Table 1: EEG Lead Placement for OpenBCI System

Central Lobe	Parietal Lobe	Occipital Lobe	Ground and Reference
C_z	P_z	O_1	A_1
C_3	P_3	O_2	A_2
C_4	P_4		

a simple way to verify that the implementation of the alpha-wave collection and robot control is successful. The second design is a Pen-holder that has two components:

1. an inner hole to fix the pen direction while moving
2. a spring support to increase the flexibility of the writing

This design can potentially demonstrate more complex controls by brain signals. For instance, the robot can draw desired trajectories based on alpha wave detection. The third design is a Laser-pointer holder. This interesting design can help to show the desired path on target poster-board in response to varying detection of alpha wave.

2.3 Data Analysis

The first task consists of detecting alpha waves, which are waves in the 9 to 14 cycle frequency range that arise when a person is in a non-aroused or relaxed state [3]. The presence of alpha waves causes the robot to move in a particular direction, while the lack of alpha waves causes it to reverse its direction. To detect the relatively high-amplitude alpha waves, the following method is used: A Fast Fourier Transform computes the Discrete Forward Fourier of a filtered, five second sequence of data (updated each second). Specifically, we perform (1) Butterworth filtering and (2) Alpha wave extraction by detecting a 10 Hz signal amplitude in the computed FFT, as shown in Figure 5 in the Results section. We used the `Eigen/FFT` interface to perform FFT and IFFT operations on the signals for filter convolution within C++, and we gathered the Butterworth filter magnitudes using Python's `scipy` module which we saved into a file (`butterworth.txt`). Simply put, defining the butterworth frequency response as b and our raw data buffer signal as s, we perform the convolution $s_f = b * s$, where s_f represents the filtered signal. The next task consists of moving the robot along a single dimension using mental signals that result from specific motions (such as moving the right hand, then the left hand) or by using the accelerometer provided on the OpenBCI unit.
We started by collecting data to enable right-left robot motion. Specifically, we collect data resulting from repeatedly clenching and un-clenching the right hand for two minutes, as well as tapping a hard surface for two other minutes. We then repeat the data collection for the same motions using the left hand. To classify the data, as well as to determine which motion would perform better in moving the robot, we separate the data into a training and test set, and classify them using an SVM implemented in Python using the `scikit` [4] library. The classifier results are shown in the Results section.

2.4 Overall Program Structure

We run three threads simultaneously in order to implement the robot control: the serial thread (functions inside `OpenBCIBoard` and `EEGWAMBot` classes), the robot control thread (functions inside `EEGWAMBot` class), and the graphics thread (GLUT library). The overall structure of the classes is shown in Figure 2.