

UNDERGRADUATE RESEARCH

---

# **BCI TECHNOLOGY**

---

By: Marco A. Zuniga

Auburn University  
Prof: Christopher B. Harris  
Department of Computer Engineering  
Spring 2019

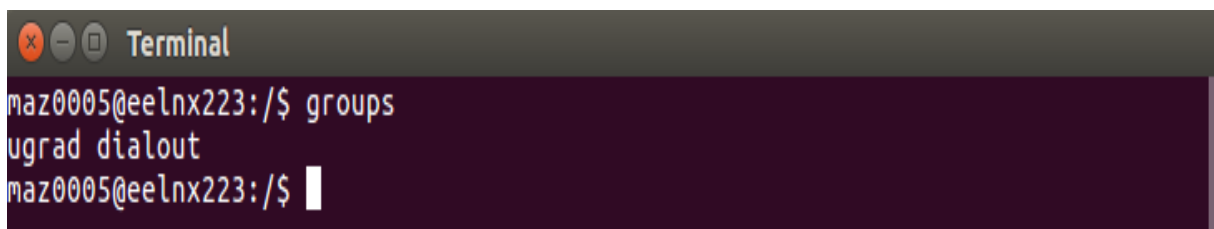
## **ABSTRACT**

This report contains everything I've come across during my undergraduate research. My research pertained to BCI technology and Electronencephalography (EEG), which is the method used to extract signals from a human brain. One will find all of the installation issues I've come across and how they were fixed. Following the installation process is where I left off in terms of extracting data from OpenVibe and using the data in an external C++ program.

## INSTALLING OPENBCI

To begin the semester, I installed the OpenBCI application. This process wasn't difficult at all and the installation guide can be found on OpenBCI's website. This step isn't required if OpenVibe is to be used. I worked with a brain-sampling EEG device called the Cyton Biosensing Board. This board is an 8-channel 32-bit signal acquisition board that reads signals from connected electrodes. All information regarding the Cyton Board can also be found on OpenBCI's website. Along with the Cyton Board comes a bluetooth dongle that receives data from the acquisition board.

I encountered my first issue when trying to access the USB-dongle from the OpenBCI application. I began by installing a recommended D2xx driver that wasn't even required. I kept trying the typical Google approach and searched the issue I was having. After several days I finally came across a forum that spoke about someone else having an issue accessing their USB device and figured out the issue. The issue was not being part of a specific Linux "group" which granted access to certain USB devices, including the Cyton USB dongle. At the start of the semester I wasn't very experienced with Linux. I could navigate through the terminal fairly well, but there was a lot about Linux that I didn't know. Being a typical Windows user, when a device is plugged into a computer, one is given access the connected device. Linux, however, uses "groups" for security purposes. It's just a way to manage the permissions of certain users. Because I was on school computer, I was restricted from running numerous commands. I needed to contact the engineering network service department to add me to the 'dialout' group. Once I was added to the group, the USB-dongle appeared on the OpenBCI application. Figure 1 shows me running the 'groups' command in the terminal with conformation that I am part of the 'dialout' group.

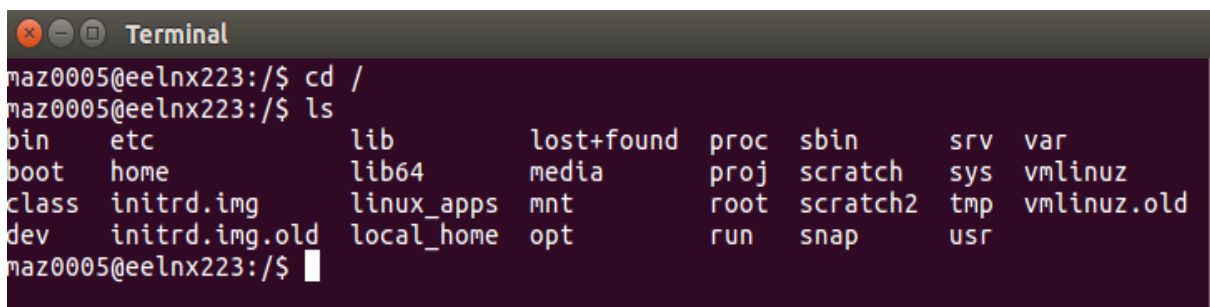
A screenshot of a Linux terminal window. The title bar shows a red close button, a grey minimize button, and a grey maximize button, followed by the word "Terminal". The terminal content shows a user prompt "maz0005@eeelnx223:/\$" followed by the command "groups". The output of the command is "ugrad dialout". The prompt "maz0005@eeelnx223:/\$" is followed by a white cursor bar.

```
maz0005@eeelnx223:/$ groups
ugrad dialout
maz0005@eeelnx223:/$
```

**Figure 1:** Groups that I am a part of

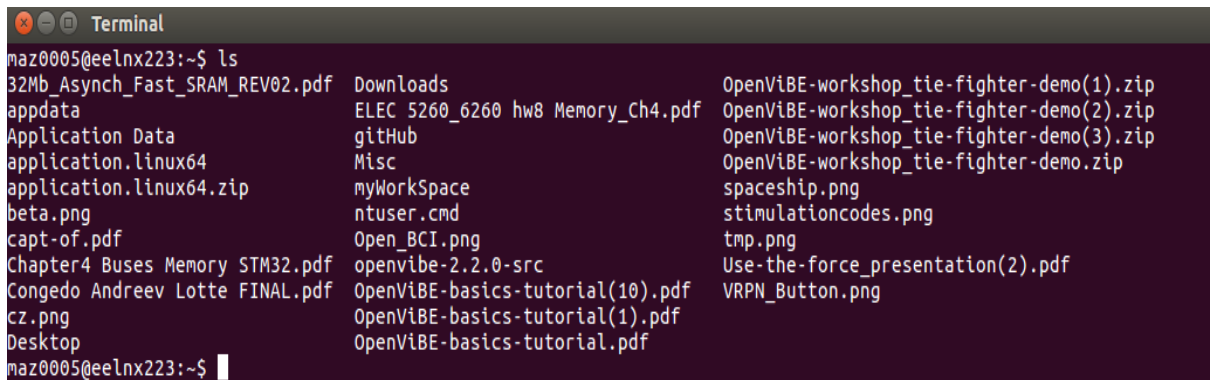
## INSTALLING OPENVIBE

The next step was to download OpenVibe which is a similar application to OpenBCI. By not having 'sudo' access, I was unable to run a specific 'shell executable file' named 'install\_dependencies.sh' that would download all of the dependencies needed to build OpenVibe. I contacted the engineering network service department again to see if they could run the executable for me, but were unable to run the executable in my home directory because 'root' cannot run out of a user's network shared homespace. The entire 'openvibe-2.2.0-src' directory was copied to the /tmp directory, which is in the root directory, and the executable was then ran. Again, this was all done by a member of the engineering network service department. By looking into the 'install\_dependencies.sh' file, I was able to see that the executable would store everything needed for building OpenVibe in the 'dependencies' directory which is within the 'openvibe-2.2.0-src' directory. I then erased everything that was in my dependencies directory, and copied everything from the dependencies directory within the 'tmp' directory, and proceeded to run ./build.sh. Figure 2 shows the 'tmp' directory available in the 'root' directory. By navigating into the 'tmp' directory, one would find an 'openvibe-2.2.0-src' directory. Figure 3 shows my homespace with the same directory. Figure 4 shows the 'dependencies' folder I was referring to.

A terminal window titled "Terminal" with a dark background. The prompt is "maz0005@eelnx223:/\$". The user enters "cd /" and then "ls". The output is a multi-column listing of the root directory contents: bin, etc, lib, lost+found, proc, sbin, srv, var, boot, home, lib64, media, proj, scratch, sys, vmlinuz, class, initrd.img, linux\_apps, mnt, root, scratch2, tmp, vmlinuz.old, dev, initrd.img.old, local\_home, opt, run, snap, usr. The prompt returns to "maz0005@eelnx223:/\$".

```
maz0005@eelnx223:/$ cd /
maz0005@eelnx223:/$ ls
bin      etc      lib      lost+found  proc  sbin    srv    var
boot    home    lib64    media      proj  scratch sys    vmlinuz
class   initrd.img  linux_apps  mnt      root  scratch2 tmp    vmlinuz.old
dev     initrd.img.old  local_home  opt      run   snap    usr
```

**Figure 2:** root directory

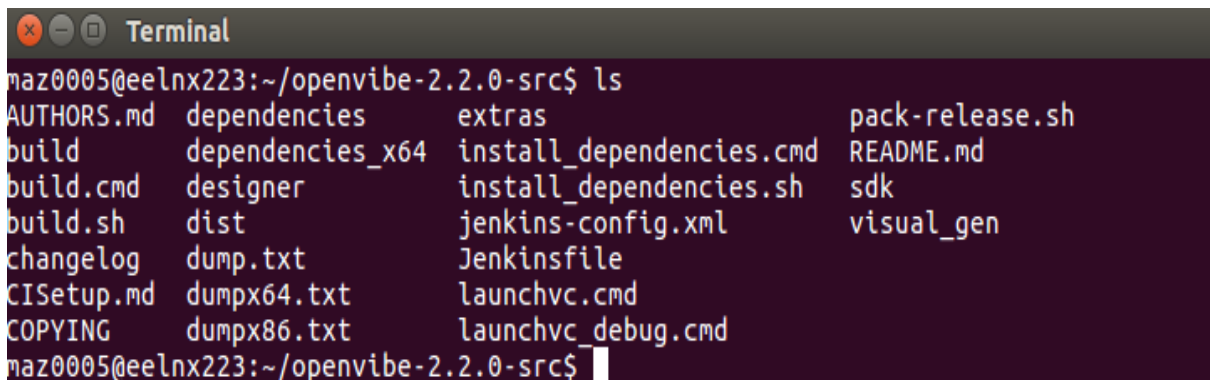


```

Terminal
maz0005@eeelnx223:~$ ls
32Mb_Asynch_Fast_SRAM_REV02.pdf  Downloads                                OpenViBE-workshop_tie-fighter-demo(1).zip
appdata                          ELEC_5260_6260_hw8_Memory_Ch4.pdf      OpenViBE-workshop_tie-fighter-demo(2).zip
Application Data                 gitHub                                  OpenViBE-workshop_tie-fighter-demo(3).zip
application.linux64             Misc                                    OpenViBE-workshop_tie-fighter-demo.zip
application.linux64.zip         myWorkSpace                            spaceship.png
beta.png                       ntuser.cmd                             stimulationcodes.png
capt-of.pdf                    Open_BCI.png                           tmp.png
Chapter4 Buses Memory STM32.pdf openvibe-2.2.0-src                     Use-the-force_presentation(2).pdf
Congedo Andreev Lotte FINAL.pdf OpenViBE-basics-tutorial(10).pdf        VRPN_Button.png
cz.png                         OpenViBE-basics-tutorial(1).pdf
Desktop                        OpenViBE-basics-tutorial.pdf
maz0005@eeelnx223:~$

```

Figure 3: OpenVibe directory



```

Terminal
maz0005@eeelnx223:~/openvibe-2.2.0-src$ ls
AUTHORS.md  dependencies  extras  pack-release.sh
build       dependencies_x64  install_dependencies.cmd  README.md
build.cmd   designer        install_dependencies.sh   sdk
build.sh    dist            jenkins-config.xml       visual_gen
changelog   dump.txt        Jenkinsfile
CISetup.md  dumpx64.txt     launchvc.cmd
COPYING     dumpx86.txt     launchvc_debug.cmd
maz0005@eeelnx223:~/openvibe-2.2.0-src$

```

Figure 4: dependencies directory

Once OpenVibe was installed, I began signal acquisition. The 'designer' tool is the main tool that deals with handling data received from specific BCI boards and also sending data to external applications. This tool, as well as other tools available from OpenVibe, are in the 'openvibe-2.2.0-src/dist/extras-Release' directory. Figure 6 shows an example of a design (i.e. can be found in /openvibe-2.2.0-src/dist/extras-Release/share/openvibe/scenarios/bci-examples/spaceship). Note that I was unable to successfully run the designer tool the first time I tried opening it. Figure 5 shows the code segment I had to comment out in the designer file. I believe it was trying to do something with MATLAB which wasn't installed on my computer. This design is going based off pre-recorded data to lift a simulated spaceship UP and DOWN. The application simulating the spaceship can also be found in the same directory as the 'designer' tool (i.e. openvibe-vr-demo-spaceship). More on this demo can be found on OpenVibe's website. By clicking "play" in the designer tool, and also having the spaceship simulator open, one will see that the spaceship will lift UP or DOWN based off of the 'beta' activity found in the recording. Electrodes will read different signals based where

they are placed on a human head. A 'beta' signal is just a signal that is between 12 Hz and 40 Hz. Physical movement or imagining movement should place one's brain in the "beta state" due to motor activity belonging in the beta frequency band. One can find the standard EEG 10-20 system used to identify electrode placement on the human head in the appendix section. In the design shown in Figure 4, I replaced the "GDF file reader" with an "Acquisition client" to attempt to lift the spaceship myself. Different tools within the designer application can be found through the search bar. An electrode was placed on the 'Cz' location of my head. Any available channel can be used for this electrode. A second electrode was placed on my nose (i.e. neutral on cyton board). A third electrode was placed on my forehead (i.e. ground on the cyton board). Once all connections were made, the signal acquisition server was opened (openvibe-2.2.0-src/dist/extras-Release/openvibe-acquisition-server.sh). Figure 7 shows the what the server looks like. By default, the port of the signal acquisition server and the acquisition client (in the designer tool) will be 1024. If a different port were to be used, one must assure that both port numbers are the same.

```
# Dump core...
# ulimit -c unlimited

# Tries to locate matlab executable from $PATH, and set the library path to the
corresponding matlab libs.
# if [ "`which matlab`" != "" ] ; then
#     MATLAB_ROOT=`matlab -e | grep "^MATLAB=" | sed -e "s/^MATLAB=/"`
#     MATLAB_ARCH=`matlab -e | grep "^ARCH=" | sed -e "s/^ARCH=/"`
#     MATLAB_LIBPATH="$MATLAB_ROOT/bin/$MATLAB_ARCH"
#     # echo Matlab libs expected at $MATLAB_LIBPATH
#     export LD_LIBRARY_PATH="$MATLAB_LIBPATH:$LD_LIBRARY_PATH"
# fi

LOCAL_BIN_PATH="/home/u3/maz0005/openvibe-2.2.0-src/dist/extras-Release/bin"
8,0-1 Top
```

**Figure 5:** Matlab section to comment out

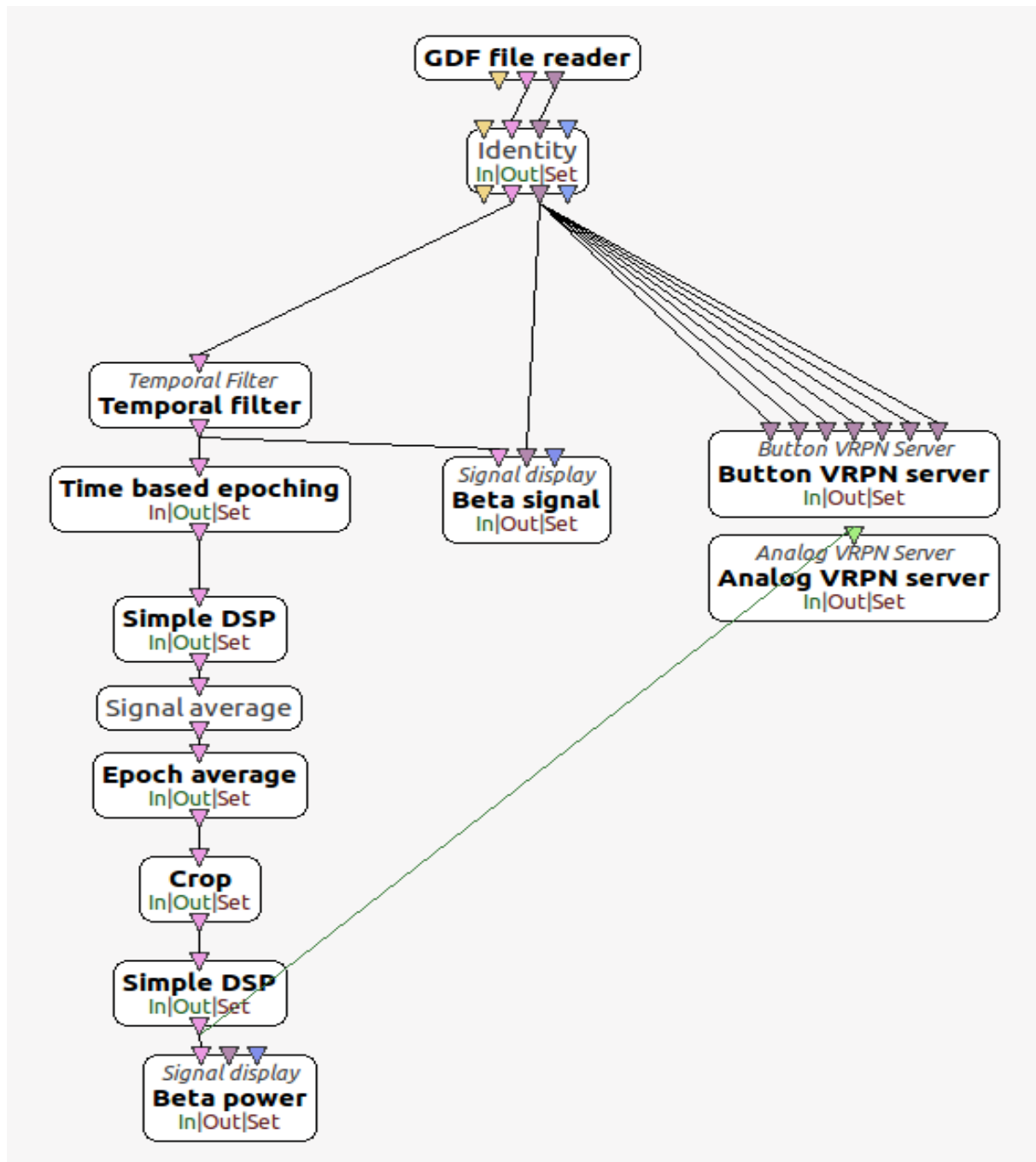
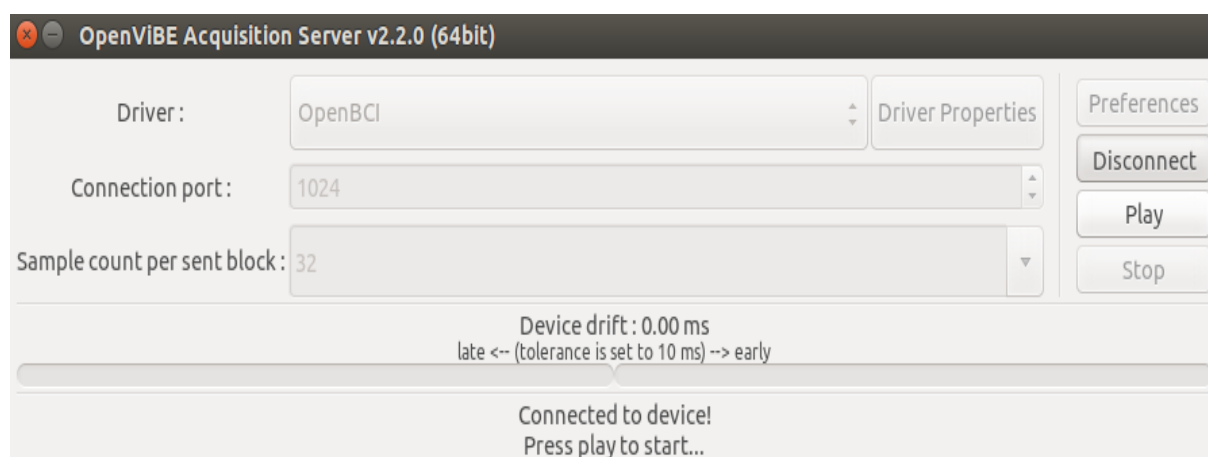


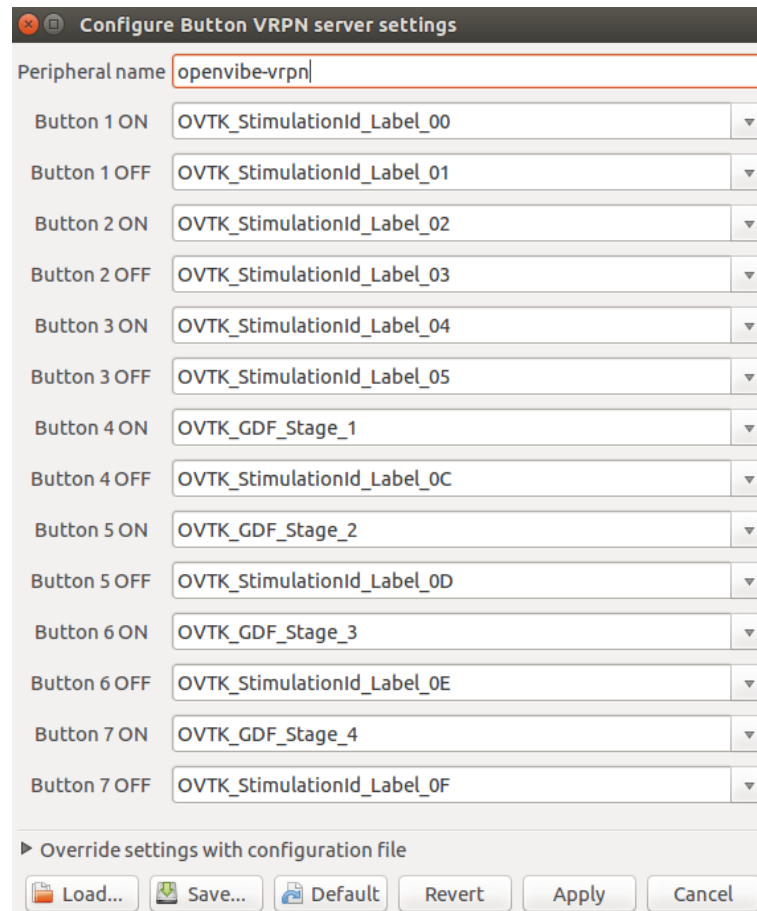
Figure 6: Spaceship design



**Figure 7:** 10-20 EEG System

Once everything was set up, I attempted to lift the spaceship. I was unsuccessful due to the incapability of creating the stimulants required by the spaceship program. The way the application works is it requires a specific stimulant before it can begin lifting the spaceship. Once the specific stimulant is found by OpenVibe, one is able to control the spaceship with the intensity of the of the beta activity found. Different "buttons" look for a specific stimulant. When connecting an external application to a VRPN server, a callback function will be called every time a button changes its state. When a specific stimulant is active, the button corresponding to that stimulant will be '1' and '0' when inactive. Figure 8 shows the buttons created for the spaceship demo. All of the buttons available for usage can be found when selecting a specific button in the VRPN Server and on OpenVibe's website . The output of the Beta Power box will indicate when the someone is moving their feet, or thinking about moving their feet. Note that the Beta Power box is just a signal display box that was renamed to indicate its purpose. I attempted to find the source code for the spaceship program, but was unable to find it. Although I was unable to lift the spaceship, I was able to see that when I moved my feet, the Beta Power box would display a HIGH signal, and would return back to the LOW state when I stopped moving my feet. Therefore, my conclusion was that my brain was producing signals in the beta frequency band, but I wasn't giving the right stimulant before I could lift the spaceship.



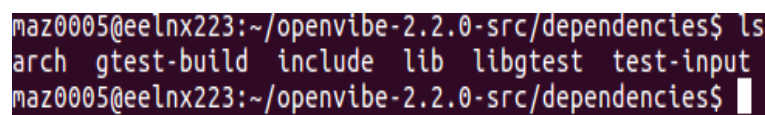


**Figure 8:** VRPN Buttons

## EXTERNAL APPLICATION

The next step was to get data from OpenVibe and do something meaningful with it. Note that my actual C++ program can be found in the Appendix section. This section is mostly to discuss the issues I had getting my code to compile. I first began by trying to assure I could compile my file with all of the dependencies needed from the OpenVibe libraries. There was an issue trying to compile due to the compiler not being able to find where certain objects were defined. OpenVibe recommends including all of the necessary header files in the <include> directory, but with restricted access in the root directory, I had to figure out a different way to compile my program. One alternative is to fully give the entire path of each header file when including it in your program (i.e all VRPN header files can be found in ~/openvibe-2.2.0-src/dependencies/include) and also link the required libraries when compiling with the "-L" flag (e.g. g++ -pthread -L/home/u3/maz0005/openvibe-2.2.0-src/dependencies/lib -lvprn

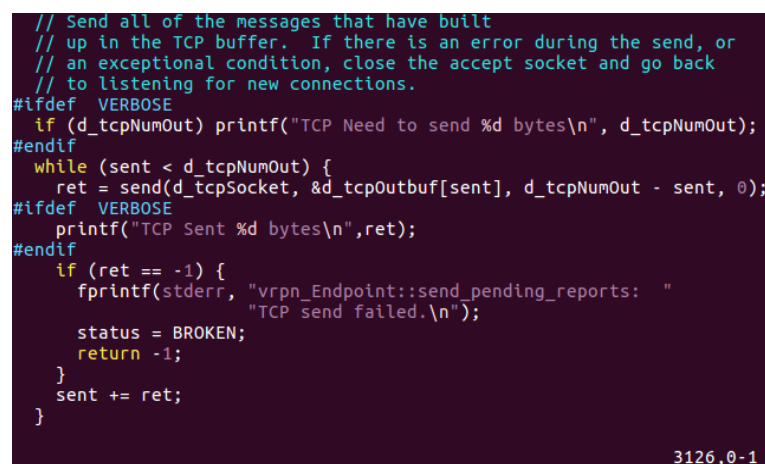
-L/home/u3/maz0005/opencvibe-2.2.0-src/dependencies/lib -lvprnserver myVRPN.cpp -o vrpn). OpenVibe has a tutorial on connecting external applications to the designer application. Even when I would specify where the necessary libraries were, the compiler still couldn't find where certain things were defined. Eventually, I looked in the "vrpn" directory (i.e. can be found in ~/opencvibe-2.2.0-src/dependencies/arch/vrpn) and found the C files where everything was defined. I then moved my file to this directory and included every C file corresponding to each header file included in my program. Using this approach allowed me to successfully compile my program. Figure 9 shows the arch, include, and lib folders in the same directory. The "vrpn" folder I mentioned previously is in the "arch" folder.



```
maz0005@eelnx223:~/opencvibe-2.2.0-src/dependencies$ ls
arch gtest-build include lib libgtest test-input
maz0005@eelnx223:~/opencvibe-2.2.0-src/dependencies$
```

**Figure 9:** Different directories with header and source code files

Now that I was able to successfully compile my program, my next task was to connect to the VRPN Button server in the designer application. More information on connecting an external application can be found on OpenVibe's website. This is where I ran into another issue. I pressed "play" in the designer application and also ran my executable, but there was issue with OpenVibe trying to send data to my application. I was able to see in the terminal, which was running the designer application, that there was something trying to access the application, but would throw an error. Figure 10 shows the code segment where the issue was happening. This is as far as I got in terms of trying to debug the issue (i.e. the following code segment can be found in ~/opencvibe-2.2.0-src/dependencies/arch/vrpn/vrpn\_Connection.C)



```
// Send all of the messages that have built
// up in the TCP buffer. If there is an error during the send, or
// an exceptional condition, close the accept socket and go back
// to listening for new connections.
#ifdef VERBOSE
    if (d_tcpNumOut) printf("TCP Need to send %d bytes\n", d_tcpNumOut);
#endif
    while (sent < d_tcpNumOut) {
        ret = send(d_tcpSocket, &d_tcpOutbuf[sent], d_tcpNumOut - sent, 0);
#ifdef VERBOSE
        printf("TCP Sent %d bytes\n",ret);
#endif
        if (ret == -1) {
            fprintf(stderr, "vrpn_Endpoint::send_pending_reports: "
                "TCP send failed.\n");
            status = BROKEN;
            return -1;
        }
        sent += ret;
    }
}
```

**Figure 10:** code section where connection is failing

## **EEG SIGNAL ACQUISITION AND PROCESSING**

A certain paper was given to me by my professor called "EEG-Based Brain-Controlled Mobile Robots: A Survey". This paper informed me of the different ways to handle brain signals (i.e. P300, SSVEP, ERD/ERS). One will find in this paper the benefits and drawbacks of each method. When deciding what approach to use for controlling a drone, I chose to go with SSVEP (Steady State Visual Evoked Potential). The idea is to have an application displayed on a monitor, with multiple figures blinking at different frequencies. Each frequency would correspond to specific direction. When a user looks a specific blinking figure, the user's brain will essentially synchronize with the frequency being focused on. Multiple videos can be found online pertaining to this method, but I found one in particular to be the most helpful. Three electrodes will need to be placed on the occipital region of the head (i.e. O1, O2, and Oz of the 10-20 system). I wasn't able to figure out exactly how data obtained from the three channels should be handled, but one will find below the different steps typically used in processing signal data. Note that this can also be found in the paper referred to earlier on page 6.

### **Signal Acquisition**

Signal acquisition is done by numerous micro-controller boards to obtain necessary data. In my application, the cyton board extracted the signals from my head via electrodes attached.

### **Signal Processing**

Signal processing is the step where noise is filtered out. This can be done through the designer application in OpenVibe once data has been obtained from the acquisition server. Note that the cyton board has a "BIAS" channel that handles noise filtering. This channel could possibly be used for this application.

### **Preprocessing**

Preprocessing is the step where filters are applied. My plan was to deal with frequencies in the 14-32 Hz range, so a low-pass filter would most likely be helpful in removing certain frequencies that aren't relevant.

## **Feature Extraction**

Feature extraction is simply the step where one must identify what features are present in the data being analyzed. In this case, the different features will be the different frequencies that are being analyzed. Another example of feature extraction is when a computer needs to identify if there's a certain object present in a picture. If a computer were to focus on houses, it would look for the typically features of a house (e.g. pointy roofs, doors, windows).

## **Classification**

Once all features have been identified, the last step is to classify what a specific feature correlates to and execute a specific command. For example, if a specific frequency correlates to moving a drone in a certain direction, a program would need to classify the feature(s) found in a signal for a certain duration, in this case a specific frequency, and move the drone in the appropriate direction.

## **WHAT'S LEFT**

I was unable to successfully connect my C++ program to the OpenVibe designer application. If this project were to continue, one would need to figure out why the connection was failing. This issue was explained in the External Application section, paragraph 2. My plan was to create seven buttons to use in the designer application, but doing more research I found that this might not be possible. However, recall that OpenVibe has pre-installed buttons available that correspond to specific stimulants. If more research is done to confirm that it is not possible to create additional buttons, these buttons can be used as an alternative. The button names are descriptive as well to indicate what stimulant each button looks for (e.g. OVTK\_GDF\_Eye\_Blink). A last minute thought I had in continuing my original plan was to use an analog server as opposed to a button server. Within one analog server, multiple channels/inputs can be created. The signal(s) being analyzed from the acquisition client (the data coming in from the cyton board) can be sent through seven different filtering stages where the output of each stage will just indicate the intensity of the frequency being analyzed, a similar strategy to the spaceship demo where the output of the Beta Power display box indicated the intensity of beta activity. OpenVibe will send a streamed matrix to your application and will be handled via the callback function declared for the vrpn client. A for-loop could then be used to to analyze the matrix sent by OpenVibe to determine what frequency was present the most. Again there is an OpenVibe tutorial that shows how all of

this can be set up.

## APPENDIX

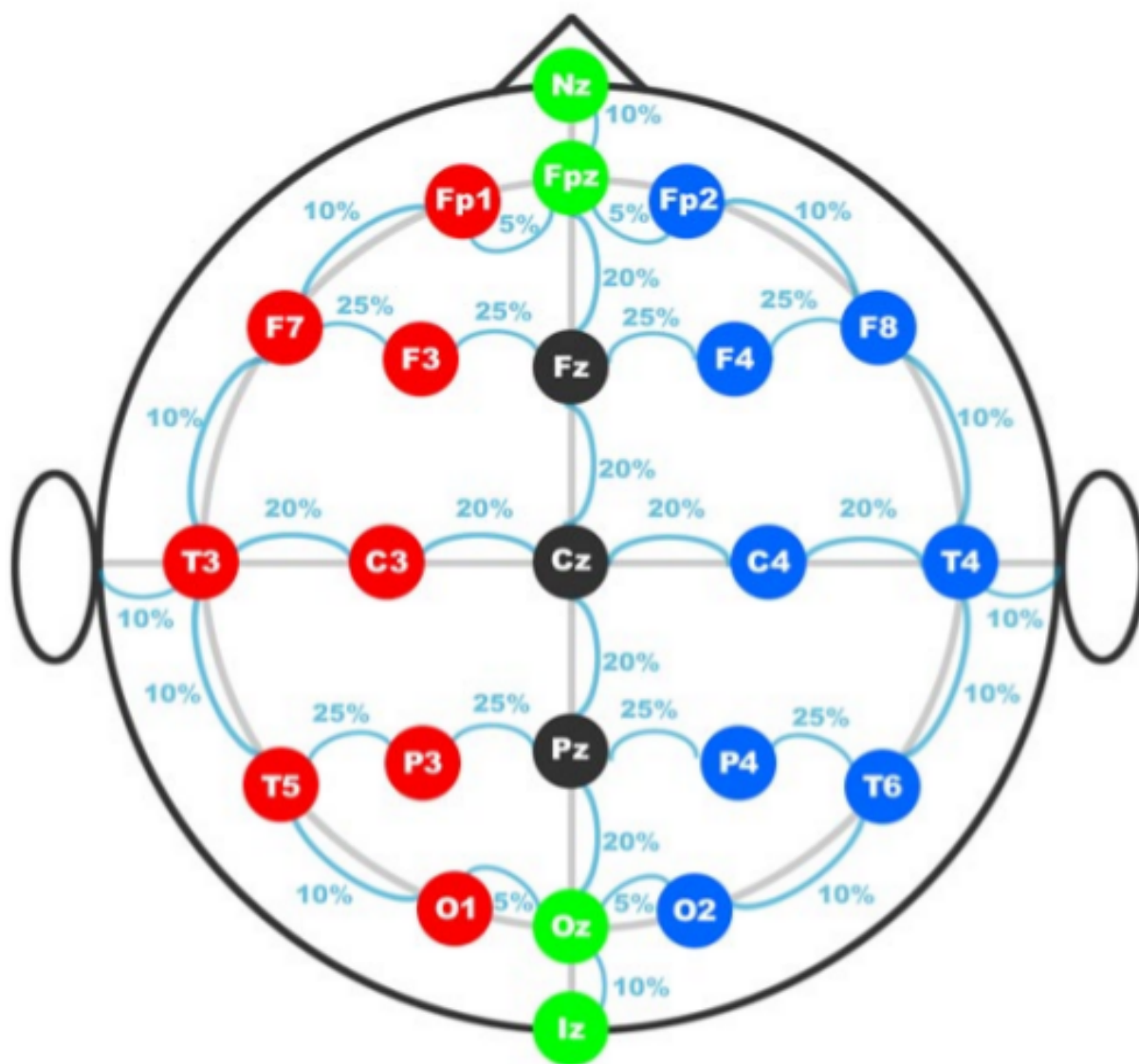


Figure 11: 10-20 EEG System

I was planning on using the following program to execute commands to a given drone. The program will run a specific application, or just read data from OpenVibe for a certain duration, and execute a certain command based off the button last pressed. Note the 'start' variable. This variable will allow the callback function to modify the 'state' variable. The seven different buttons correspond to the seven different frequencies explained in the EEG Signal Acquisition and Processing section. A certain application will run for a small duration with seven squares blinking at different frequencies. Once a specific button changes, the callback handler will be called and will determine which button was pressed. A button being pressed is just OpenVibe recognizing when a certain stimulant was found based off what each button is looking for. If this approach isn't possible, one could try using the multiple channels of the analog server which was also explained earlier.

```
/**
 * Auburn University
 * A possible approach to controlling a drone with data
 * coming in from OpenVibe
 * The idea is to connect to a VRPN Button Server in the OpenVibe designer tool
 * and execute a certain command to a drone based off a certain stimulant found.
 * The stimulant can be distinguished by determining what "button" was pressed.
 */
#include <iostream>
#include "vrpn_Button.h"
#include "vrpn_Button.C"
#include "vrpn_Analog.h"
#include "vrpn_Analog.C"
#include "vrpn_BaseClass.h"
#include "vrpn_BaseClass.C"
#include "vrpn_Connection.h"
#include "vrpn_Connection.C"
#include "vrpn_Serial.h"
#include "vrpn_Serial.C"
#include "vrpn_Shared.h"
#include "vrpn_Shared.C"
#include "vrpn_FileConnection.h"
#include "vrpn_FileConnection.C"
#include <pthread.h>
```

```
#include <semaphore.h>

using namespace std;

enum Direction{UP = 0, DOWN = 1, LEFT = 2, RIGHT = 3, FORWARD = 4,
BACKWARD = 5, STOP = 6}; /*Different directions*/
volatile enum Direction state;
uint8_t start;
uint32_t frequencies[7] = {}; /*Each index corresponding to UP, DOWN, ..., STOP, resp
uint32_t max;

/**
 *@brief Figure out which button switched its state and set to global variable: state
 Note: This handler will be called every time a button changes its state
 */
void VRPN_CALLBACK vrpn_button_callback(void* user_data, vrpn_BUTTONONCB button) {
if (!start) return; /*Used to assure the variable 'state' holds its value*/

switch (button.button) { /*Identify which button was pressed*/
case UP:
state = UP;
break;

case DOWN:
state = DOWN;
break;

case LEFT:
state = LEFT;
break;

case RIGHT:
state = RIGHT;
break;
```



```
case FORWARD:
state = FORWARD;
break;
```

```
case BACKWARD:
state = BACKWARD;
break;
```

```
default: /*Assumes button 7 was pressed. STOP*/
state = STOP;
break;
}
}
```

```
/**
```

```
 *@brief Get data from each channel and add to its appropriate array
```

```
 *Continue to set max value and update state.
```

```
 *i.e. Each index in array corresponds to a specific frequency
```

```
 */
```

```
void VRPN_CALLBACK vrpn_analog_callback(void* user_data, vrpn_ANALOGCB analog) {
    for (int i = 0; i < analog.num_channel; i++) {
        frequencies[i] += analog.channel[i]; /*Once data is received ADD data to appr
        if (frequencies[i] > max){ /*Check if new data is greater than max value.*/
            state = i;
            max = frequencies[i];
        }
    }
}
```

```
int main() {
start = 0;
max = 0;
```

```
/*Connect to either a button or analog server based off the approach used to identify
```

```
vrpn_Button_Remote* VRPNButton;
VRPNButton = new vrpn_Button_Remote("openvibe_vrpn_button@localhost");
VRPNButton->register_change_handler(&running, vrpn_button_callback);

vrpn_Button_Remote* VRPNAnalog;
VRPNAnalog = new vrpn_Analog_Remote("openvibe-vrpn_analog@localhost");
VRPNAnalog->register_change_handler(&running, vrpn_analog_callback);

while(1) {

    start = 1; /*Allow state variable to change*/

    /*Run external application for a certain duration*/

    start = 0; /*Prevent erroneous errors from callback function*/

    /*Stop running application*/

        memset(frequencies, 0, sizeof(someArray)); /*Reset array to 0 for next run*/

    /*Read the state variable and send signal to drone*/

    memset(frequencies, 0, sizeof(someArray)); /*Reset array to 0 for next run*/
}
return 0;
}
```