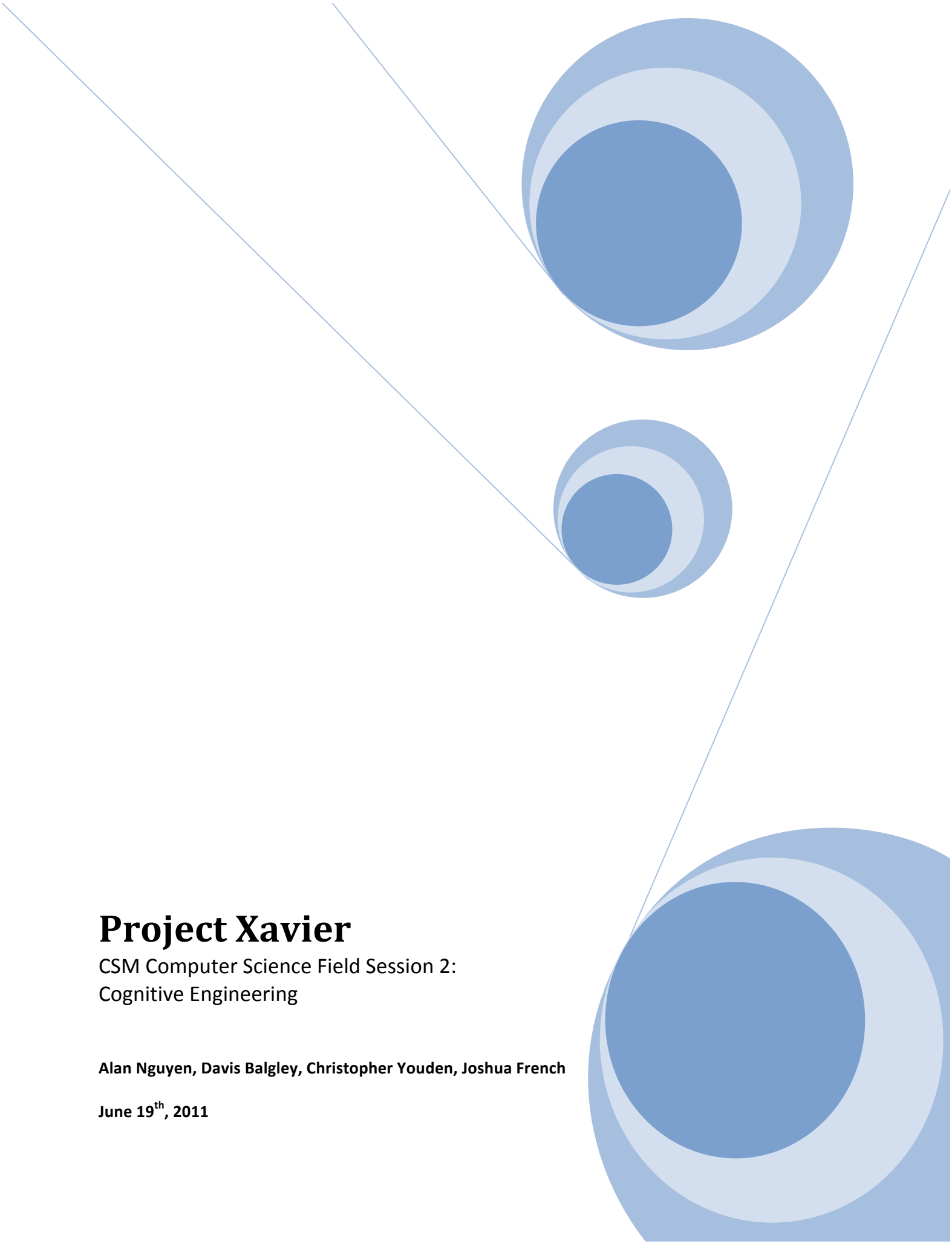

Project Xavier

CSM Computer Science Field Session 2:
Cognitive Engineering

Alan Nguyen, Davis Balgley, Christopher Youden, Joshua French

June 19th, 2011

Table of Contents

Abstract

The human brain is a complex computing machine. As such, research projects involving analysis of brain signals can become difficult to perform without assistance from analysis software. The Smart Helmet research project seeks to augment current technologies to provide the physically disabled with mobility devices controllable by thought. This involves training a program to recognize particular cognitive signals as commands to control the devices. Project Xavier began as collaboration with Smart Helmet to develop software that efficiently analyzed electroencephalogram signals. The project has since evolved into developing a proof of concept application demonstrating the ability for a program to classify and identify signals using a machine learning algorithm.

This project's approach to developing a proof of concept application involved using the Java programming language and the Java Swing library to develop an intuitive graphical user interface. Signals are graphically represented on the graphical user interface as waveforms using the Java Open Graphics Library. For ease of use, the Xavier Platform implements a drag and drop system for applying effects to signal data. The use of Jython to develop the drag and drop system facilitated the addition of a simple form for creating custom effects. Classifying and identifying signals used an assisted machine learning algorithm that implements k-means clustering.

To comprehend and utilize the complexities of the human mind we use technology to further our studies into how it functions. Project Xavier demonstrates a simple program for classifying and identifying signal data that can potentially be applied to cognitive projects. In the context of the Smart Helmet project, The Xavier Platform, the program, is trainable to separate and identify different thought command signals from collected data. This is but a single application of classifying signals. The Project Xavier Team hopes to provide a starting block for various greater projects.

Introduction

Working with brain signal data can be complex and difficult to do without analysis software. The client, the Smart Helmet research group, aims to augment brain signal collection and processing technologies to provide the physically disabled with mobility devices controllable by thought. The Smart Helmet Research Group required a program equipped with a set of tools for analyzing electroencephalogram (EEG) signals for their experiments with signal data. The program was expected to be written in Java and also to include an intuitive graphical user interface (GUI). The GUI required the ability to display and analyze signals. This was to be accomplished through the inclusion of requested effects that could be applied to selected portions of signal data. The program also required the ability to categorize and identify specific brain signal patterns. The project has since evolved into developing a proof of concept application that still includes the above-mentioned features.

Requirements

A. Functional Requirements:

- Accept EEG file type
- Ability to view multiple brain wave signal channels from different electrodes
- Ability to scroll through the recorded brain wave to view different areas of the signal
    - Brain waves in separate electrodes should scroll together
- Graphical User Interface
    - Ability to dock/undock components
    - Toolbox with effects/editing tools
    - Visual representation of signal data
- Ability to edit the brain wave tracks
    - Cropping
    - Scaling
    - Save/Load
- Ability to apply effects to the individual signal channels (e.g.):
    - Fast Fourier Transform
    - Inverse Fast Fourier Transform
    - Windowing Functions
    - Custom Effects
- Ability to recognize specific waveforms
- Must be able to save/load data (i.e. signals, applied filters, recognized waveforms, and any other changes)

B. Non-Functional Requirements:

- GUI:
    - User friendly
    - Light text on dark background
- All code must be written in Java
    - Code should be easy to follow for non-programmers
    - Code must be modular so the program can be expanded upon
- Use of SVN as a repository for code development
- Write code side by side with client

C. Scope

The scope of the project involved creating an easy to use GUI with signal analysis tools. The final application can display a graphical representation of data parsed from a WAV or formatted TXT file. The application allows users to apply effects to parsed signal data through the use of a visual drag and drop system. Users can use the effects included in the program or define their own custom effects. The signal data can be used in conjunction with the program's machine learning algorithm to categorize and/or identify the data as a specific pattern.

System Design

Figure 1 below shows how the application would accept and display data obtained from an EEG. The application is then able to process the data via built-in or user-defined effects and display the results. An MLA can be applied to the processed data in order to identify occurrences of known signal patterns. The individual components will be discussed in the following sections.

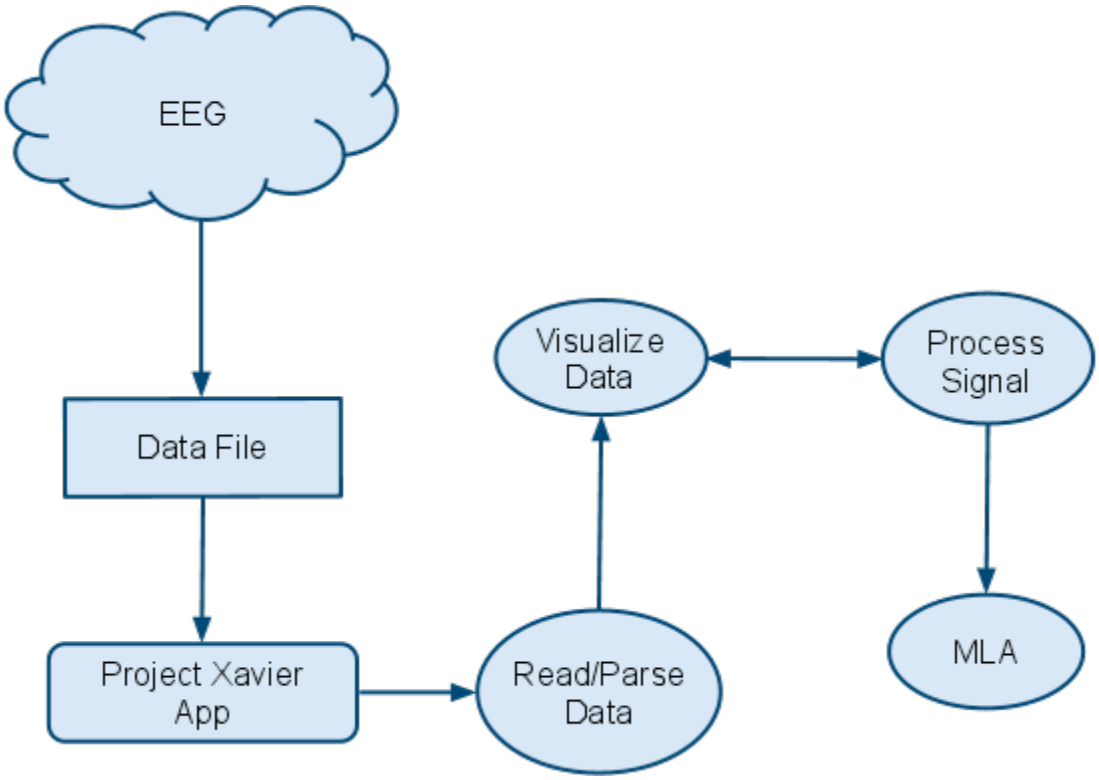

Figure 1. High Level Overview

Figure 2 illustrates how the user is able to use the MLA. The user has the option to "teach" the application a new signal or analyze a signal based on previously taught data. The MLA is an assisted learning algorithm that implements k-means clustering.

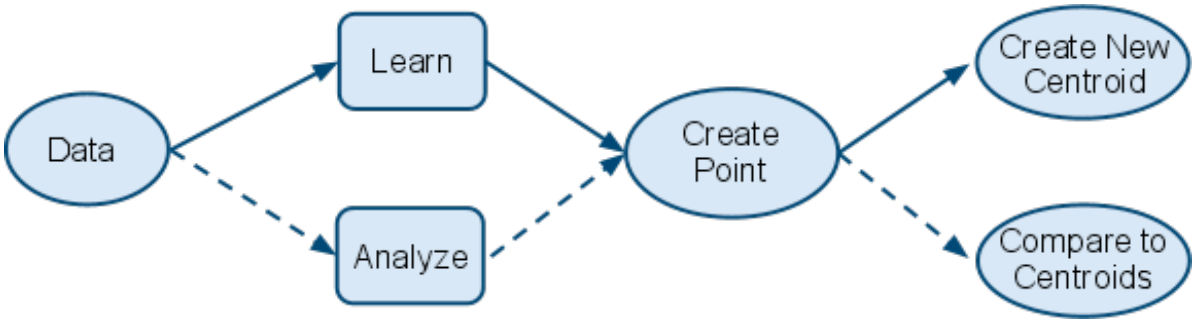

Figure 2. Machine Learning Algorithm Overview

When a new signal is taught to the algorithm, an identifier must be specified. This identifier defines what the signal represents. Once the signal has an identifier, its amplitude data is split into three equally sized segments. For each segment, the data is averaged. This produces three Cartesian coordinates composing a point which represents the signal. If this point is the first signal recorded for the identifier, it is labeled as the centroid for that identifier and saved. If a centroid already exists for the identifier, the newly created point is averaged into the existing centroid, and updated.

When a signal is analyzed, it is subjected to the same process it would undergo if it were being learned. The difference is that the user is not prompted to specify an identifier nor is a centroid created or updated. Instead the algorithm takes the point representing the signal data and compares the Euclidean distance between it and saved centroids. The point in question is associated with the nearest centroid and the algorithm reports that centroid's identifier. This method of creating centroids through averaging and identifying the closest centroid is a form of k-means clustering. This algorithm is a basic and simple one that demonstrates the application's potential.

Figure 3 below illustrates the communication between displays in the application. The Visual Editor, which receives data from the Raw Signal Display, provides an easy way to apply effects to the raw signals. The Processed Signal Display shows the results of raw signals after the application of effects. Processed signal data can be sent to the Visual Editor to undergo further user manipulation if necessary.

Figure 3. Signal Display Interactions

Figure 4 below explains how effects work in The Xavier Platform. Effects are applied to raw signals to alter their data for analysis. A processed signal can undergo further processing should the user wish to do so. Users may use preset effects in the program or define their own. For custom effects, a dialog allows users to define mathematical functions to apply to signal data. Custom effects can be saved for later use. Some examples of preset effects in the program include the Fast Fourier Transform and the Blackman Window Function.

The user is able to apply various effects to multiple signals in the Visual Editor. The Xavier Platform does not perform the calculations involved for each effect until the user decides to start it. The user begins processing via a run button or from the menu bar. Once processing completes, the results are displayed in the Processed Signals Display. Results can also be re-added to the Visual Editor and have more effects applied.

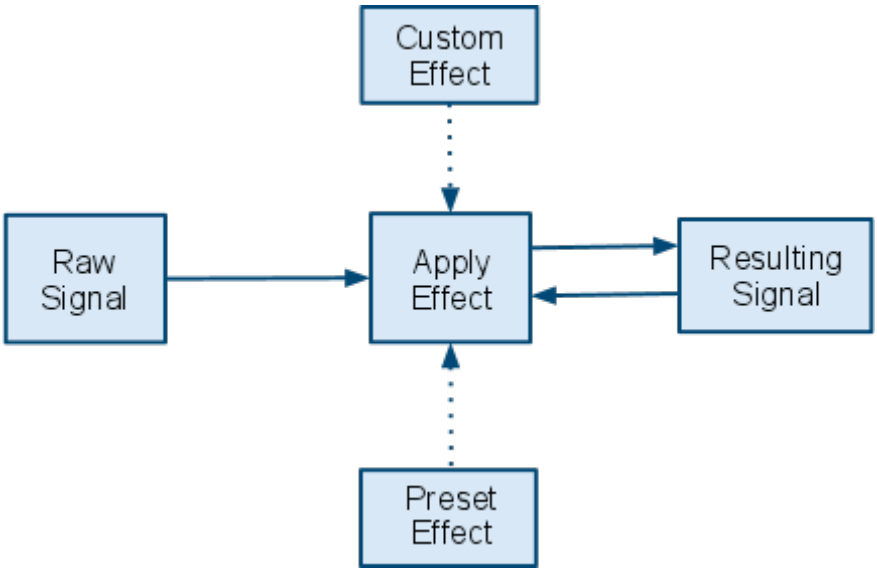

Figure 4. Signal Processing

Implementation

The program was developed using the Eclipse IDE. The client initially requested for the program to be developed using Oracle's NetBeans IDE because it features a module system and windowing API. These features were meant to ease implementation of new components in the application and provide a windows docking/undocking feature. NetBeans proved to be problematic however. The IDE did not properly communicate with the subversion repository making it unreliable for storing and maintaining versions of the code. Troubleshooting the subversion issue and learning the IDE, its module system, and windows API began cutting into the allotted work time. As such, the decision was made to move development to Eclipse and forego NetBeans since the conveniences it was supposed to provide were not functional.

The client specifically requested for development to be done using the Java programming language. A C-based language would have been more efficient for processing signal data, but the client expressed a desire to continue expanding the project at a later date and was more comfortable programming in Java.

The modular effects system implements Jython. To create custom effects, it was necessary to use a scripting language that could easily read and create scripts in lieu of the user's knowledge of computer programming.

Taking a cue from DSPExplorer, another signal processing application, the module system was designed to enable the visualization of the effects that the user wishes to apply to signals. Additionally, this method allowed the program to process multiple signals at a time. In order to provide effects, knowledge behind various effects would have had to be learned. Implementation time was saved as the algorithms for the preset effects were provided by the author of DSPExplorer.

OpenGL was chosen for the graphical representation of signal data. This decision was made with the client's preference for, and prior experience with, OpenGL in mind. Because OpenGL uses the C programming language, the project makes use of a wrapper library known as Java OpenGL (JOGL). This makes OpenGL usable in a Java application.

Audio WAV files were used in place of EEG files because EEG files were unavailable. Reading WAV files in Java is possible but difficult to implement. Code provided by Greensted[3] gave the program an easy method to read WAV files.

System Testing

Acceptance Tests:

- Verify that signals are displayed correctly
    - With and without effects
- Verify that multiple signals can be displayed
- Verify that signals can be selected
- Verify that signals can be scaled
    - Verify that scrolling is functional
- Verify that filters are producing correct results
- Can the user save/load a project?
- Can the user save/load custom created filters?
- Can the user teach a signal to the application's MLA?
- Can the application recognize taught signals?

Integration Testing:

- Being a modular application, it is important to ensure that each individual component of the application is communicating with the other components. This includes communication between the application and input/output files, effects and raw signals, and processed signals with the results panel.

Results

The main goal for this project was to design an application that would input signal data, display the data, include tools to apply effects to the data, and display the result. That result would then be taught to the program, or analyzed, using an MLA. The application is able to read in various file types via a file parser. The signals, both raw and processed, are displayed via OpenGL. The processed results are able to be taught, or analyzed, via an assisted MLA using k-means clustering.

The application was subject to various constraints. The client was unable to neither provide nor produce results that would be used with their particular EEG. Upon doing research, it was discovered that each EEG stores its data differently. Without knowing what the particular device is, it becomes impractical to create an import function that parses EEG data. Due to the inaccessibility of the EEG, the program was changed into a proof of concept application and was set up to work with WAV files. The use of WAV files became an analog to EEG data. The underlying concept of categorizing and identifying signal patterns remained unchanged.

Future Direction

Given the project's scope and limited access to necessary hardware, the application was developed with the idea of extending its functionalities. Presently, the application can read in WAV and formatted TXT files. The file parser can be extended to accept other file types. The module system allows for the creation and extension of custom modules.

Another possible addition to the application would be to implement a database for MLA data as opposed to the flat file system that it currently uses. Flat files, given enough data, become bulky and can take up large amounts of disk space. Flat files may also affect performance of the application as it must traverse the entire contents for learning and analysis. Querying a database (e.g. SQLite) would be much faster and more efficient than scanning through a flat file. It would also reduce disk space usage. Since the application would most likely never be used concurrently with multiple users, SQLite could be an appropriate database. Should the need for concurrency arise, the use of MySQL would resolve that issue.

For analysis of signals, the application could be extended to run the MLA using spectral data as opposed to the current temporal data. By using spectral data, the user would be analyzing the data with respect to frequency instead of time. Pattern recognition—for extracting relevant information—is a subject of research in it's own right, thus, It would be advisable to research and implement a more efficient MLA.

Extension of the displays is possible as well so that the user may display data in different representations.

Conclusion

The human mind is an infinitely complex organ. Its subtle nuances and complex design make it a challenge to analyze and impossible to completely understand. The applications of a program that can assist in “reading the mind” are truly limitless. The ability to move a wheelchair with but a thought is an incredible beginning to a future of fascinating technological applications.

Lessons learned:

- Splitting the work between people has its benefits and drawbacks
    - Integration can be arduous
    - Increased productivity
- Visualization on a whiteboard is very helpful
- Take breaks
- Independent work time away from the team can be beneficial
- Make realistic goals
- Constant change in a project is frustrating
- Attain concrete design specifications before coding

Glossary

API – Application Programming Interface

Assisted Learning Algorithm - As opposed to unassisted learning algorithm, it requires user input to provide a basis for the algorithm to learn from.

DSP – Digital Signal Processing

EEG – Electroencephalogram; device that reads changes in voltages in brain activities

Effect – The mathematical manipulation of signal data to alter the signal.

GUI – Graphical User Interface

IDE – Integrated Development Environment

JOGL – Java Open Graphics Library; java wrapper library for native OpenGL Libraries

MySQL – A relational database management system

SQLite – An embedded relational database management system

Spectral – Data with respect to frequency

SVN – Subversion

Temporal Data – Data in terms of amplitudes over time

The Xavier Platform – The name of the application developed by the Xavier Project Team

Bibliography

[1] Barashev, Dmitry, Alexandre Thomas, et al. *GanttProject Home*. 2003. Web. May 2011. <http://www.ganttproject.biz/>.

[2] *DSPExplorer*. Web. 20 May 2011. <http://www.ae6ty.com/DSPExplorer.html>.

[3] Greensted, Andrew. The Lab Book Pages - Java Wav File IO. 25 Sept. 2010. Web. May 2011. <http://www.labbookpages.co.uk/audio/javaWavFiles.html>.

[4] Jones, Tim M. Artifical Intelligence: A Systems Approach. Jones and Bartlett Publishers. 2008.

[5] Khronos Group. *OpenGL - The Industry Standard for High Performance Graphics*. Web. May 2011. <http://www.opengl.org/>.