSK7 // MSSP address masking (7 Bit address masking
mode)

// CONFIG4L
//#pragma config WPFP = PAGE_127  // Write/Erase Protect Page Start/End Location
(Write Protect Program Flash Page 127)
//#pragma config WPCFG = OFF      // Write/Erase Protect Configuration Region
(Configuration Words page not erase/write-protected)

// CONFIG4H
#pragma config WPDIS = OFF      // Write Protect Disable bit (WPFP<6:0>/WPEND region
ignored)
#pragma config WPEND = PAGE_WPFP// Write/Erase Protect Region Select bit (valid
when WPDIS = 0) (Pages WPFP<6:0> through Configuration Words erase/write protected)
#pragma config LS48MHZ = SYS48X8// Low Speed USB mode with 48 MHz system clock
bit (System clock at 48 MHz USB CLKEN divide-by is set to 8)

#pragma CLOCK_FREQ 20000000

void main(void)
{
	init_usart(86);

	init_motor();
	enable_motor();
```

```
  do
  {
   control_motors(getc());
  }while(true);
}
```

8.2.5 TKI_MotorControl.c

```
#include <system.h>

#include "TKI_MotorControl.h"
#include <stdlib.h>

void init_motor(void)
{
        trise = 0;      // port e to output (motor driver)
        trisa = 0;      // port a to output (motor driver)
        trisc.0 = 0;   // RC0 to output (motor driver)
        trisc.1 = 0;      //RC1 to input (LED)RCX
        trisc.2 = 0;
        trisb.0 = 0;      //port b to output
        trisb.1 = 0;
        trisb.2 = 0;
        trisd.3 = 0;      //RD3 to output
        trisd.2 = 0;

        enable_motor();

        setAllBrake();
}

void enable_motor(void)
{
        latb.0 = 1;//RBO Enable motor 5
        lata.0 = 1;//RA0 Enable motor 4
        latd.3 = 1;//RD3 Emable motor 3
        late.2 = 1;//RE2 Enable motor 2
        lata.3 = 1;//RA3 Enable motor 1
}

void setAllBrake()
```

```
{
                set_motor(M1, MOTOR_BREAK);
                set_motor(M2, MOTOR_BREAK);
                set_motor(M3, MOTOR_BREAK);
                set_motor(M4, MOTOR_BREAK);
                set_motor(M5, MOTOR_BREAK);
                setLED(LED_OFF);
}

void control_motors(char input)
{
        // 1: set all to break
        if(input == KEY_1)
        {
                setAllBrake();
        } // 2: set all to forward
        else if (input == KEY_2)
        {
                set_motor(M1, MOTOR_FORWARD);
                set_motor(M2, MOTOR_FORWARD);
                set_motor(M3, MOTOR_FORWARD);
                set_motor(M4, MOTOR_FORWARD);
                set_motor(M5, MOTOR_FORWARD);
        } // 3: set all to reverse
        else if (input == KEY_3)
        {
                set_motor(M1, MOTOR_REVERSE);
                set_motor(M2, MOTOR_REVERSE);
                set_motor(M3, MOTOR_REVERSE);
                set_motor(M4, MOTOR_REVERSE);
                set_motor(M5, MOTOR_REVERSE);
        }
        /* keyboard motor controls:
        // column corresponds to motor
        // row1: break
        // row2: forward
        // row3: reverse */

        // M1
        else if (input == KEY_A || input == KEY_Q || input == KEY_Z)
```

```
	{
		controlM1(input);
	} // M2
	else if (input == KEY_W || input == KEY_S || input == KEY_X)
	{
		controlM2(input);
	} // M3
	else if (input == KEY_E || input == KEY_D || input == KEY_C)
	{
		controlM3(input);
	} // M4
	else if (input == KEY_R || input == KEY_F || input == KEY_V)
	{
		controlM4(input);
	} // M5
	else if (input == KEY_T || input == KEY_G || input == KEY_B)
	{
		controlM5(input);
	}
	else if(input == KEY_L || KEY_O)
	{
		controlLED(input);
	}
}

void set_motor(char motor, char state)
{
	if(motor == M1)
	{
		setM1(state);
	}
	else if(motor == M2)
	{
		setM2(state);
	}
	else if(motor == M3)
	{
		setM3(state);
	}
	else if(motor == M4)
	{
		setM4(state);
	}
	else if(motor == M5)
	{
		setM5(state);
	}
}

void setLED(char state)
{
	if(state == LED_OFF)
	{
		latc.1 = 0;
	}
	else if(state == LED_ON)
	{
		latc.1 = 1;
	}
}

void setM1(char state)
{
	if(state == MOTOR_BREAK) // break
	{
		latc.0 = 0;//RC0
		lata.5 = 0;//RA5
	}
	else if (state == MOTOR_FORWARD) // forward
	{
		latc.0 = 0;//RC0
		lata.5 = 1;//RA5
	}
	else if (state == MOTOR_REVERSE) // reverse
	{
		latc.0 = 1;//RC0
		lata.5 = 0;//RA5
	}
}
```

```
void setM2(char state)
{
        if(state == MOTOR_BREAK) // break
        {
                late.1 = 0;
                late.0 = 0;
        }
        else if (state == MOTOR_FORWARD) // forward
        {
                late.1 = 0;
                late.0 = 1;
        }
        else if (state == MOTOR_REVERSE) // reverse
        {
                late.1 = 1;
                late.0 = 0;
        }
}

void setM3(char state)
{
        if(state == MOTOR_BREAK) // break
        {
                latd.2 = 0;
                latc.2 = 0;
        }
        else if (state == MOTOR_FORWARD) // forward
        {
                latd.2 = 0;
                latc.2 = 1;
        }
        else if (state == MOTOR_REVERSE) // reverse
        {
                latd.2 = 1;
                latc.2 = 0;
        }
}

void setM4(char state)
{
		if(state == MOTOR_BREAK) // break
		{
				lata.1 = 0;
				lata.2 = 0;
		}
		else if (state == MOTOR_FORWARD) // forward
		{
				lata.1 = 0;
				lata.2 = 1;
		}
		else if (state == MOTOR_REVERSE) // reverse
		{
				lata.1 = 1;
				lata.2 = 0;
		}
}

void setM5(char state)
{
		if(state == MOTOR_BREAK) // break
		{
				latb.1 = 0;
				latb.2 = 0;
		}
		else if (state == MOTOR_FORWARD) // forward
		{
				latb.1 = 0;
				latb.2 = 1;
		}
		else if (state == MOTOR_REVERSE) // reverse
		{
				latb.1 = 1;
				latb.2 = 0;
		}
}

void controlLED(char input)
{
		if(input == KEY_L)
		{
        setLED(LED_ON);
    }
    else if (input == KEY_O)
    {
        setLED(LED_OFF);
    }
}

void controlM1(char input)
{
    if (input == KEY_Q)
    {
        set_motor(M1, MOTOR_BREAK);
    }
    else if (input == KEY_A)
    {
        set_motor(M1, MOTOR_FORWARD);
    }
    else if (input == KEY_Z)
    {
        set_motor(M1, MOTOR_REVERSE);
    }
}

void controlM2(char input)
{
    if (input == KEY_W)
    {
        set_motor(M2, MOTOR_BREAK);
    }
    else if (input == KEY_S)
    {
        set_motor(M2, MOTOR_FORWARD);
    }
    else if (input == KEY_X)
    {
        set_motor(M2, MOTOR_REVERSE);
    }
}
```

```
void controlM3(char input)
{
	if (input == KEY_E)
	{
		set_motor(M3, MOTOR_BREAK);
	}
	else if (input == KEY_D)
	{
		set_motor(M3, MOTOR_FORWARD);
	}
	else if (input == KEY_C)
	{
		set_motor(M3, MOTOR_REVERSE);
	}
}

void controlM4(char input)
{
	if (input == KEY_R)
	{
		set_motor(M4, MOTOR_BREAK);
	}
	else if (input == KEY_F)
	{
		set_motor(M4, MOTOR_FORWARD);
	}
	else if (input == KEY_V)
	{
		set_motor(M4, MOTOR_REVERSE);
	}
}

void controlM5(char input)
{
	if (input == KEY_T)
	{
		set_motor(M5, MOTOR_BREAK);
	}
	else if (input == KEY_G)
	{
				set_motor(M5, MOTOR_FORWARD);
		}
		else if (input == KEY_B)
		{
				set_motor(M5, MOTOR_REVERSE);
		}
}
```

8.2.6 TKI_MotorControl.h

```
#ifndef _TKIMOTORCONTROL_H_
#define _TKIMOTORCONTROL_H_

#define M1 1
#define M2 2
#define M3 3
#define M4 4
#define M5 5

#define LED_OFF 0
#define LED_ON 1

#define MOTOR_BREAK 0
#define MOTOR_FORWARD 1
#define MOTOR_REVERSE 2

#define KEY_1 49
#define KEY_2 50
#define KEY_3 51

#define KEY_Q 113
#define KEY_A 97
#define KEY_Z 122
#define KEY_W 119
#define KEY_S 115
#define KEY_X 120
#define KEY_E 101
#define KEY_D 100
#define KEY_C 99
#define KEY_R 114
#define KEY_F 102
#define KEY_V 118
```

```
#define KEY_T 116
#define KEY_G 103
#define KEY_B 98

#define KEY_L 108
#define KEY_O 111

void control_motors(char input);
void init_motor(void);
void enable_motor(void);
void setAllBrake(void);

void set_motor(char motor, char state);

void setLED(char state);
void setM1(char state);
void setM2(char state);
void setM3(char state);
void setM4(char state);
void setM5(char state);

void controlLED(char input);
void controlM1(char input);
void controlM2(char input);
void controlM3(char input);
void controlM4(char input);
void controlM5(char input);

#endif // _TKIMOTORCONTROL_H_
```

8.2.7 TKI_UART.c

```
#include <system.h>

#include "TKI_UART.h"
#include <stdlib.h>

void init_usart(unsigned short rate)
{
        trisc.6=0;  // transmit (txd) to output
        trisc.7=1;  // receive (rxd) to input
        baudcon1.3=1;   // BRG16
```

```
            spbrg1=rate;
            txsta1 = 0b00100100;
            rcsta1 = 0b10010000;

}

volatile bit trmt@PIR1.4;

void putc(char value)
{
 while(!trmt);
 txreg1=value;
 return;
}

volatile bit da@PIR1.5;

char getc(void)// to do: make functions for each motor
{
        char v;
        while(!da);
        v=rcreg1;
        putc(v);

        while(da);
        return v;
}

void puts(char *strn)
{
    int i=0;
    while(strn[i]!=NULL)
    {
        putc(strn[i]);
        i++;
    }
}

void putd(unsigned short dat)
{
  unsigned short val;   // ascii results
  unsigned short temp;
  unsigned short div;
  unsigned short data;
  char i;
  char digit;
  data = dat;  // make it unsigned
  div = 10000;
  for(i=0; i <= 4; ++i)   // get all 5 digits
  {
    val = data/div;       // get most signif. digit
    putc(val + '0');    // print digit
    data -= val * div;   // what we?ve printed
    div=div/10;         // adjust divisor
  }
  return;
}

unsigned short read_short(void)
{
  int i;
  char c;
  unsigned short a=0;

  for (i=0; i<5; i++)
  {
    c = getc();
    if (c==13)
      break;
    if (c>=48 && c<=57)
    {
      a *= 10;
      a += c-48;
    }
  }
  return a;
}
```

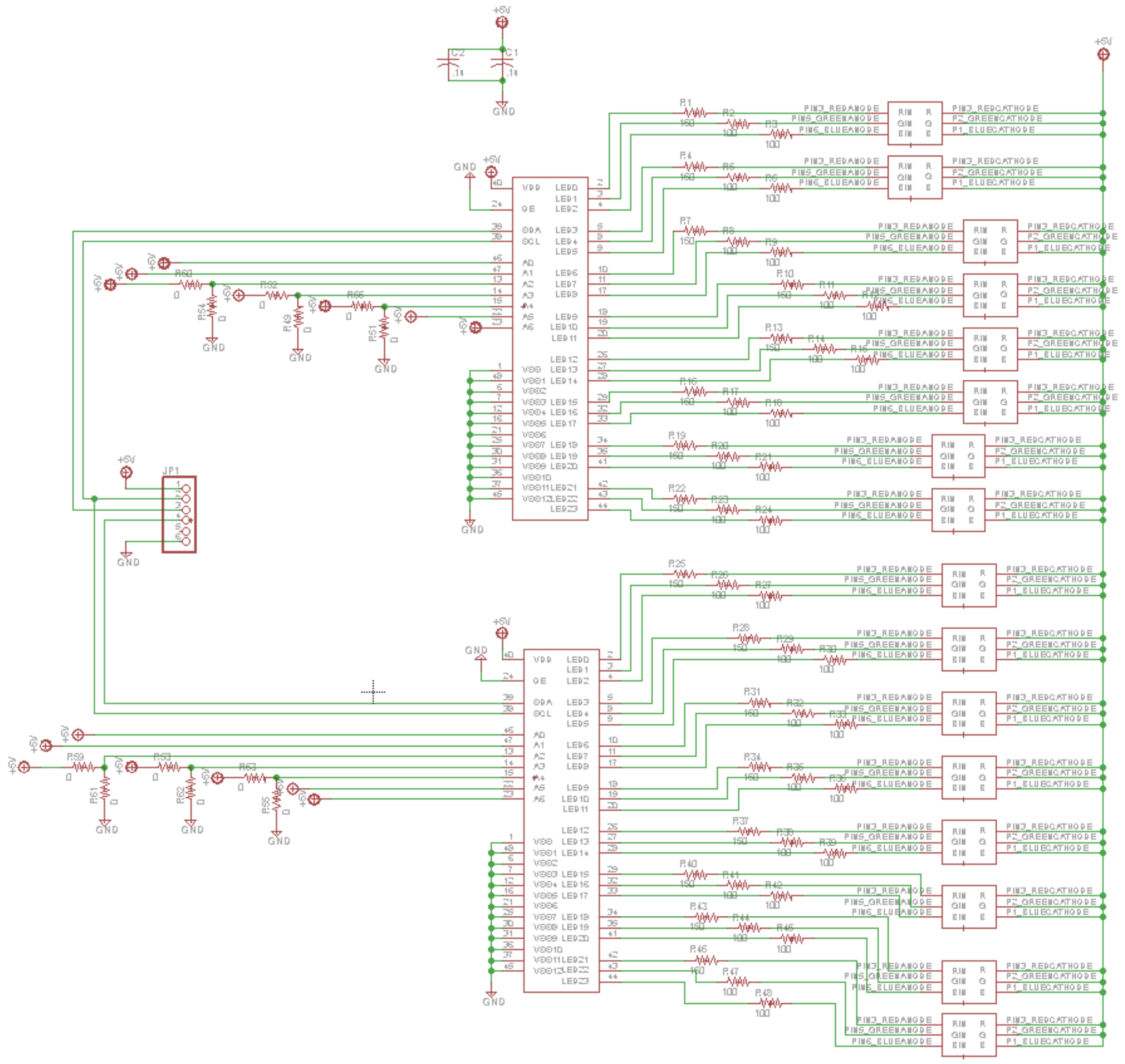
8.2.8 TKI_UART.h

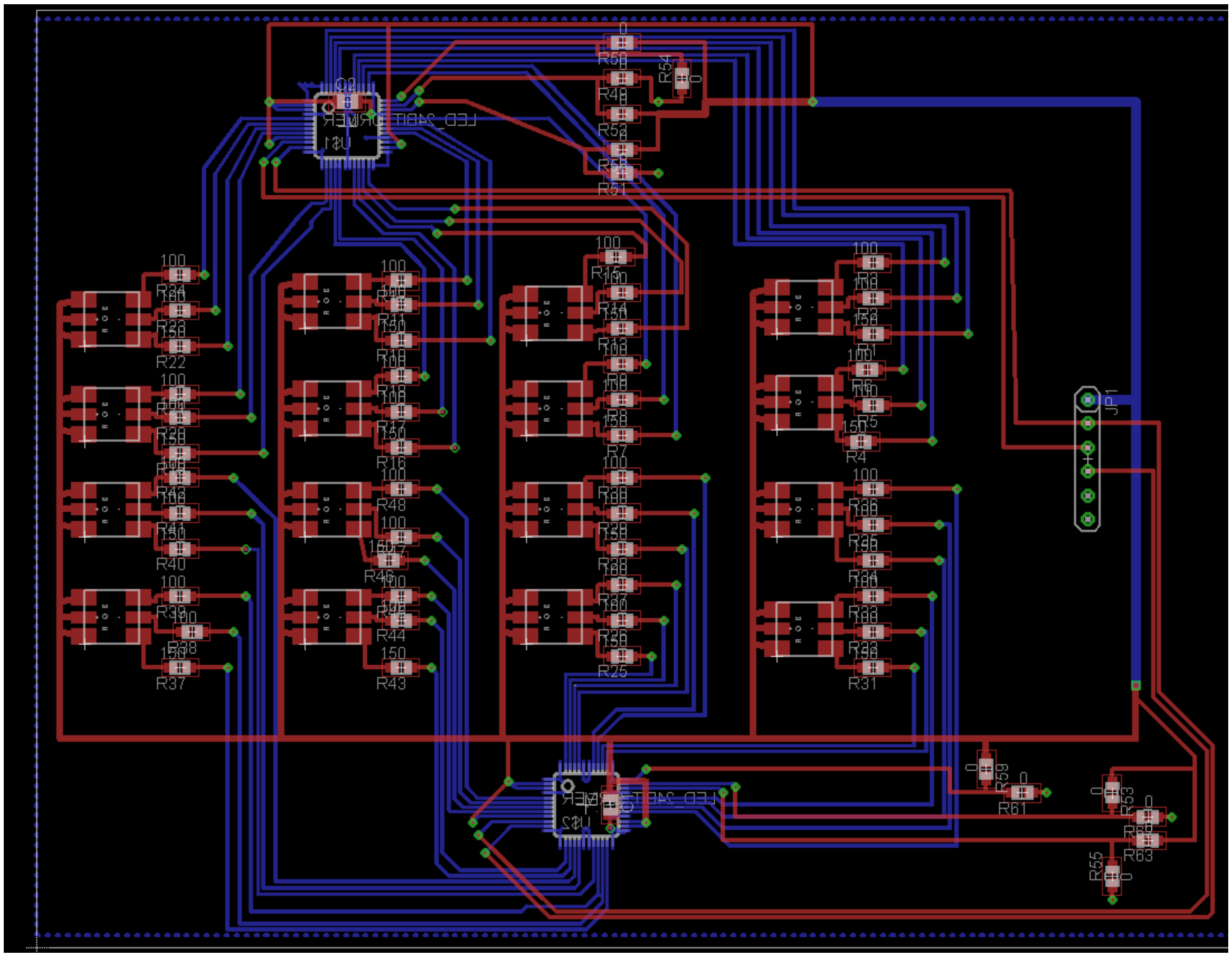
```
#ifndef _TKIUART_H_
```

```
#define _TKIUART_H_

void putc(char value);
char getc(void);
void init_usart(unsigned short rate);
void puts(char *strn);
void putd(unsigned short dat);
unsigned short read_short(void);

#endif // _uart_motordriver_H_
```

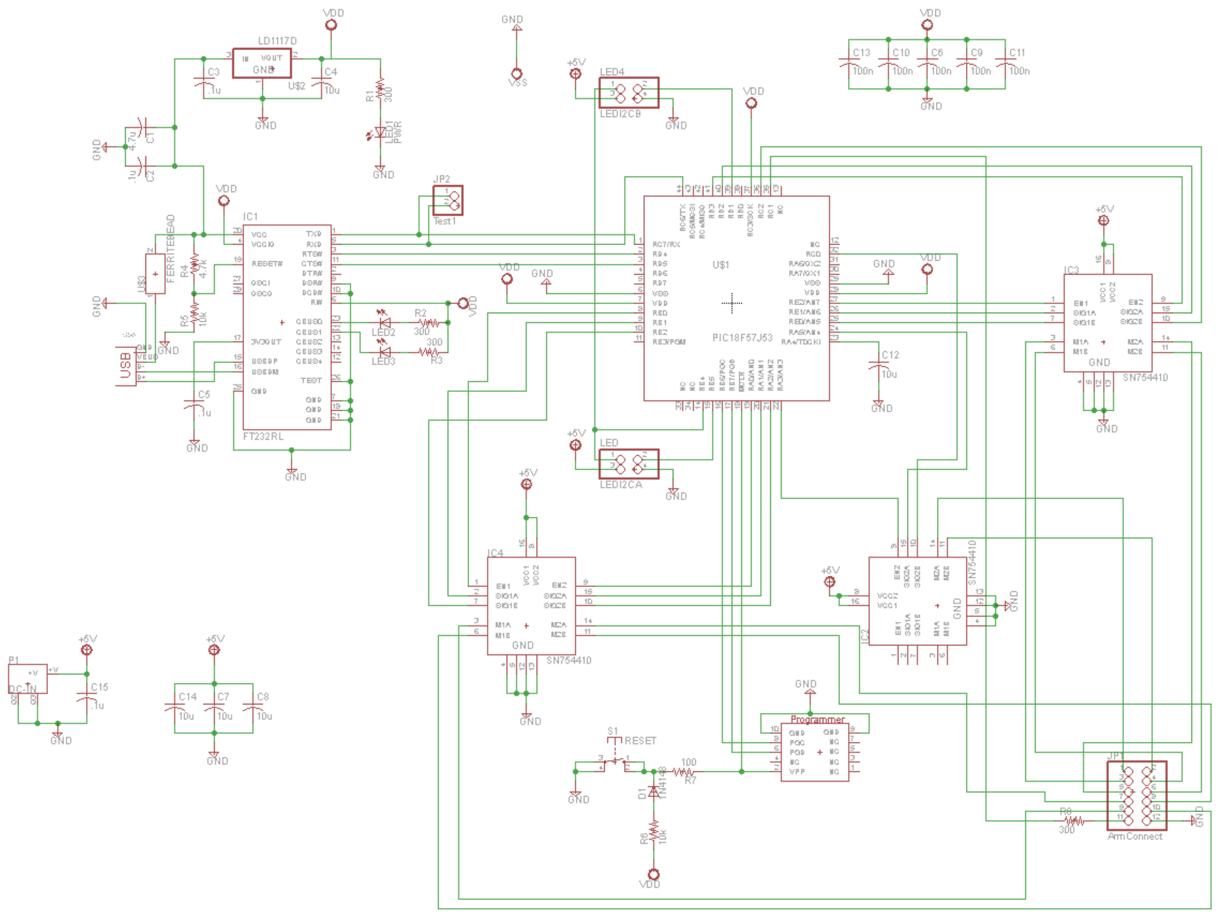
8.3 Hardware Schematics

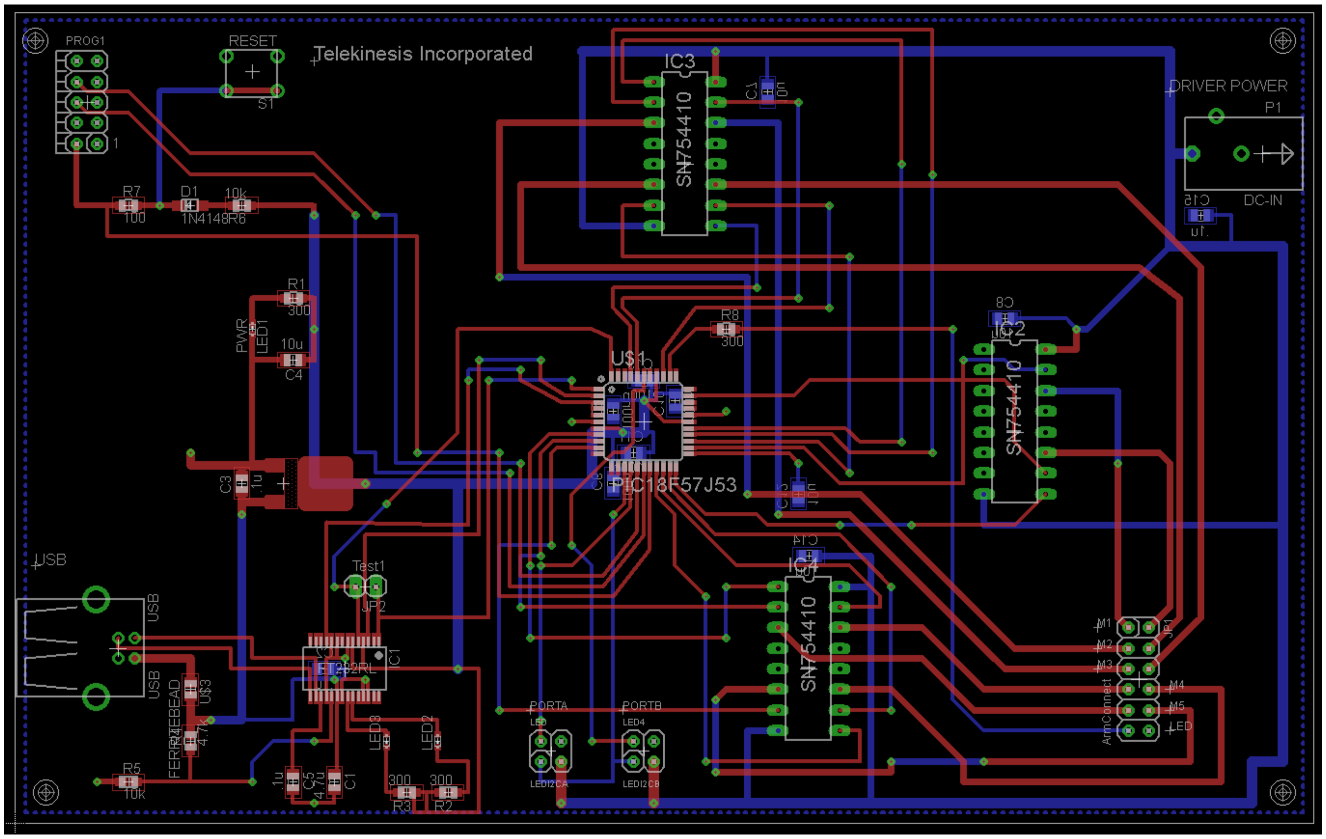
8.3.1 LED Board





8.3.2 Main Board


LD1117D
VDD
GND
VSS
+5V
LED4
LEDI2CB
C13
C10
C6
C9
C11
100n
C3
C4
.1u
10u
U$2
R1
300
LED1
PWR
C1
C2
4.7u
JP2
Test1
IC1
FT232RL
FERRITEBEAD
R4
R5
4.7k
10K
USB
LED2
LED3
R2
R3
U$1
PIC18F67J53
C12
IC3
SN754410
C5
LED
LEDI2CA
IC4
IC2
Programmer
S1
RESET
R7
100
D1
1N4148
R6
P1
DC-IN
C15
C14
C7
C8
JP1
Arm Connect
R8



8.4 Datasheet Links

8.4.1 Motor Drivers

http://www.ti.com/lit/ds/symlink/sn754410.pdf

8.4.2 LED Drivers

http://www.nxp.com/documents/data_sheet/PCA9626.pdf

8.4.3 Microcontroller

http://ww1.microchip.com/downloads/en/DeviceDoc/39964B.pdf

8.4.4 EEG Headset

http://www.emotiv.com/upload/manual/sdk/Research%20Edition%20SDK.pdf

NOTE: Look to this datasheet only for headset specs; we did not purchase the sdk software

8.4.5 RS-232 Chip

http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf