Department of Electrical and Computer Engineering

332:428 Capstone Design - Communications Systems Spring 2012

The Brain Wave App

Final Project Design Report

Group Members:
Mrunal Shah
Richard Roman

Project Director: Dr. David G. Daut

May 9, 2012

Table of Contents

1. Design Project Overview

There is no technology on the market at the moment that allows communication by thought. We wanted to change that. We made an application that would allow people to just think and convert their brain waves into tangible actions. Through this project we wanted to introduce a new medium and a new way to interact with surroundings, just by using brain waves. People from all walks of life would benefit from such a project. For example, disabled people can establish two-way communications with someone living far away. Additionally, they would be able to control appliances in your house just by thinking.

In our approach, we procured Emotiv Epoc headset, an EEG headset that reads potential difference between different parts of the brain. Additionally, we used signal processing to output P300 brain waves. Essentially, we are interpreting what the using is thinking. We take the signals from Emotiv headset and pass it through our custom software to decide what emotions or brain patterns are coming from the user. Using the Twitter API, we send this message over the Internet so that it can be read by anyone with Internet access.

In the second part of our project, we are controlling a television just by thinking. Just like in the previous part, custom software is used to decipher what signals are coming out of the headset. We then translate those signals to interpret what actions the user is trying to achieve. The resulting signal is passed along to our remote tool, which utilizes the Arduino Uno, to control the television.

The project has been successfully completed. We are able to decipher those brain waves and communicate via Twitter, along with control a television via remote signals. Additionally, we have been successful in achieving two-way communications. Not only can we send messages to the Internet, but we can also indicate if messages were received. A reply back to a tweet can be indicated by the flashing of an LED connected to a protoboard. We have effectively, enabled communication and convenience in one cheap, easy to use product.

2. Technical Specifications

Hardware Technical Specifications:

The project used three different pieces of hardware for our project:

i. Emotiv Epoch EEG headset.

ii. Arduino Microcontroller.

iii. TV remote control.

i. Emotiv Epoc EEG Headset:

The Emotiv headset comes along with the following hardware.

i. USB Transceiver Dongle

ii. Hydration Sensor Pack with 16 Sensor Units

iii. Saline solution

iv. 50/60Hz 100-250 VAC Battery Charger

Number of channels	14 (plus CMS/DRL references)
Channel names (Int. 10-20 locations)	AF3, AF4, F3, F4, F7, F8, FC5, FC6, P3 (CMS), P4 (DRL), P7, P8, T7, T8, O1, O2
Sampling method	Sequential sampling, Single ADC
Sampling rate	~128Hz (2048Hz internal)

Resolution	16 bits (14 bits effective) 1 LSB = 0.51μV
Bandwidth	0.2 - 45Hz, digital notch filters at 50Hz and 60Hz
Dynamic range (input referred)	256mVpp
Coupling mode	AC coupled
Connectivity	Proprietary wireless, 2.4GHz band
Battery type	Li-poly
Battery life (typical)	12 hrs.
Impedance measurement	Contact quality using patented system

Table 1: EEG Headset Technical Specifications [0]

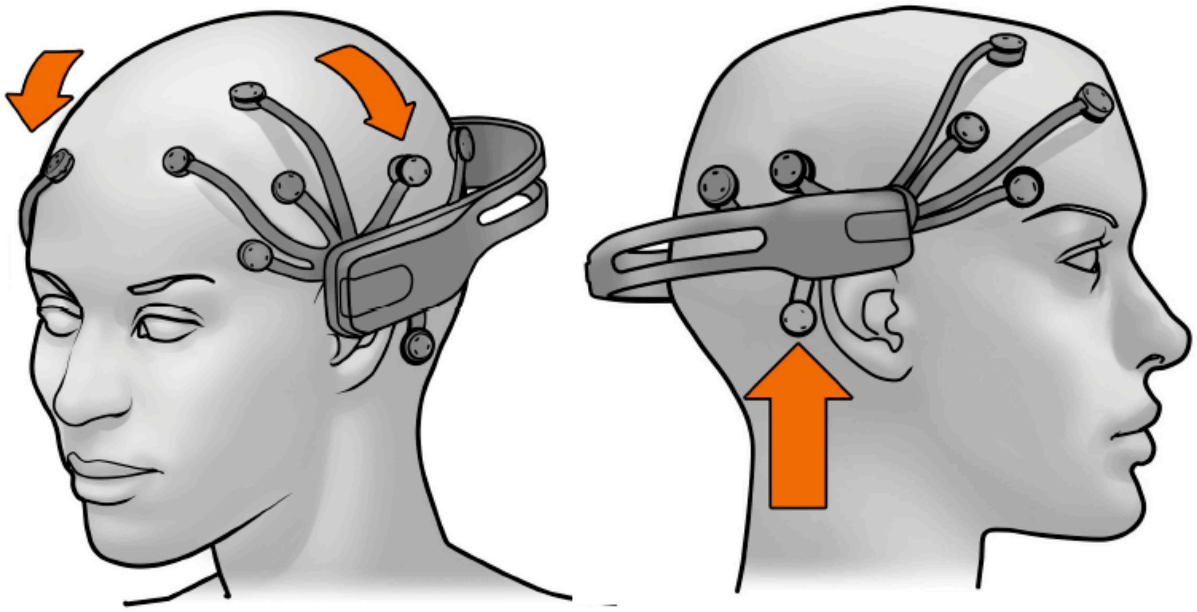


Image 1: Ideal placement of individual sensors on the scalp

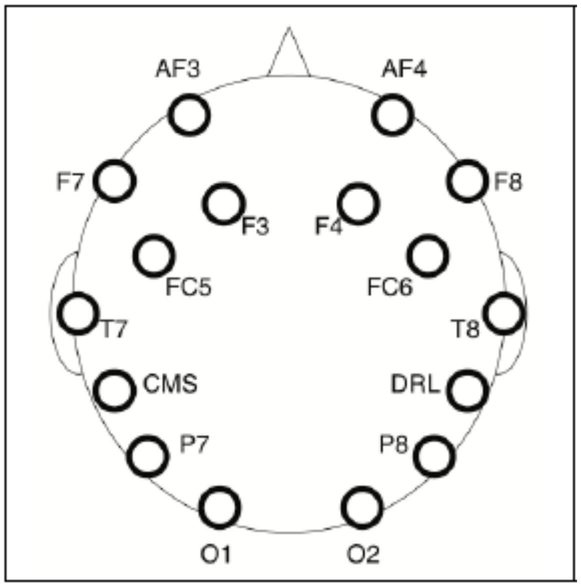


Image 2: Electrode Positions

All the information wirelessly transmitted from the headset is encrypted and is decrypted on computer by edk.dll making it hard to know what information individual sensors are collecting.

Signal Processing done for brain wave detection:

Preprocessing done by Emotiv headset:

i) Low-pass filter with a cutoff at 85Hz.

ii) High-pass filter with a cutoff at 0.16Hz.

iii) Notch filter at 50Hz and 60Hz.

After the pre-processing signal is obtained, it is streamed to the computer wirelessly to be made available to the edk.dll. The Sampling Rate is 128 samples per second and is actualized on the computer using Sampling Algorithm (Appendix 2, #1).

To reduce the anomalies introduced into digital signal as a result of signal processing, the artifact removal algorithm is used. To detect blinks the mean and standard deviation of each near-eye electrode has been computed and then scanned for samples exceeding a standard-deviation-related limit. Those samples are joined into nearby groups and finally fitted (extended to the left and right) until the value has fallen below another standard-deviation-related limit (code in Appendix 2, #2,#6). It is then passed through a 2nd -order IIR-Filter and Averaging is done. Averaging is done under the assumption that noise is randomly distributed.

ii. Arduino Uno Microcontroller:

The Arduino Uno is an open source microcontroller useful for rapid electronic prototyping.

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Table 2: Technical Specifications for the Arduino Uno [2]

iii. TV-Remote:

We used Sanyo TV’s remote to detect what information was being sent using to the television, so that it could be emulated on Arduino. We used an IR detector to detect what information was being pulsed from the remote. Using Oscilloscope we were able to see the pulses on screen. The circuit diagram to detect the pulses can be seen in image 3.

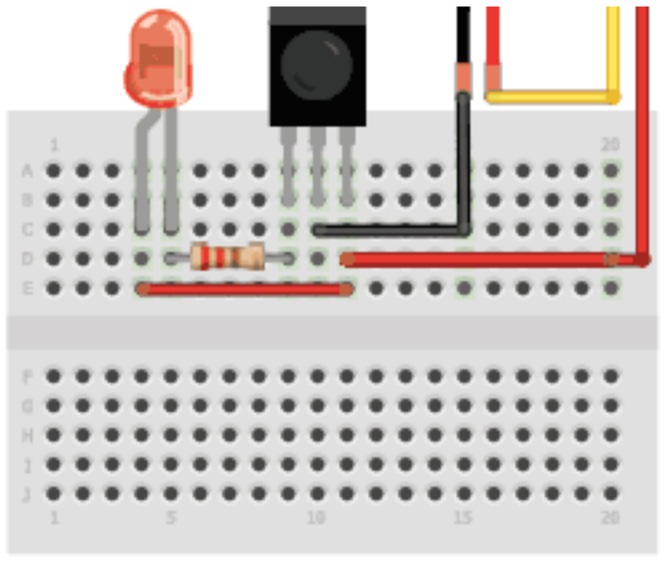


Image 3: Circuit Diagram to detect infrared pulses

The infrared LED detects the output. The black wire is connected to ground and the red wire is connected to 5V power. The waveform obtained is displayed in image 4.

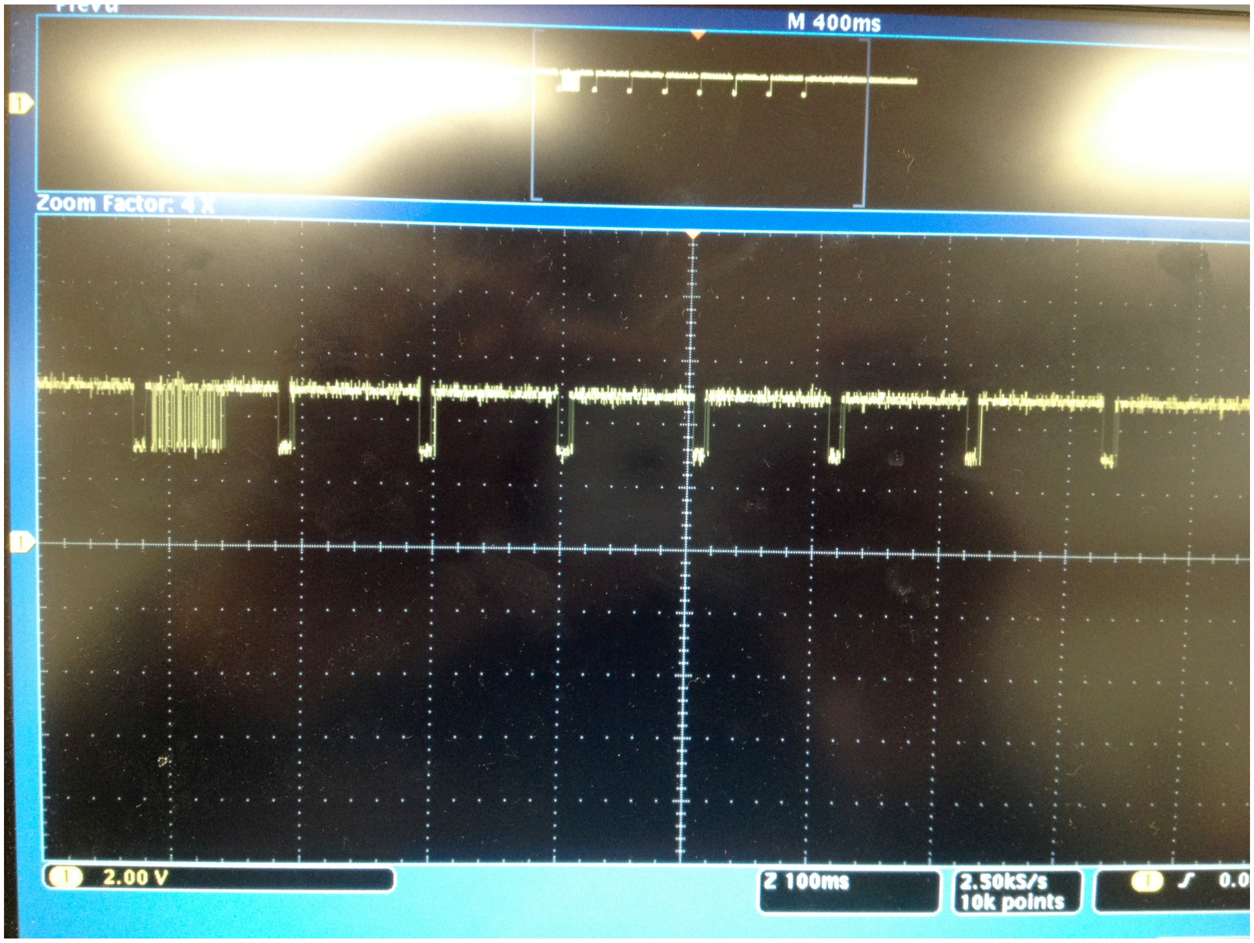


Image 4: Waveform obtained from the infrared LED detector

Using the Cursor we were able to find the time between 2 pulses. High is IR-LED on/off at 38KHz frequency. Low is IR-LED off.

Software Technical Specifications:

Our project utilizes a number of different software components:

i. Emotiv Control Panel

ii. Mind Your OSC's

iii. Processing

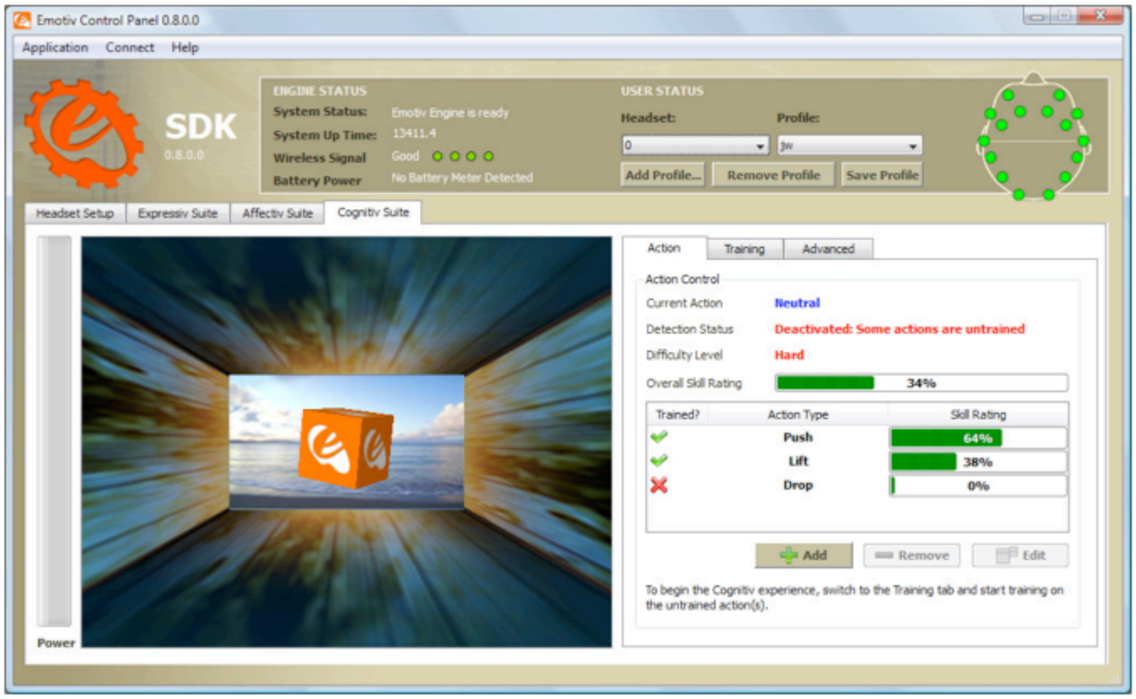
iv. Twitter API

v. Python Script

vi. Arduino IDE

i. Emotiv Control Panel

The Emotiv Control Panel is a proprietary program that presents the user with a logical, graphical interface for using the headset. It allows the user to utilize all of the capabilities of the headset and customize actions according to brain waves. It consists of four main tabs. However, only one tab was used for our purposes. Depicted in image 5 is the Cognitiv Suite. This screen allows the user to designate certain thoughts or facial movements to certain actions. Actions used in our program include push, pull, lift, and disappear.



System Status: Emotiv Engine is ready
System Up Time: 13411.4
Wireless Signal: Good
Battery Power: No Battery Meter Detected
Headset: 0
Profile: jw

Action Control
Current Action: Neutral
Detection Status: Deactivated: Some actions are untrained
Difficulty Level: Hard
Overall Skill Rating: 34%

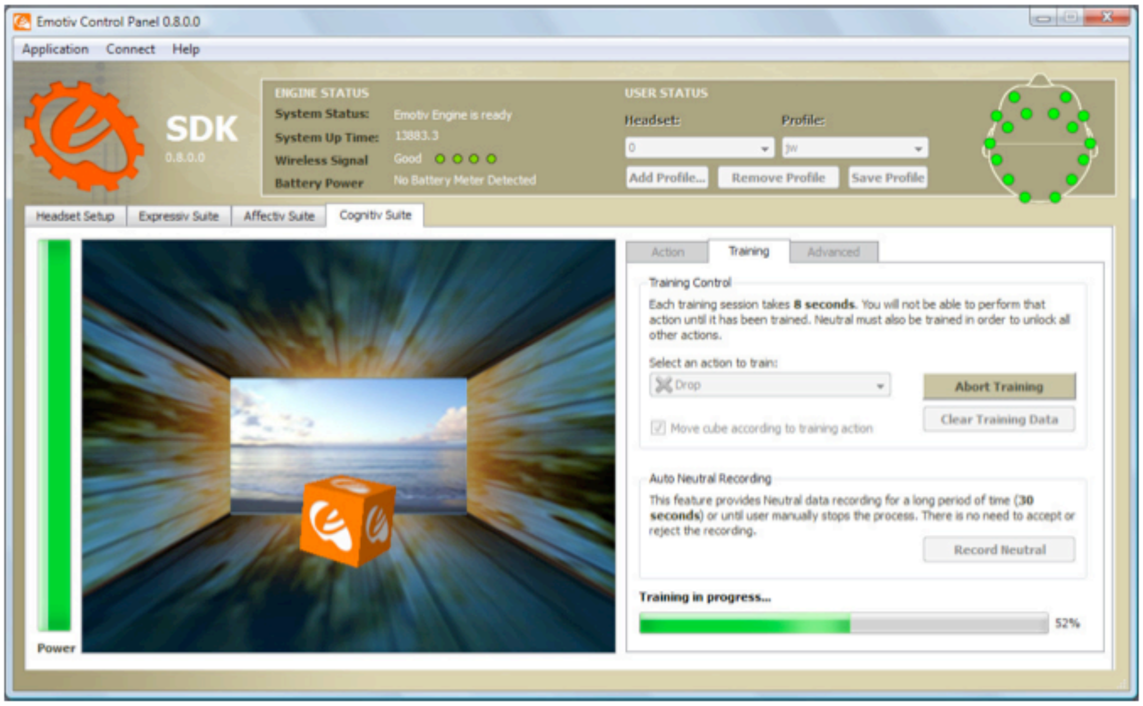
Trained?	Action Type	Skill Rating
✓	Push	64%
✓	Lift	38%
✗	Drop	0%

To begin the Cognitiv experience, switch to the Training tab and start training on the untrained action(s).

Image 5: Emotiv Control Panel: Cognitiv Suite

Also, notice the upper panel. It displays the essential information regarding the status of the headset. Information includes, the wireless signal, user profile, and sensor status (green means working well).

In order to use the chosen actions, each user must train the program to recognize appropriate brain waves. This is where the Training Screen comes into play.



ENGINE STATUS
System Status: Emotiv Engine is ready
System Up Time: 13883.3
Wireless Signal: Good
Battery Power: No Battery Meter Detected

USER STATUS
Headset: 0
Profile: jw

Training Control
Each training session takes **8 seconds**. You will not be able to perform that action until it has been trained. Neutral must also be trained in order to unlock all other actions.

Select an action to train: Drop

☑ Move cube according to training action

Auto Neutral Recording
This feature provides Neutral data recording for a long period of time (**30 seconds**) or until user manually stops the process. There is no need to accept or reject the recording.

Training in progress... 52%

Image 6: Emotiv Control Panel: Cognitive Suite – Training Screen

Trainings must be complete for each action in order for the headset to work. Training consists of the user doing the desired facial action or think of the idea that will be tied to each action continuously, for a short interval of time. After this is completed the program will recognize which action the user is trying to accomplish by reading the users “thoughts”. [3] While the Control Panel does the difficult job of taking in the raw data from the headset and translating it into more easily used data, the data outputted from the Control Panel is still not easily utilized in other programs. To solve this problem, we found another program helps out the process.

ii. Mind Your OSC's

Mind Your OSC's is an open source project that changes the output from the Control Panel into OSC packets.[4]Open Sound Control (OSC) is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices.[5] OSC information is sent in regulated data packets that have a distinct format that can be easily read by other programs. There are libraries in place in our other components that are custom tailored to OSC protocol. They include functions that deal specifically with OSC packets to accomplish convenient and effective processing. This program provides a much easier and more effective way of transmitting our information between components.

iii. Processing

Processing is a programming language, development environment, and online community that since 2001 has promoted software literacy within the visual arts. Initially created to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing quickly developed into a tool for creating finished professional work as well.[6]

Processing provided the group with an easy to use development environment that is very flexible and allows programmers many conveniences. Because it uses a language that is based heavily off of Java, the project team was able to focus on developing our project without the disadvantage of having to learn a new language. Additionally, Processing is able to interface with many different systems, such as Twitter and Arduino. This enabled us to experiment with different methods of design, relatively easily. This facilitated a dynamic design process that allowed change and let us refine our design plan quickly and effectively.

The open source nature Processing also afforded us another advantage. Because it has been developed and improved by many different people in the open source community, there are a variety of different libraries that interface with the systems listed above. These libraries are easy to install and add a great deal of capabilities to any Processing program.

The oscP5 library was written by Andreas Schlegel for the programming environment Processing. The library features Automatic Event Detection, in which oscP5 locates functions inside your sketch and will link incoming OSC message to matching functions automatically. Additionally incoming OSC messages can easily be captured within the sketch. In order to install it, one only has to unzip the downloadable file and put the extracted oscP5 folder into the libraries folder of your processing sketches. [7]

Twitter4J is an unofficial Java library for the Twitter API. With Twitter4J, we easily integrated the Java application with the Twitter service. It features 100% pure Java (works on any Java Platform version 1.4.2 or later), zero dependency (no additional jars are required), built-in OAuth support (the standard authorization method used by Twitte), and out-of-the-box gzip support. In order to install it the user just adds add twitter4j-core-2.2.5.jar to the application classpath.[8]

iv. Twitter API

The Twitter API is a great tool for developers. It allows a programmer to easily access the Twitter system to do a number of operations. Posting tweets, reading the TwitterFeed, sending and reading Direct Messages are all operations that can be accomplished. A programmer just needs to visit dev.twitter.com and go through a few steps to get authorized to access his or her twitter account remotely. After registration, the account menu holds all of the information needed to allow an application to successfully complete OAuth. OAuth is an authorization system that allows the user to grant access to your private resources on one site (which is called the Service Provider), to another site (called Consumer, not to be confused with you, the User). The purpose of this is to provide a standard way for developers to offer their services via an API without forcing their users to expose their passwords (and other credentials).[9]The fact that the credentials are easily obtained after registration is a big benefit and assists our system greatly.

v. Python

Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains. Python has some key features that assisted us in the completion of this project. They include:

- very clear, readable syntax
- strong introspection capabilities
- intuitive object orientation
- natural expression of procedural code
- full modularity, supporting hierarchical packages
- exception-based error handling
- very high level dynamic data types
- extensive standard libraries and third party modules for virtually every task

The language itself is a flexible powerhouse that can handle practically any problem domain. Additionally, python runs on Windows with minimal configuration.[10] Due to the fact that it is open source, there are libraries available that help to interface with many different systems, including in our case, Twitter.

The pySerial libary encapsulates the access for the serial port. It provides backends for Python running on Windows, Linux, BSD (possibly any POSIX compliant system), Jython and IronPython (.NET and Mono). The library contains a module that automatically selects the appropriate backend for the python script. The library features:

- Same class based interface on all supported platforms.

- Access to the port settings through Python properties.
- Support for different byte sizes, stop bits, parity and flow control with RTS/CTS and/or Xon/Xoff.
- Working with or without receive timeout.
- File like API with “read” and “write” (“readline” etc. also supported).
- The files in this package are 100% pure Python.
- The port is set up for binary transmission. No NULL byte stripping, CR-LF translation etc. (making the library universally useful
- Compatible with io library (Python 2.6+)

The package can be easily installed by downloading the archive off of the website, unpacking it (command: pyserial-x.y), and running it (command: python setup.py install) .[11]

The simplejson library is a simple, fast, complete, correct and extensible JSON encoder and decoder for Python 2.5+. It is pure Python code with no dependencies, but includes an optional C extension for a serious speed boost.[12] **JSON** (JavaScript Object Notation) is a lightweight data-interchange format. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.[13] The installation method used by the pySerial library is standard for all Python libraries. Thus it can be used to install the simplejson library as well.

The httplib2 library is a comprehensive HTTP client library that supports many features left out of other HTTP libraries. Features include:

- Support for both http and https
- Support for keep-alive - keeping the socket open and performing multiple requests over the same connection if possible.
- Supports the following authentication types:
    - Digest
    - Basic
    - WSSE
    - HMAC Digest
    - Google Account Authentication
- Caching
- Can handle every http method, not just get and post
- Redirects - Automatically follows 3XX redirects on GETs.
- Compression - Handles both 'deflate' and 'gzip' types of compression.
- Lost Update Support - Automatically adds back ETags into PUT requests to resources we have already cached.
- Unit Tested - A large and growing set of unit tests.

Can be installed in the same way that the other python libraries were installed.[14]

The python-oauth2 class handles authorization. This is not specific to any application it handles general oauth.

The python-twitter library provides a pure Python interface for the Twitter API. The library provides a Python wrapper around the Twitter API and the Twitter data model.

vi. Arduino

Arduino provides not just hardware for our project, but also software. Along with the Uno microprocessor, the Arduino development environment was also used. It is an environment that interfaces specifically with the Arduino microprocessor. Luckily, there are already extensive standard libraries that are in place, so no new libraries were needed to finish the program that was loaded onto the Arduino.

3. Final Project Summary

3.1. System Design – Final Version

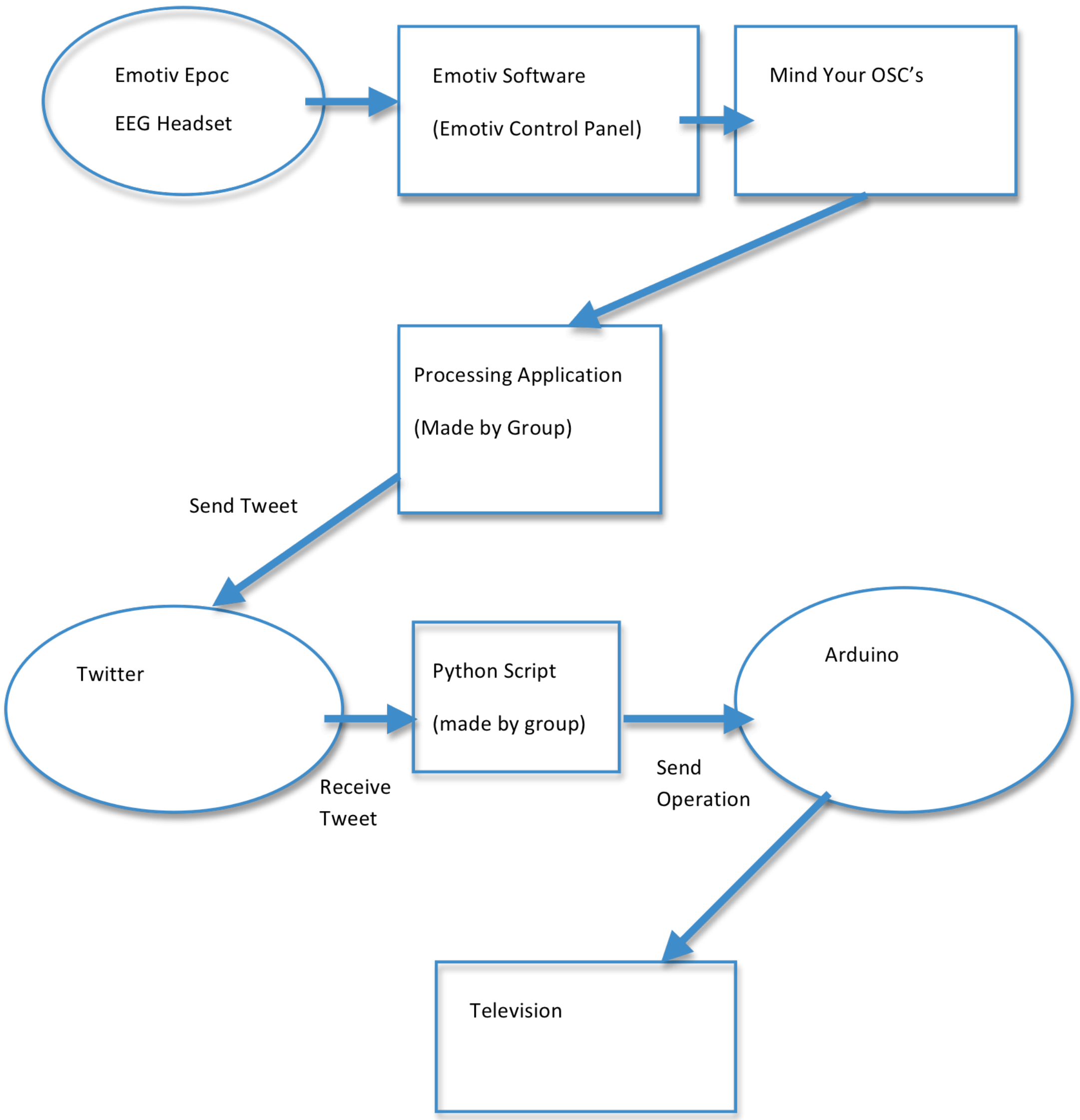


Image 7: Final Version Block Diagram

Our project is designed to integrate new technology with custom programs. The design features only a few products that will cost the consumer money. The rest of the components are open source. Even the custom programs were written on open source programs. This is not only cost effective, but also encourages consumers to customize it to their needs.

It all starts with the Emotiv Epoc Headset. It reads the user's brain waves and, over wi-fi, outputs data into the Emotiv SDK (Software Development Kit). The Emotiv SDK translates the raw data into information that is recognized by the Emotiv Software Suite. Unfortunately this information is only understood by Emotiv propriety programs. To remedy this we utilized Mind Your OSC's. This open source program translates the information into OSC data packets (explanation will be provided in future sections). A custom program written in Processing recognizes what operation the user is trying to do and outputs one of four pre-loaded tweets onto the user's Twitter account. From here, anyone with an Internet access can see the outputted tweets (Twitter allows account protection so that only accepted users can view tweets, but this is out of the scope of our project). This allows long-range communication. From here, a custom python script, which runs in the background, reads the last tweet in which the user was mentioned. It looks for flags in the tweet (hashtag keyword) that either correspond to remote operations, or cause an indication LED to be turned on. It sends a signal to the Arduino Uno. From here the Arduino decides whether to manipulate an infrared LED, or to light an indicator LED. In the latter, the LED is turned on, letting the user know that their tweet has been answered. If a remote operation has been recognized, the Arduino pulses the infrared LED to a certain frequency that then completes the operation on the television.

After all is said and done, our system allows the user incredible convenience and communicative capabilities. We have also established a new medium through which communication can take place.

3.2. System Performance

Theoretically the system should be very fast. Due to light nature of the processing needed, the computer processing should only take milliseconds total (one second maximum). The other components communicate over an Internet connection. Seeing as how fast the Internet has become, this should not be a problem. Communication should take a few seconds at the most. The other communication is done over wi-fi (headset to computer) and infrared frequencies (infrared LED to TV), both of these mediums are very quick and reliable and should not even have one second of latency. All in all, the entire system should just take seconds to complete.

Unfortunately this is not an ideal world, and our system does have some dependencies that cause latency. Firstly, it is essential that there is reliable, and fast wi-fi for the headset to send consistent and accurate data to the computer. This was never a problem during development and testing, so we assume that this is not a strong vulnerability for the system. The computer processing is dependent on the stress of the computer. Because of the low load of processing data in our program, the cpu is generally not overworked. Just like the wi-fi this is a weak vulnerability of the system.

The first significant latency issue we run into lies in the Twitter API. We made the design choice to use Twitter as the communication medium for the remote operations. This was due to the fact that it was already being used for communication purposes, and it was an easy addition to utilize the Twitter API to send operations to our remote tool. While this is a reliable way of sending operation instructions, it is not the fastest. Naturally, the Twitter API must be accessed over an Internet connection.   Unfortunately this exposes the system to inconsistent dependencies. A reliable, and fast internet connection is needed to not only send tweets out, but also to read the tweets back into the system. Additionally, we are dependent on the speed of Twitter's API. If the site is experiencing large amounts of traffic, that could slow down the posting of tweets, and therefore slow our system down. However, this is not the biggest problem. It is impossible to be able to tell how slow it would be in the worst case. However, in a bad case it would still only take up to thirty seconds to post a tweet.

The biggest latency comes from the Python script that recognizes incoming tweets. The script was designed to poll Twitter every twenty seconds. Because of this interval, the speed of recognition depends upon the time the tweet was received in relation to the latest polling. There is an average wait time of ten seconds. However it is possible that recognition could take anywhere from one second to twenty seconds. This easily causes the biggest latency, and stops our system from being as responsive as we hoped.

3.3. System Design Iterations

Iteration 1

Our initial proposal idea was to make a Brain wave app that could control an iPhone just by thinking. We were going to achieve this by making an iPhone (iOS) Application that would understand information that was being streamed to it from the headset.

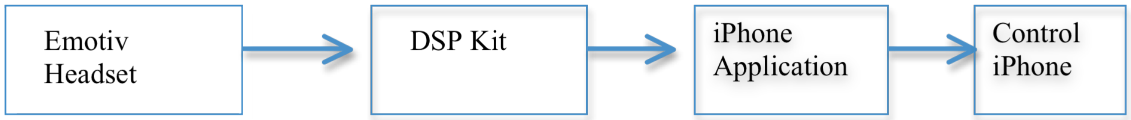


Image 8: Iteration 1 Block Diagram

However we hit many hurdles while working on this particular way of controlling our iPhone.

i. The iPhone is locked by Apple, which means that Apple determines how applications can interact with the iPhone.

ii. The only way to break the barriers of communication with iPhone is the to jailbreak it. Jail breaking is a process through which user gets the administrative privileges or becomes the “root” of the phone. As a result, the user can install any application he or she wants. In this setup certain ports and files are accessible for further development. Jail breaking is legal under laws of US. However, every country has its own sets of law and rules governing jail breaking. This limits the extent to which the application can be distributed. Jail breaking is also not recommended by Apple.

iii. We jail broke the iPhone that we were using. However, even after this and getting root access to all the files of our phone, we were not able to effectively stream the to the smartphone using current existing technology. We could send information as files, but could not stream continuous information to our smartphone.

Iteration 2:

In the next stage of our project, we changed our approach. We decided to make an application that could send emergency messages just by thinking. This would enable the user (presumably disabled) to turn on or off LED's depending on what he or she was thinking.

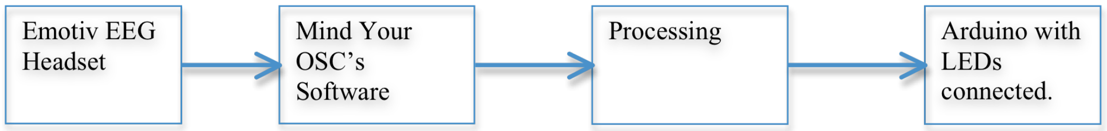


Image 8: Iteration 2 Block Diagram

We successfully achieved this part of our project and were able to send information to an Arduino from the Emotiv EEG Headset.

Iteration 3:

After achieving local communication, we decided to extend this even further by attempting more universal communication. We chose to attempt communication over the Internet. After researching various Internet systems, we decided on using Twitter. Our messages could be sent as tweets, and would reach a large amount of people.

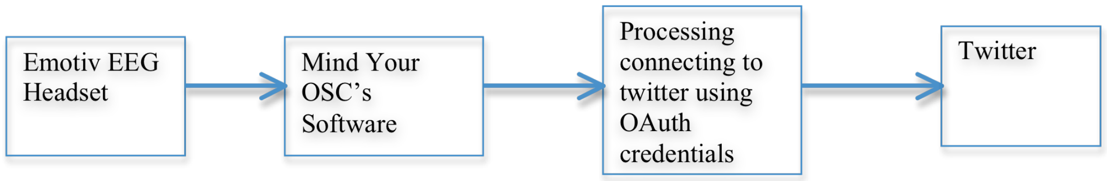


Image 9: Iteration 3 Block Diagram

Using the flow given in the diagram above we were able to connect to Twitter and send information as predefined tweets. This enabled the user at home to communicate with anyone that had a Twitter application installed on his or her phone. Our communications could now range over remarkably long distances.

We found that there were two ways of connecting to Twitter. We could connect over the computer's Internet connection using Processing, or through the Ethernet Shield add on to the Arduino Uno. In order to use the Ethernet Shield, the Arduino would have to be hard-wired into either an Ethernet port, or a router. This added a new dependency and a new inconvenience. Because of this inconvenient fact, we settled upon using Processing. Since we were already using the computer to process the headset signal, it was convenient to use it to submit the tweets.

Iteration 4:

We were able to successfully establish one-way communication using the Emotiv Headset. However, we wanted to establish two-way communication. We wanted to indicate to the user if a response had been received. The response from the initial recipient would be signaled by an LED turning on or off, depending on the information in the response tweet. To achieve this we wrote a custom python script that would connect to our private twitter account, parse information and look for certain keywords. If those keywords were found the LED's on the Arduino would turn on or off.

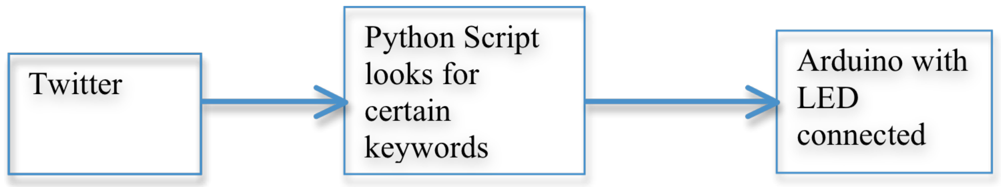

Twitter
Python Script looks for certain keywords
Arduino with LED connected

Image 10: Iteration 4 Block Diagrams

Iteration 5:

We pushed our project further by attemping to control a television just by thinking. We integrated an infrared LED into our system in order to send pulses of information to our TV based on certain keywords found in tweets addressed to our Twitter account. The user would send a tweet including the account username. The python script would parse the information and send a signal to the Arduino. The Arduino had a custom program uploaded that, upon receiving a certain signal, would pulse the infrared LED. This would send information to the television at a rate of 38kHz.

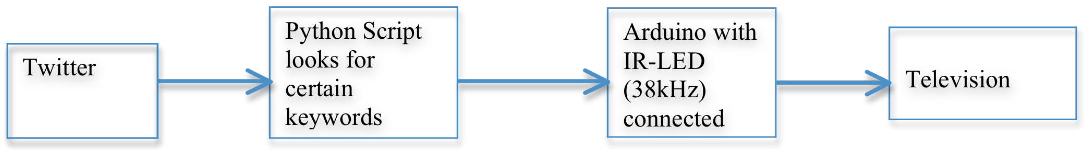

Twitter
Python Script looks for certain keywords
Arduino with IR-LED (38kHz) connected
Television

Image 11: Iteration 5 Block Diagram

4. Task List and Work Distribution

- Idea Conception – Mrunal Shah
- Background Research – Mrunal Shah
- Compatibility Research – Rich Roman – 50%, Mrunal Shah – 50%
- Component Research – Rich Roman 50%, Mrunal Shah 50%
- Material Procurement – Rich Roman – 50%, Mrunal Shah – 50%
- Prototype Construction – Rich Roman – 50%, Mrunal Shah- 50%
- Tweeting Module – Rich Roman
- Tweet Recognition Module – Rich Roman – 30%, Mrunal Shah – 70%
- Remote Operation Module – Mrunal Shah
- Proposal – Rich Roman – 50%, Mrunal Shah – 50%
- Design Review 1 – Rich Roman
- Design Review 2 – Rich Roman
- Final Project Report – Rich Roman – 70%, Mrunal Shah 30%
- Engineering Open House Demonstration – Mrunal Shah – 60%, Rich Roman - 40%
- Poster Day Demonstration – Rich Roman – 50%, Mrunal Shah – 50%
- Presentation – Rich Roman – 40%, Mrunal Shah 60%
- Weekly Progress Reports – Mrunal Shah 60%, Rich Roman – 40%

5. Design Project Details

5.1. Sub-System 1 – Tweeting Module

Designer of Sub-Section – Rich Roman

Author of Section – Rich Roman

5.1.1. Theoretical Considerations

The Tweeting module is the connection between Mind Your OSC's and Twitter. It connects to Twitter using OAuth and posts the pre-loaded tweets that are set in the module.

5.1.2. Design Procedure

Initially this was not a "Tweeting" but was instead a messaging module. It would send messages to the Arduino that would light LED's that were attached to the microprocessor. The first step to creating this module was to send messages using the Emotiv Epoc headset. We started attempting this by using the Processing development environment and Mind Your OSC's. Processing has a library that allows a program to control an Arduino. As a result there were no compatibility issues. We made a program that allowed us to turn on and off an LED that was connected to the Arduino board. The program used thresholds to determine when the incoming OSC values were intended to send a message. This proved to the group that we would be able to send messages using thoughts and facial expressions. It also succeeded in our initial goal of sending messages to an Arduino microprocessor.

We soon realized we could push the messaging concept further. We looked to the Internet for solutions and settled on Twitter because of its technological ubiquity. The first step was learning how to authorize our application to log into and tweet from the project Twitter account. Luckily, Twitter makes it very easy to link applications with Twitter accounts. After registering our application, we were given OAuth credentials that were used in our code.

We started by using the Ethernet Shield on the Arduino Uno to tweet. Because there is a Twitter library that is available for download for the Arduino, this was not a terribly difficult task. Using the credentials given by the Twitter API, we were able to log in and post tweets to the project Twitter account. However we realized that in order for the Ethernet Shield to work, it would have to be hard-wired into a router or modem. This presented the group with a new inconvenience.

We chose instead to use the Processing development environment and forego using the Arduino. Instead, Processing allowed us to use the computer's Internet connection to log into Twitter and post Tweets. A different but equally effective library was available for Processing that assisted us in the process. Soon, we had a program that quickly and effectively posted tweets to our Twitter account using the computer's Internet.

We were able to merge the codes with little hassle. We also extended the code to handle more than one tweet and more than one OSC command. This is when we extended it from 1 pre-loaded tweet to 4. This became our stable version of the code, and is the version that is in the working prototype.

5.1.3. Observed and Measured Results

Because of the nature of our project there were no measurements that needed to be taken; so all of our results were observed. The communication between the module and Twitter is directly dependent on the Internet connection. Luckily a regular connection showed almost instant results between the program sending a signal and a tweet being received and posted.

The communication between the headset and our program was somewhat inconsistent. This was through no fault of the computer or of our module. The problem lied in the headset's recognition. The sensors were quite delicate and sensitive. As a result if they were in the wrong place or were jostled slightly, connection would not be optimal. Additionally, correct outputs required the headset to be able to accurately recognize the brain waves associated with the thoughts or facial expressions you were making. Consistent brain wave training solved this problem. Even this was only a small issue and did not pose a problem on the results as long as the user took care to properly adjust and train the headset. It can be concluded that the results of this module were close to if not faster than the expected values.

Code is located in Appendix 2, #5

5.2. Sub-System 2 – Tweet Recognition Module

Designer of Sub-Section – Mrunal Shah

Author of Sub-Section – Rich Roman

5.2.1. Theoretical Considerations

The Tweet Recognition Module connects the Twitter API with the Arduino program. The script is written in python. It reads tweets directed at the user's Twitter account and sends signals to the Arduino program. The types of signals it sends are dependent on predefined flags (hashtag keywords) present in the tweets. It is the module that deciphers what action the user means to accomplish, and sends the necessary signal to the Arduino to accomplish it.

5.2.2. Design Procedure

This module is a later adaptation of one of the initial codes that was used in the making of the Tweeting module. That initial code, written in Arduino, took a message and performed the on/off operation for an LED connected to the Arduino microprocessor. In essence this module does just that. It takes in messages and outputs signals that are dependent on the message. The main difference is that this module takes in tweets in which the user's profile is mentioned and deciphers commands through keywords present in the tweet. The pre-defined keywords are formatted as #keyword.

After some research it was an easy decision to change development environments for the module from Arduino to Python. The python code allows us to continuously poll the Twitter API. Also like the design process of the Tweeting module, we adopted connection over the computer's Internet connection instead of through a hard-wired Ethernet Shield for convenience purposes. Unfortunately, we could not sidestep this problem with the Ethernet Shield because the Arduino Uno, by itself, cannot connect to the Internet without the help of a computer. Luckily the connection between the python script and the Arduino is seamless, so there are no downsides to this switch.

The interfaces between the python script and the Twitter API are also very easy to integrate. The same credentials that were used in the Processing program in the Tweeting module can be used in the python script. Also, similar to the Processing program, there are libraries in place that interface the Twitter API with any python script very effectively. Even though we are using yet another development program, this module creates a seamless transition between Twitter and Arduino.

Code is located in Appendix 2, #4

5.2.3. Observed and Measured Results

While this module does not present any compatibility issues, it is the main source of latency in our project. Because we chose to use a constant polling method for tweet recognition, there is a constant interval present in which a tweet is not recognized. The module polls Twitter every twenty seconds. This means that the responsiveness of the system depends directly on what time the tweet is submitted. A tweet can take twenty seconds to be recognized or it could take one second to be recognized. After testing our system we determined that it took an average of ten seconds from the time a tweet is posted until the python script recognizes it. We determined this by submitting tweets and measuring the response time in a large sample, and averaging out the data. While the noticeable lag is not desirable, given the accuracy of the system of the module, it is acceptable.

5.3. Sub-System 3 – Remote Operation Module

Designer of Sub-Section – Mrunal Shah

Author of Sub-Section – Mrunal Shah

5.3.1. Theoretical Considerations

Arduino Remote Control: There is two ways of emulating a remote control on Arduino.

The first one is to find information being transmitted on oscilloscope.

And the second one is to find information being transmitted using Arduino.

5.3.2. Design Procedure

We will go in detail on how both the methods work.

1) Finding information sent using Oscilloscope. As we can see from Image 4, there are pulses of information in the beginning that is distinct for every remote and every command.

   The information attached below is when we sent On/Off signal to our TV and captured it on oscilloscope. Using cursor and zooming in, we were able to find distance in microseconds between pulses and record it. We used that information and embedded it in our code to send information to our TV.

2) We can also connect IR-Detector to breadboard and connect it to Arduino to precisely detect the distance between two pulses in microseconds and code that in our Arduino software.

   The final Arduino code used to send On/Off signal can be seen in Appendix 2, #3.

5.3.3. Observed and Measured Results

We had to play with the timing a bit to get just the right numbers so the TV would turn on and off, as there is always an error of +-10% when the information was being sent from the remote.

But after trial and error we were able to get just the right combination for which our TV worked.

6. Sub-System Integration Considerations

Integration was a constant issue in our project. Due to the amount of different components we have in our system, compatibility needed to be kept as a central focus. Before a decision was made on the structure of any custom program, we first needed to decide what the best application to use to produce the program. This was a process of research, but also of trial and error.

The first compatibility issue we ran into affected our overall project idea. The Emotiv headset and the Apple iPhone were both proprietary systems that were not friendly to tampering. They were also highly incompatible. We tried to solve the problem by using go between programs such as Mind Your OSC's. This program solved the issue with the Emotiv headset, but the issue remained with the Apple iPhone. We quickly realized that an iPhone App would not be feasible. Instead we focused on accessing the iPhone through another application. We decided to use Twitter. Because Twitter offers a free application to any iPhone, we still were able to use the iPhone as a tool of communication. The compatibility issues that were present previously were no longer obstacles. Twitter allows users to easily register an application with a Twitter Account, and provides the OAuth credentials to let remote devices or programs access the Twitter API. Additionally, the Twitter API can be easily interfaced with our custom programs in Processing using an open source, downloadable library (Twitter4j). The program that initially solved the issue with the Emotiv Epoc headset previously, also interfaced well with the Processing development environment because of a similar library (oscP5). This overall integration solved the initial goal of our revised project plan, and allowed users to communicate with smartphones. Another tweeting method was attempted using the Arduino Uno Ethernet Shield. Even though it was able to connect to the Twitter API and post tweets, it required a hard-line into a router or modem. This added an inconvenience that was easily circumvented using the Processing approach.

After completing initial goal, we set to work on the recognition aspect of our project. We wanted to let our users not only send tweets to other people, but also have some indication when response tweets were received. We quickly learned that we would need to adopt another new system to be able to have the project come full circle. The Ethernet Shield presented the same problems on the Twitter receiving end as it did on the Twitter sending end so it was immediately discarded. Because the Arduino Uno was not able to connect to the Internet itself we needed to find a program to bridge the Twitter API with the Arduino. We settled upon using python. Both Twitter and Arduino easily interfaced with python. Just like in the Processing code, Twitter had a downloadable library that interfaced with Python. On the other end communication with the Arduino was accomplished using the computer's serial ports. Both Arduino and Python had libraries that enabled serial communication.

Having achieved two-way communication, we tried to find other ways to use the incoming tweets. We found that using signals outputted by our Python script we would be able to complete actions on remote appliances. We had to fashion a remote tool. We knew that remotes use an infrared LED to communicate with their respective appliances. The issue was instead how to use the infrared LED to submit accurate commands. The solution was a circuit that utilized an infrared LED detector (Image 3). Using an oscilloscope and a television remote control, we were able to record the infrared LED signatures that corresponded to the remote operation for on/off. After some measurements we were able to output the specific information that needed to be sent through the infrared LED. The only remaining step was to make an Arduino program that accepted the signals that came in on the shared serial port and manipulate the infrared LED at the correct frequencies to send the information to the television to turn it on. Our project only features one operation. However, this same procedure can be followed to enable other operations to occur on any remote appliance. Additionally the python script can be easily extended to send a multitude of signals. In the end, all of the components used are needed to effectively integrate and synchronize our entire system.

7. Economic Considerations

7.1. Cost Analysis – Prototype

In order to find out the cost analysis of the prototype, one simply has to look at the work done by the group this semester. The group completed a working prototype. The statistics from this semester are displayed below.

Startup costs:

- A computer
    - Windows or Mac OSX operating system
    - 2 usb ports
- An appliance with a remote
    - Note: for our case a television

The group had the necessary startup materials.

Project Materials:

- Emotiv Epoc Headset and Emotiv SDK - $500 in combined package
- Arduino Uno Microprocessor - $40
- Infrared LED - $2
- Resistor - $1
- Protoboard - $40

The group was able to borrow the Emotiv headset and SDK, the Arduino Uno, a Protoboard, and a resistor from Winlab, and the ECE Department

The group paid out of pocket for the Infrared LED.

Manhours:

- 2 people working
- Roughly 136 hours total
    - Note: This includes background research, and code development. The number would be much smaller once the working code is already discovered and used (which is currently the case)

Variations:

- Custom programming would need to be done for every Twitter account, remote operation
    - This could be done by the consumer, but for our purposes we are assuming that this will be done before it reaches the consumer.
- Additional Manhours to program Twitter Account: 1-2 hours
- Additional Manhours to program new compatibility for operation on each distinct appliance: 4-5 hours

7.2. Cost Analysis – Final Version

In order to accurately determine the cost analysis of the finished product, we must take a different perspective than was used when considering the prototype. All of the costs that were incurred while building the prototype, no longer apply.

Because our system is made of many different licensed consumer hardware, and licensed open source software, our system would not be marketed with all of the components presented in the project. Most likely it would be marketed as an add-on to those that already possess those consumer products (this will be explored in further detail in following sections). For this reason, the most expensive materials that were procured in the prototype no longer have to be taken into account (Emotiv products, Arduino Uno). Additionally because we would not be licensed to send out Arduino Uno's, the costs of other components (infrared LED's, resistors) would fall on the consumer. The hardware will have to be built by the consumer. We will provide a manual guiding the consumer through installation, but there are no packaging concerns on our end.

The next expense to consider is manpower. Building our project as a large-scale product holds a distinct advantage in this category over the prototype. Most of the man-hours spent making the prototype were dedicated to researching the capabilities of our project, and developing the code that is in place. Because neither of those things would have to be done, the large numbers of man-hours can be significantly cut down. Because our product would have to be installed on the users computer, very little work is needed on our part. A package would have to be developed to automate installation, but that would be available for download on the Internet, and would not require manual labor. Also, because there is a distinct way (documented in section 2) to find out the infrared signatures of operations on different products, a module could be deployed that would allow consumers to do it themselves, further cutting down on labor needed on our end. Creating this package would result in an initial spike in man-hours, but it would settle back down.

As with most open source projects, support can be done in a forum, and would not need to be immediate (ie. phone line). This cuts down on man hours needed for support. We would still need somebody to answer these forums but it would be a significantly reduced number than if we had a dedicated customer support line.

In the end there are no real costs for our system. Because we are not providing any physical hardware, and because our product will be available online for download, there are no material costs. Because most of the development is done (not including application packaging), and support can be done in a forum, there does not need to be any workers hired (Mrunal and Rich will enough). Because there were no startup costs in building the prototype, and no workers will be hired there will be no startup costs. So taking this large-scale in reality will be quite a cheap endeavor.

8. Manufacturability

As can be seen already, our system requires consumer products that are licensed and already on the market. Additionally, the system uses open source software that is also licensed. In reality, if this were a small time operation, this probably would not matter. However in the interest of this report, we have to assume that this would become a legitimate business. If this became a legitimate business venture, we would have to gain permission from the companies that own these products to use them and sell them in our package. This would be difficult to do. Realistically, even if we gained permission, we would have to order these products in bulk in order to get a reduced price. This is the only way that manufacturing would be profitable. Buying $500 headset packages in bulk, even with a markdown to $300 would be very difficult to handle considering we would be starting our own business. That is not even taking into account the money spent on purchasing Arduino Uno's in bulk. This would most likely not be a successful business venture.

Assuming that the previous option is not feasible, we could take the time to develop our own propriety components. We would have to make our own headset, our own microprocessor, and make new software that doesn't run on the open source programs we are currently using. This would take an incredible amount of time and money. It is simply not realistic to assume that it would be possible for our project to be extended to include a custom EEG headset and microprocessor. Assuming we did create prototypes of both, it would be unrealistic to think that we would be able to secure the funding needed to create facilities that could make these things on the scale needed to start and maintain a successful company.

Consequently, it would be better to view this more as an open source project. In this case, users would have to purchase the headset, the Arduino Uno, and an infrared LED. We would provide instructions on what other open source programs would be needed, and guide users on installing them and integrating them into a system. It would be very easy to deploy the custom programs in their current state. We would only need to package the code, and package a configuration module for remote tools. We have code that allows a user to read the infrared LED signatures off of any remote performing any function. We would need to make this code into an easy to use module that would automatically fill in the values needed to integrate these operations into our system. That would take a few man-hours, but it could be done relatively easily.

The main "manufacturing" would, in essence, only be a few steps that the current project team could do given a few more weeks. We would simply need to make a website to deploy the project package for download, provide instructions on installation and customization, and create a forum that would allow us to provide support for the product. The forum could also serve as a place that would allow users to share their customizations, and improvements. Our system would take on the spirit of a true open source project. This means that there would be no cost associated with "manufacturing" our product besides man-hours. Due to the fact that there is no need for extra workers, it would just be members of the project team. So, in reality, there would be no "manufacturing" cost at all.

9. Marketability

As mentioned in the previous section, we are taking the approach of an open source project. As such, we are not pursuing any profit. Because there is no cost deploying our project this is not a problem.

Luckily, this is actually a marketable idea. There is a big open source community on the Internet that shares projects and improves them. The only marketing we would need is to post our project on websites that publicize these open source systems. Websites such as www.instructables.com feature open source projects just like ours. Project creators provide instructions to install their program and encourage people in the open source community to customize it and make it better. There would be no mass marketing needed. The only marketing needed would be postings on websites like www.sourceforge.net and on the forums of products that we are featuring in our system. We would post in forums for Arduino, Emotiv, and Processing. The people that visit these forums are the exact people that would download and install our system and try to make it better. We are lucky that there is a receptive community for our system and it is localized in places that are easy to access.

One of the best things about having people work on our project is that it would give us a large amount of people that will voluntarily develop our project and improve it. These same people will also provide support for our project in our forum on our website. We would have users effectively joining our development and support team. Additionally we would be able to release updates to our product that utilize the improvements made by the users. This is a common approach and is used by companies such as Ubuntu that deploy open source products. This would only occur two or three times a year, and thus would not take that much time from the members of the project team.

Releasing our product in the open source community can allow our small "company" to use essentially free labor to improve the product much more than what would be possible for a propriety system being sold on the consumer market.

10. Individual Team Member Discussions

10.1. Rich Roman

10.1.1. Overview Discussion of the Project

This project integrates a number of consumer products and open source software to open up a new medium of communication and new methods for user convenience. We have enabled long-range communication by thought using the Internet. We have utilized new technology to enable a new mode of completing actions. Brain waves can now be used to communicate over great distances, and control remote appliances around the house. This is all accomplished using the Emotiv Epoc EEG headset, Twitter, and a custom remote tool that features an infrared LED and the Arduino Uno microprocessor. We utilized programs such as Mind Your OSC's, Processing, Python, and Arduino to form seamless communication and cooperation between the components listed before.

The end result took on a different form than what was initially proposed. The initial idea consisted of controling an Apple iPhone using a custom made application and the Emotiv Epoc headset. The Apple iPhone turned out to be a road-block. However we wanted to still use the "thought based" technology provided by the EEG headset. Utilizing the Twitter API we were able to still send messages that interacted with smartphones.

Further exploration of the product led the team down a few dead ends before providing a clear path. We attempted to use the Arduino Ethernet Shield to communicate over the Internet. It was abandoned altogether because of the lack of convenience that it afforded the user. We also attempted to make our own headset and replicate the functions of the Emotiv Epoc. This was a failed attempt due to lack of project time and resources. This path can be pursued further by a team with more time and resources (we attempted this towards the end of the semester).

Even at the beginning of the semester we understood that this project would most likely not yield a viable consumer product. Due to the fact that we were utilizing very cutting edge technology, the main goal was to produce something that can be built upon further by other project groups or other developers. We were able to produce a working prototype that shows that development can be done using brain waves to accomplish tangible actions. As such, we chose to take the approach of an open source project. We wanted to have a system that could be made available to the public for no cost. This will encourage people in the open source community to take on the task of further development. Having a large sampling of possible users and developers would allow this project to further improve and gain more features after the term of this project. Being as this was more of an exploratory project, this approach fits right in with the mindset of the group. One day, products that allow users to control products accurately and effectively with their minds will be on the market. Hopefully we will be able to say that our project helped to pave the way for those products to be developed.

10.1.2. Detailed Discussion of Pertinent Sub-Systems

My role in the project team was that of exploration and documentation. I served the role of discovering the capabilities and limitations of the components in use. I had to experiment with the consumer hardware to discover how what capabilities we could use, and where the products needed to be extended. I had to research and learn what could be done with the hardware components that would be used. I found the limitations in the Arduino Uno in Internet connectivity along with the limitations present with communicating with the necessary software components. I also introduced and eventually abandoned the Ethernet Shield for the Arduino Uno as well. It was introduced because it was the only way to allow the Arduino Uno autonomous network connectivity. However, I soon realized that the inconvenience it presented (already documented) overcame the benefits of its inclusion in the project. I also explored interfacing our system with the Twitter API. After trial and error with different programming techniques, I found the best way to connect our components to the Twitter API. The information and methods I discovered were used in both the Tweeting Module, which I designed and implemented, and the Tweet Recognition Module. I also assisted Mrunal in the construction of our headset prototype, which upon further research and exploration was eventually discarded.

I also assumed the role of providing full documentation. A successful project must be accompanied with comprehensive documentation in order to give the project team the credit that it deserves. I provided and compiled most of the official documentation. I compiled the proposal, design reports, and most of this document in the best way to communicate the strengths, weaknesses, accomplishments, failures, and heartaches that the project team encountered along the way.

I am extremely proud of the finished product that Mrunal and I have put forth. With hard work and determination, we were able to explore a new technology and make some headway in the field of brain control. Maybe one day our advisor Dr. Daut will have the ability to input the grade of this project into his computer just by thinking of the letter A.

10.2. Mrunal Shah

For the whole project I was heavily involved in making sure that we didn't hit any compatibility issues. We went through 5 iterations, each one having its own set of compatibility issues. Consequently we had to discard few design to achieve our desired goal. Since a private company makes Emotiv headset and since it's not open source, compatibility was a big issue. Also since the way Emotiv headset communicates with the computer is encrypted, we had to work around passing information through set of software each doing its own specific thing. Emotiv headset's working along with its tech specs has been heavily discussed in the sections above. Information from Emotiv Headset is passed to open source software named "Mind Your OSCs" which converts the information coming headset into packets of useable information, which can be used for processing. I was involved till this process and Richard worked on how after passing information to Processing, connections to twitter was made so that data could be sent to someone far away. I worked on getting information back to Arduino board using custom python script that parsed information looking for certain keywords. How Python works and its detailed technical information is discussed in the sections above. Also I was heavily involved in prototyping of Arduino remote control using infrared LED. Using custom circuit I was able to get the waveform on the oscilloscope for further analysis. 36 relevant bits of information were exchanged at 38kHz frequency. I was able to successfully find out distances between each pulse and program that on Arduino so that we could use it as remote control.

Appendices

Appendix 1 – List of Equipement

1. Emotiv Epoch EEG Headset
2. Arduino Microcontroller, Board Model : UNO
3. IR-LED, Radio Shack SKU/No Cat 276-0142
4. 38kHz IR receiver module, Radio Shack SKU/No Cat 276-640
5. Breadboard obtained from the lab
6. Oscilloscope
7. Laptop

Appendix 2 – Program Code

1. Sampling Algorithm: used by Emotiv Epoc EEG Headset[1]

```
FixTimeStamps(SAMPLING_RATE)

miliSecondsPerSample 1000=SAMPLING_RATE

referenceTime  nil

n 0

for all sample do

if referenceTime = nil then

referenceTime getTimeStamp(sample)
```

```
    end if
    setTimeStamp(referenceTime+(n _ miliSecondsPerSample))
    n  n+1
    end for
```

2. Artifact Algorithm: used by Emotiv Epoc EEG Headset[1]

```
 findPeaks( y, mean, stdDev)
 PEAK_LIMIT  <- 3.5 .stdDev
 CLUSTERING_DISTANCE  <- 300
artifacts  ;
lastArtifact   (1,1)
for t = 1 to n do
if yt > PEAK_LIMIT then
 if t  CLUSTERING_DISTANCE <- lastArtifact
then
 {extend the last artifact}
lastArtifact
 else
{add a new artifact}
 if lastArtifact = ( 6 1,1) then
artifacts   artifacts [ lastArtifact
lastArtifact   (t, t)
 end if
 end if
 end if
end for
if lastArtifact = ( 6 1,1) then
artifacts   artifacts [ lastArtifact
end if
return artifacts
```

3. Arduino Sketch: used in Remote Operation Module

Note: comments that explain code are on lines starting with //

```
int sentDat; // defining sentDat object
int IRledPin =  13;  // defining pin 13 as IR-LED pin
void setup() {
  Serial.begin(9600);   // telling Arduino to communicate using port 9600
  pinMode(IRledPin, OUTPUT); // tell Arduino that pin 13 is IR-LED pin
}
void loop() {
  if (Serial.available() > 0) { // start sending information when the port become
                                                       available
    sentDat = Serial.read();
    //red control
    if(sentDat == 'a'){ //if character "a" comes from python code send pulse IR-LED
     SendTVCode();
    }
    }
}
// This procedure sends a 38KHz pulse to the IRledPin
// for a certain # of microseconds. We'll use this whenever we need to send codes
void pulseIR(long microsecs) {
 // we'll count down from the number of microseconds we are told to wait
 cli();  // this turns off any background interrupts
 while (microsecs > 0) {
   // 38 kHz is about 13 microseconds high and 13 microseconds low
  digitalWrite(IRledPin, HIGH);  // this takes about 3 microseconds to happen
  delayMicroseconds(10);        // hang out for 10 microseconds
  digitalWrite(IRledPin, LOW);   // this also takes about 3 microseconds
  delayMicroseconds(10);        // hang out for 10 microseconds
              // so 26 microseconds altogether
  microsecs -= 26;
 }
 sei();  // this turns them back on
}
void SendTVCode() {
 // This is the code we got from out remote by pulsing On/Off.
//pulseIR(microseconds); we find out microseconds from Oscilloscope.
 pulseIR(8840);
 delayMicroseconds(4280);
 pulseIR(620);
 delayMicroseconds(460);
```

```
pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1600);

pulseIR(600);

delayMicroseconds(1600);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1600);

pulseIR(600);

delayMicroseconds(1600);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);
```

```
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1600);
pulseIR(600);
delayMicroseconds(1600);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1600);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(500);
pulseIR(600);
delayMicroseconds(1600);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1600);
pulseIR(600);
delayMicroseconds(480);
```

```
pulseIR(620);

delayMicroseconds(1600);

pulseIR(600);

delayMicroseconds(1600);

pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(1600);

pulseIR(600);

delayMicroseconds(1600);

pulseIR(620);

delayMicroseconds(1600);

pulseIR(600);

delayMicroseconds(39820);

pulseIR(8860);

delayMicroseconds(2120);

pulseIR(620);

delayMicroseconds(28160);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);

delayMicroseconds(28140);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);
```

```
 delayMicroseconds(28140);
}
```

4. Python Code: used by Tweet Recognition Module

Note: comments that explain code are on lines starting with #

```
import serial
import twitter
import time
#setup the serial port.
ser = serial.Serial('/dev/tty.usbmodemfd131', 9600, timeout=1)
#Access the Twitter API with your unique key
api = twitter.Api(consumer_key='dOnOE8J1zj6RWHlLf70w',
consumer_secret='D5wCQe1CUBqFwN6ilyuIvzOoVGI379yyUjbCBa88',
access_token_key='223248327-
WecHEZhwtxaKr3rCStmwy0G5n4uRCoGHSS60gEQm',
access_token_secret='QkmitdIRKIpJJ6cb4YHmAepgRcC6XEbTZHSd9JH2UI0')
while(1):
  #Look at @ mentions from Twitter api
  TweetAtMe = api.GetMentions()
  TweetList = [u.text for u in TweetAtMe]
  #Look at most recent @ mention
  RecentPost = TweetList[0]
```

```
print RecentPost

#splits the most recent post into discreet words

#and stores these words in a list

CurrentTweetList = RecentPost.split()

#checks to see what #colors are in the list

#and sends serial commands to the Arduino accordingly

#currently checking for three colors, but capable of 4 (see below)

if "#red" in CurrentTweetList:

  print "list contains", "#red"

  ser.write('a')

else:

  print "no white"

  ser.write('b')

if "#nored" in CurrentTweetList:

  print "list contains", "#nored"

  ser.write('c')

else:

  print "no red"

  ser.write('d')

if "#white" in CurrentTweetList:

  print "list contains", "#white"

  ser.write('e')

else:
    print "no white"
    ser.write('f')
# Conditional statement for unused color port
# Set it to be whatever color you like
# if "#blue" in CurrentTweetList:
#    print "list contains", "#blue"
#    ser.write('g')
# else:
#    print "no blue"
#    ser.write('h')
  #Checks the current status of the rate at which you are allowed to access Twitter
  #Delays for minimum rate limit wait time plus one second
  #This allows you to ping twitter as much as possible without exceeding the limit
and getting blocked
  rateLimitWait = api.MaximumHitFrequency()
  print rateLimitWait
  time.sleep(rateLimitWait + 1)
```

5. Processing Code: used by Tweeting Module

Note: comments that explain code are on lines starting with //

```
import oscP5.*;
import twitter4j.conf.*;
import twitter4j.internal.async.*;
import twitter4j.internal.org.json.*;
import twitter4j.internal.logging.*;
import twitter4j.auth.*;
import twitter4j.api.*;
import twitter4j.util.*;
import twitter4j.internal.http.*;
import twitter4j.*;
        //setting of OAuth variables as Twitter account credentials
static String OAuthConsumerKey = "hOKCMtICQzCPlDQLYcBkg";
static String OAuthConsumerSecret =
"lKgH0TxbYVO6n6TS52iynFHLQTMAcgZCsuI2bMwwy08";
static String AccessToken = "526114727-
5ZBaRMuM3doWuUD27AbXoH9qG8ctpm2T0lGmDJQc";
static String AccessTokenSecret =
"JfZm7VI2AcBVHAE8c1e3oUvceBCZo0u7JLdcDgzXQ";
        //initialization of environment variables for other programs
Twitter twitter = new TwitterFactory().getInstance();
Arduino arduino;
OscP5 oscP5;
float disappear = 0;
float push=0;
float pull=0;
float lift = 0;
```

```
}

        //initial program setup
void setup() {
size(125, 125);
frameRate(10);
background(0);
println(Serial.list());
loginTwitter();
        //functions that capture OSC values from Mind Your OSC’s
  oscP5 = new OscP5(this, 7400);
  oscP5.plug(this,"getDisappear","/COG/DISAPPEAR");
  oscP5.plug(this,"getPush","/COG/PUSH");
  oscP5.plug(this,"getPull","/COG/PULL");
    oscP5.plug(this,"getLift","/COG/LIFT");
}
        //functions that take OSC values and assign them to a variable
void getDisappear (float theValue) {
disappear = theValue;
  println("OSC message received; new disappear value: "+disappear);
}
void getPush (float theValue) {
push = theValue;
  println("OSC message received; new push value: "+push);
}
```

```
void getPull (float theValue) {
  pull = theValue;
  println("OSC message received; new pull value: "+pull);
}
void getLift (float theValue) {
  lift = theValue;
  println("OSC message received; new pull value: "+lift);
}
        //function that logs into Twitter using OAuth
void loginTwitter() {
twitter.setOAuthConsumer(OAuthConsumerKey, OAuthConsumerSecret);
AccessToken accessToken = loadAccessToken();
twitter.setOAuthAccessToken(accessToken);
}
        //utility function for loginTwitter()
private static AccessToken loadAccessToken() {
return new AccessToken(AccessToken, AccessTokenSecret);
}
        //function that posts a tweet to Twitter
void postMsg(String s) {
try {
Status status = twitter.updateStatus(s);
println("new tweet --:{ " + status.getText() + " }:--");
```

```
}
catch(TwitterException e) {
println("Status Error: " + e + "; statusCode: " + e.getStatusCode());
}
}
     //main function of program
void draw() {
background(0);
float count = random(50); //random number generator
text("simpleTweet_00", 18, 45);
text("@msg_box", 30, 70);
     //pre-loaded tweets
String pushMes= "@mrunalshah6";
String disMes= "@rlroman90 help me";
String liftMes = "I'm tweeting by thinking!";
     //threshold determinators – tweet a simple message if OSC value is greater
     //than threshold
if(disappear >= 0.5) {
  postMsg(disMes);
}
 else if(push >= 0.5) {
  pushMes = pushMes + “ count” + “ #red”; //add count to tweet so each submitted
             //tweet is unique and will not be rejected by the Twitter API
```

```
                //#red is the keyword that turns on and off the television
   postMsg(pushMes);
  }
  else if(pull >= 0.5) {
 pushMes = pushMes + “ count” + “ #white”; //#white is the keyword that stops the
                                                                  //television operation
   postMsg(pushMes);
  }
  else if(lift >=0.5) {
   postMsg(liftMes);
  }
}
```

6. Baseline Removal Algorithm: used by the Emotiv Epoc Headset [1].

The purpose of the algorithm is to remove the mean of recording, so that values of the signal would be distributed around zero.

```
size <-64
initBuffer(buffer, size)
for $i$ = 1 to $n$ do
push(buffer, $xi$)
if $n$ _ size then
```

```
size   *xi* <-mean(buffer)
**end if**
**end for**
**return** *y*
```

References

1. EEG's Signal Processing and Emotiv's Neuro Headset by Andre Hoffmann

   http://data.text20.net/documentation/thesis.emotivsp.pdf

2. Arduino – Arduino Board Uno

   http://arduino.cc/en/Main/ArduinoBoardUno

3. Emotiv Software Development Kit – User Manual

   UserManual.pdf

4. Mind Your OSC's

   http://sourceforge.net/projects/mindyouroscs/

5. The Open Sound Control 1.0 Specification

   http://opensoundcontrol.org/spec-1_0

6. Processing

   http://processing.org/about/

7. oscP5 Processing Library

   http://www.sojamo.de/libraries/oscP5/

8. Twitter4j Processing Library

   http://twitter4j.org/en/index.html

9. OAuth

   http://oauth.net/about/

10. Python

   http://www.python.org/about/

11. pySerial Python Library

   http://pyserial.sourceforge.net/

12. simplejson Python Library

   http://pypi.python.org/pypi/simplejson

13. JSON

   http://www.json.org/

14. httplib2 Python Library

   http://code.google.com/p/httplib2/