BRAIN-CONTROLLED INTERFACING AND ITS USAGE IN THE MOVEMENT OF ROBOTIC ARMS

A Thesis By

BRENDON MAUS
ORCID iD:

California State University, Fullerton
Spring, 2025

In partial fulfillment of the degree:
Master of Science in Computer Engineering

Department:
Department of Electrical and Computer Engineering

Approval Committee:
Kiran George, Ph.D., Department of Electrical and Computer Engineering, Committee Chair
Rakeshkumar Mahto, Ph.D., Department of Electrical and Computer Engineering
Kenneth John Faller II, Ph.D., Department of Electrical and Computer Engineering
Pradeep Nair, Ph.D., Department of Electrical and Computer Engineering

DOI:

Keywords:
Brain-controlled Interface (BCI), Artificial Intelligence, Signal Processing, Neural Network, Supervised Learning, Robotic Control

Abstract:
Brain-Computer Interfaces (BCIs) enable direct communication between the brain and machines, offering new possibilities for human-robot interaction. This study explores the use of electroencephalography (EEG) to detect brain activity and translate it into commands for robotic control. The primary objective is to identify specific brainwave patterns associated with intentional movement and assess their reliability in operating a robotic arm. Experiments were conducted with participants performing predefined thought processes while EEG data was collected. Advanced signal processing techniques and machine learning algorithms were applied to improve the accuracy of classifying brain signals, addressing challenges such as signal noise and individual variability. Results indicate a promising level of accuracy in converting thought patterns into robotic actions, demonstrating the potential of BCIs in assistive technologies and industrial automation. Beyond technical feasibility, this study discusses ethical considerations, societal impacts, and future improvements necessary for real-world applications. By providing a framework for thought-driven robotics, this research contributes to the advancement of intuitive and efficient human-machine collaboration, highlighting the need for continued innovation and responsible development.

TABLE OF CONTENTS

BACKGROUND..……………………………………………………………………………...1

OVERVIEW…………………………………………………………………………………3

HARDWARE SETUP………………………………………………………………………7

EEG Setup……………………………………………………………………………...7

Robotic Arm Setup……………………………………………………………………..8

SOFTWARE SETUP……………………………………………………………………...9

Software Selection……………………………………………………………………...9

Data collection………………………………………………………………………...12

Algorithm Selection Process…………………………………………………………..16

RESULTS……………………………………………………………………………….…...24

DISCUSSION………………………………………………………………………………38

Challenges and Limitations…………………………………………………………....38

Potential Optimizations………………………………………………………………..43

ACKNOWLEDGEMENTS………………………………………………………………...47

REFERENCES……………………………………………………………………………..48

Background

The foundations of Brain-Computer Interfacing (BCI) can be traced back to early research in electrophysiology and neural signal processing [1], [2]. The first observations of electrical activity in the brain were made in the 19th century, when Richard Caton recorded cortical electrical potentials in animals. These discoveries were later expanded upon by Hans Berger in the 1920s, who introduced electroencephalography (EEG) as a method for non-invasively measuring brain activity [9]. Through Berger's work, it was demonstrated that distinct patterns of electrical oscillations could be observed, leading to further investigations into the potential applications of brain signal interpretation. His identification of alpha waves, later referred to as "Berger's waves," provided an early indication that neural activity could be categorized and analyzed systematically. This research established a foundation for the later development of methods capable of detecting, classifying, and interpreting neural signals for practical applications.

During the mid-20th century, advancements in computational technologies facilitated the development of methods for analyzing neural activity with greater precision. The emergence of digital computers allowed for the processing of complex physiological signals, enabling researchers to identify patterns within EEG data. By the 1970s, it had been shown that brain signals could be used to control external devices, marking the earliest demonstrations of direct brain-to-machine communication [1]. The term "Brain-Computer Interface" was first introduced in the 1970s by researchers at the University of California, Los Angeles (UCLA), where early experimental systems were developed to explore real-time interaction between the human brain and external systems. Initial BCI systems focused primarily on communication and control for individuals with severe motor impairments, as researchers sought ways to restore lost functions through direct neural interaction. These studies laid the groundwork for subsequent investigations into BCI-based assistive technologies.

Throughout the late 20th and early 21st centuries, significant progress was made in both invasive and non-invasive BCI technologies. The introduction of more sophisticated machine learning algorithms and signal processing techniques enabled the accurate classification of neural signals, allowing for improvements in communication, mobility assistance, and neurorehabilitation applications. Early invasive BCI studies were conducted using microelectrode arrays implanted in the motor cortex, providing direct access to neural firing patterns. One of the most notable advancements was demonstrated in the early 2000s when researchers successfully enabled individuals with quadriplegia to control robotic arms through thought alone. Concurrently, non-invasive approaches using EEG and functional near-infrared spectroscopy (fNIRS) were refined, expanding BCI applications beyond medical rehabilitation into fields such as gaming, cognitive enhancement, and neurofeedback therapy. As interest in BCI research expanded, collaborations between neuroscientists, engineers, and medical professionals contributed to the refinement of existing methodologies, further solidifying the role of BCI as a transformative technology in human-computer interaction. The continuous miniaturization of sensors, improvements in wireless communication, and integration with artificial intelligence have further accelerated the development of more efficient and user-friendly BCI systems. Research into hybrid BCI models, which combine multiple signal acquisition techniques, has demonstrated increased classification accuracy and enhanced system robustness. Additionally, efforts to decode more complex cognitive functions, including language processing and emotional states, have broadened the scope of BCI research.

Despite these advancements, challenges related to signal reliability, real-time processing, and long-term usability remain under active investigation. Issues such as signal interference, individual variability in neural activity, and the need for extensive calibration procedures have limited widespread adoption. Ethical considerations, including concerns regarding user privacy, autonomy, and potential misuse of neurotechnological data, have also been increasingly discussed. Nevertheless, ongoing research continues to refine BCI systems, with future developments expected to enhance accessibility, improve classification accuracy, and expand applications beyond assistive technology into domains such as human augmentation, neuroadaptive computing, and direct brain-to-brain communication.

Overview:

The development of this brain-computer interface (BCI) project followed a structured approach, integrating hardware, software, and data processing methodologies to achieve real-time control over a robotic arm using EEG signals [3], [4], [5], [6], [10], [11], [12]. The process began with the selection of appropriate hardware, including a specialized EEG headset to capture brain activity and a robotic arm to execute corresponding movements [17], [18]. The EEG headset was strategically designed to collect brainwave data from key regions associated with motor function, ensuring the system could detect subtle neural signals linked to movement intentions. Additionally, the robotic arm was selected based on its ability to perform precise and repeatable movements, making it an ideal platform for translating classified EEG signals into real-world actions. These two components formed the foundation of the system, with each playing a crucial role in the overall functionality. The selection process involved evaluating various settings on an EEG headset for signal quality, electrode placement, and comfort, while the robotic arm was assessed for motor precision, degrees of freedom, and ease of programmability. The integration of these hardware components required extensive testing to confirm compatibility, signal reliability, and responsiveness, forming the basis for the subsequent stages of the project.

Once the hardware was established, the focus shifted to software development and integration. The EEG headset's data was initially accessed through MATLAB’s Simulink environment, allowing for real-time signal acquisition and preprocessing. This step was essential, as raw EEG signals contain noise and artifacts that can interfere with accurate classification. By leveraging Simulink, various preprocessing techniques such as filtering, normalization, and artifact rejection were applied to enhance the quality of the recorded signals. At the same time, work on the robotic arm’s control system progressed, with initial tests conducted using predefined movement commands. These tests ensured that each servo motor responded correctly to software commands and operated within its designed torque limits. As development continued, the control system evolved from a basic interface into a more sophisticated framework that allowed dynamic adjustments based on incoming signals. This transition required restructuring the codebase to enable real-time control, implementing direct servo manipulations via Python, and establishing a robust communication link between the EEG processing system and the robotic arm. This was done by utilizing a communication called UDP communication, or User Datagram Protocol. Through multiple iterations, these improvements ensured seamless data transmission and execution of movement commands, paving the way for accurate and responsive robotic control. Debugging and optimizing software routines were essential at this stage to minimize latency, prevent erroneous movements, and fine-tune the synchronization between EEG signal interpretation and motor actuation.

Data collection and machine learning model development played a crucial role in translating EEG signals into meaningful commands. To achieve this, a structured data collection process was established, where the user observed and imagined specific robotic arm movements while EEG data was recorded. This process ensured that the model would learn to differentiate between different movement intentions effectively. Feature extraction was then performed to enhance signal clarity, as EEG data contains a complex mixture of neural activity, environmental noise, and irrelevant patterns. By isolating key features such as spectral power, temporal variations, and connectivity measures, the system was able to extract meaningful patterns from the EEG recordings. Various machine learning models were evaluated, including support vector machines, neural networks, and decision trees, to determine the most effective classification approach. After extensive testing, a TreeBagger model, a type of decision tree model, was selected due to its superior accuracy and consistency across different time segmentations. The model underwent hyperparameter tuning to optimize its performance, including adjustments to tree depth, leaf count, and bootstrap sample sizes. Additionally, feature selection techniques were employed to reduce computational complexity while maintaining high classification accuracy. This phase required significant experimentation, as balancing model performance and real-time processing efficiency was a key challenge. Through iterative refinements, the classification accuracy improved, enabling reliable translation of EEG signals into robotic arm commands. Extensive validation was conducted by testing the model across various trials, adjusting parameters based on misclassifications, and refining the dataset to improve generalization and robustness.

The final phase of the project involved real-time implementation and iterative refinement. A UDP-based communication protocol was established to transmit processed EEG data from MATLAB to Python, ensuring efficient data transfer with minimal latency. This step was critical for enabling real-time interaction, as any delay in communication could negatively impact the user experience. The trained machine learning model received the processed EEG data and generated a predicted movement class, which was then mapped to predefined robotic arm movements. To ensure smooth transitions between movements, a sliding window technique was employed, where predictions were made at overlapping intervals. This technique helped balance real-time responsiveness with prediction stability, reducing abrupt or unintended movements caused by classification noise. Further refinements included implementing safeguards to prevent excessive jitter, optimizing the signal filtering pipeline, and adjusting response timing based on user feedback. These improvements enhanced the system's overall performance, making it more reliable and intuitive to use. Continuous testing was conducted to evaluate the system's real-world performance, leading to additional adjustments in feature extraction parameters, model thresholds, and control logic. The integration of these refinements resulted in a highly responsive BCI system capable of accurately interpreting user intentions and translating them into robotic actions with minimal delay. The iterative nature of testing, combined with dynamic feedback loops during training, allowed the system to progressively improve its response accuracy and adaptability under different operating conditions.

Through continuous testing and fine-tuning, the system evolved into a functional real-time BCI capable of interpreting user intentions and translating them into robotic movements. This iterative process not only enhanced the accuracy and efficiency of the system but also provided insights into optimizing real-time EEG classification for practical applications. The development journey involved overcoming various challenges, including reducing classification errors, improving signal processing techniques, and ensuring robust communication between system components. Each phase of the project contributed to refining the system's capabilities, from initial hardware selection to final implementation. The final result was a fully integrated platform demonstrating the potential of BCI technology in human-machine interaction. The system's ability to provide real-time control over a robotic arm highlights the feasibility of non-invasive brain-controlled interfaces for assistive and rehabilitative applications. Moving forward, further research and development can explore enhancements such as adaptive learning algorithms, improved user calibration techniques, and integration with more advanced robotic systems. The successful completion of this project marks a significant step toward making BCI-driven robotic control more accessible, efficient, and practical for real-world use. Future developments may also involve multi-user adaptation, improved error correction mechanisms, and extended use-case scenarios, further broadening the impact and applicability of this technology in various fields.

Hardware Setup

EEG Setup

The system architecture comprised two primary hardware components: an EEG headset for capturing the user's brain signals and a robotic arm for executing the corresponding movements. For neural signal acquisition, the project utilized the g.Nautilus Sahara EEG headset, manufactured by g.tec, Austria. This wireless headset includes 8 active electrodes positioned on the scalp and two reference electrodes attached to the user's neck. The electrode placement targets brain regions associated with motor imagery, such as the premotor cortex, supplementary motor area, and cerebellum, to maximize the capture of movement-related neural activity. The configuration adheres to the international 10–20 system, which spaces electrodes at 10–20% intervals across the skull, ensuring standardized and comprehensive coverage of relevant brain signals.

The headset transmits the EEG data wirelessly to a receiver unit connected to a computer. This wireless link allows real-time streaming of brainwave data into the software environment for processing. The captured signals are thus immediately available for filtering, feature extraction, and classification by the system's software components.

Robotic Arm Setup

For the robotic manipulator, the project employs a HiWonder xArm, a 6-degree-of-freedom educational robotic arm kit. This arm is equipped with six bus servos, each capable of approximately 25 kg·cm of torque at stall, providing sufficient strength for the required motions. The xArm offers a maximum reach of 426 mm and a gripper that opens up to 55 mm, with five actuated joints plus a gripper mechanism. These specifications ensure the arm can perform a range of movements and grasp objects as needed for demonstration purposes.

The robot arm weighs about 1 kg and is mounted on a base measuring 155 × 277 mm. It is powered by a 7.5 V, 6 A DC supply that drives an onboard ESP32-based control board. To maintain stability during operation, the base is outfitted with four suction pads (one at each corner) that secure the arm to the working surface. The xArm unit was provided pre-assembled, so integration efforts were concentrated in the development of the software control interface for its servo motors.

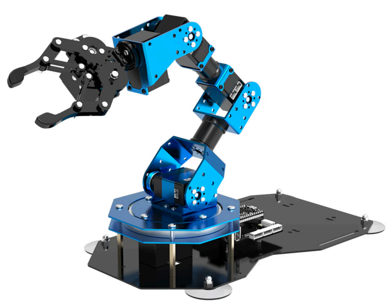


Image of the HiWonder xArm

Software control enabled direct manipulation of each servo's angle and speed, allowing fine-tuning of complex arm movements. In this project, a set of 15 discrete movement classes was defined to encompass all intended arm actions under EEG control. Specifically, each of the six servos can be commanded to rotate either in a positive or negative direction, accounting for 12 basic movement classes, with two directions for each servo. Two additional classes were reserved for speed modulation commands, and a final class represented a complete stop command. This taxonomy of 15 classes covers the full range of movements and states for the robotic arm, forming the basis for translating brain signals into arm control commands.

Software Setup

Software Selection

In parallel with the hardware, a variety of software tools were utilized to ensure proper signal processing, communication, and control throughout the system. The development of the software pipeline was highly iterative: many stages were revised multiple times before arriving at the final implementation. Initially, the project made use of MATLAB for both EEG data acquisition and for interfacing with the xArm's provided control software [13]. Early prototypes involved collecting EEG signals via MATLAB/Simulink and controlling the robotic arm using the manufacturer's GUI-based interface. This approach was chosen because the EEG headset's data could only be accessed through the g.tec Simulink interface, and at the time the robotic arm's control was most readily achieved through its developmental software.

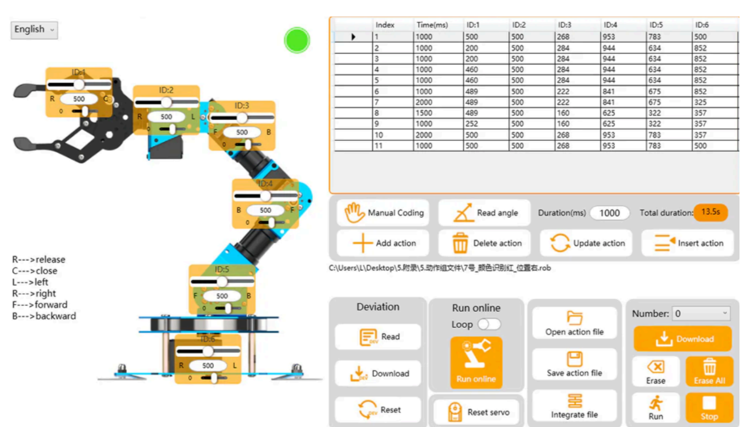


Image of the HiWonder xArm software for programmable and manual control

As the project progressed, a more direct control method for the robotic arm was identified. It was discovered that the xArm could be controlled programmatically in Python by bypassing the vendor's GUI interface. This discovery significantly improved the system's flexibility: direct Python control offered finer command over each joint and made it possible to send custom control signals, such as those derived from EEG classifications, to the arm's motors. The primary constraint remained the EEG data acquisition, which was locked into MATLAB/Simulink. This mismatch meant that, initially, the EEG processing and robotic control subsystems could not easily communicate in real time. The original plan was to develop the data acquisition and robotic arm control components independently and integrate them at the end, hoping the parts would interface seamlessly. However, this approach carried risk, and a better integration strategy was eventually implemented, as discussed later.

During development, an interim solution was used to test the robotic arm control logic in the absence of a live EEG-driven signal. The arm control code was structured around a set of 15 conditional (if) statements corresponding to the 15 movement classes defined for the servos. Each if statement checked for a specific input condition and, if triggered, executed the associated movement, such as moving a particular servo in a given direction, adjusting speed, or stopping the arm. Because the EEG-to-arm communication was not yet established at that stage, keyboard inputs were used to simulate the EEG classification outputs. In other words, key presses were mapped to each of the 15 movement classes, allowing the ability to manually trigger each type of arm movement for movement verification. This approach ensured that the software handling of all motions was correct and responsive before introducing actual EEG control.

The control logic was carefully designed to avoid erratic or continuous commands that could overwhelm the servos. For example, when a key corresponding to a movement was pressed, the system would register the new command and lock it in until it was released or changed, rather than repeatedly sending the same command in rapid succession. Without this locking mechanism, holding down a key, or eventually sustaining a particular EEG output, could result in an excessive stream of commands that might cause the motors to stall or behave unpredictably. By only registering state changes, or transitions between different commands, and ignoring repeats, the software prevented unintended rapid firing of identical commands. This design translated well to the eventual EEG-driven mode, since the classification output at any given moment could be treated similarly to a “pressed key” indicating the current intended motion. Additional provisions were made for more nuanced control once the basic command structure was validated. Initially, the concept was to let the duration of the user’s thought determine how long the arm moved. For instance, thinking longer about a movement would cause a longer movement. In practice, reliably using thought duration as a control signal proved too challenging. Instead, a manual “stop” command was implemented as one of the movement classes, allowing the user to halt the arm’s movement at will. The user also had two movement classes dedicated to speed control, such as a “faster” and “slower” mental command, which enabled fine adjustments to how far and how quickly the arm moved when combined with the stop functionality.

To synchronize the system’s operation, a fixed update interval was introduced. The robot control program checks for new incoming commands on a 0.5-second cycle, or every 500 ms. At each interval, the system processes the latest predicted class from the EEG interface. If the predicted class has changed since the last interval, the software executes the corresponding servo movement, speed adjustment, or stoppage. If the predicted class remains the same as in the previous interval, no new action is taken, as that movement is already in progress. This half-second cycling of command updates serves two purposes: it smooths out rapid fluctuations in the predictions and prevents the arm from jittering with every minor change in EEG output, and it ensures timely execution of genuine changes in the user’s intent. The 0.5 s interval was chosen empirically as a balance between responsiveness and stability — it is short enough to feel reactive, but long enough to filter out very brief spurious changes in the signal.

Data Collection

With the hardware and basic control software in place, the next phase involved collecting EEG data to train the machine learning model that would map brain signals to robotic commands. Data collection was structured around the 15 defined movement classes for the robotic arm. For each class, the user performed a series of trials consisting of two stages: an observation stage and an imagination stage. In the observation stage, the user would watch the robotic arm execute the specific movement associated with that class (this could be done by manually controlling the arm or playing back a recorded movement). Immediately after, in the imagination stage, the user would close their eyes, remain as still as possible, and vividly imagine performing the same movement with the robotic arm. Each imagination trial lasted approximately 20 seconds. This duration was an arbitrary choice intended to provide a substantial amount of EEG data per trial while avoiding excessive fatigue or loss of concentration on the part of the user. Between trials, the user rested to prevent mental exhaustion. Ten such trials were conducted for each of the 15 classes, yielding a total of 150 EEG recordings (trials) for the training dataset.

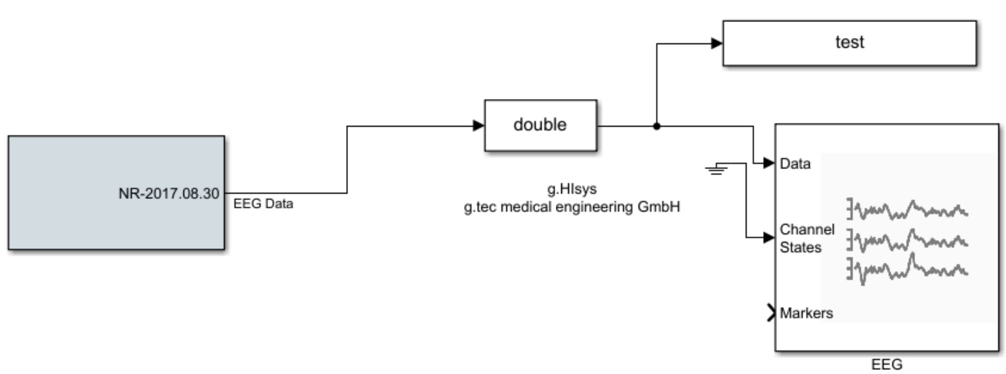


Image of the Simulink sandbox for recording each trial

With the collection of data came the way the data was collected. Because of the constraint with using the g.tec software, the settings for the headset were built-in to the Simulink sandbox environment. As such, the main variables that could be modified were the frequency, input range, bipolar channel, the bandpass filter, the notch filter, the common average referencing (CAR), and the noise reduction. Apart from the frequency, each of these variables could be independently modified for each of the 8 channels for the headset. In the case of this project, all 8 nodes were provided the same settings. These were the settings in question: The input range for each channel was set to be $\pm$ 2250 millivolts. This was the largest possible range for the headset, and helped ensure that all of the raw EEG data would properly be captured without being clipped. One of the other settings that could be modified was the bandpass range. The bandpass range was from 5 - 60 hertz, and the notch was set to 58 - 62 hertz. These two settings were used to modify the frequency ranges for the values. The temporal data was able to be converted into a spectral range using a fast fourier transform. Because of this, the headset was able to measure bandpower ranges, and the main data was in the range of 5 hertz to 60 hertz.

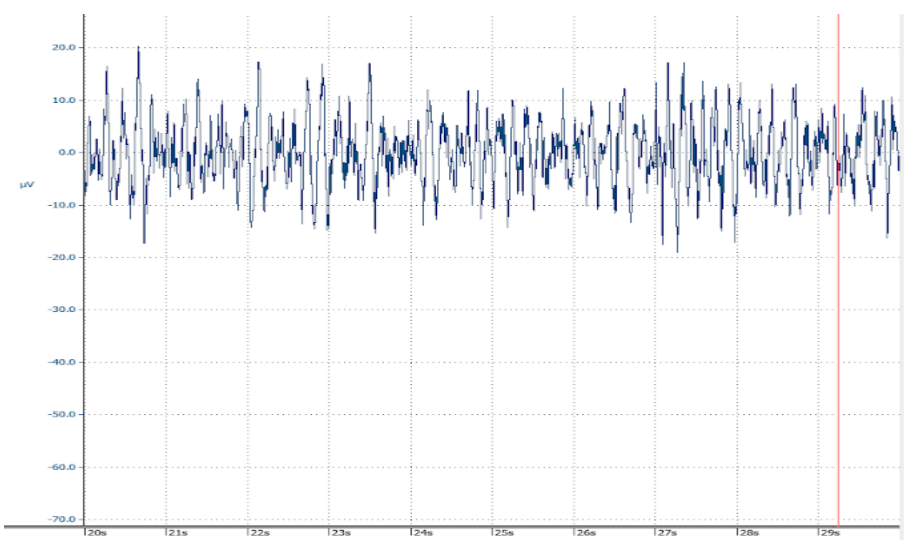


Image of the collected temporal data from one channel on the EEG headset

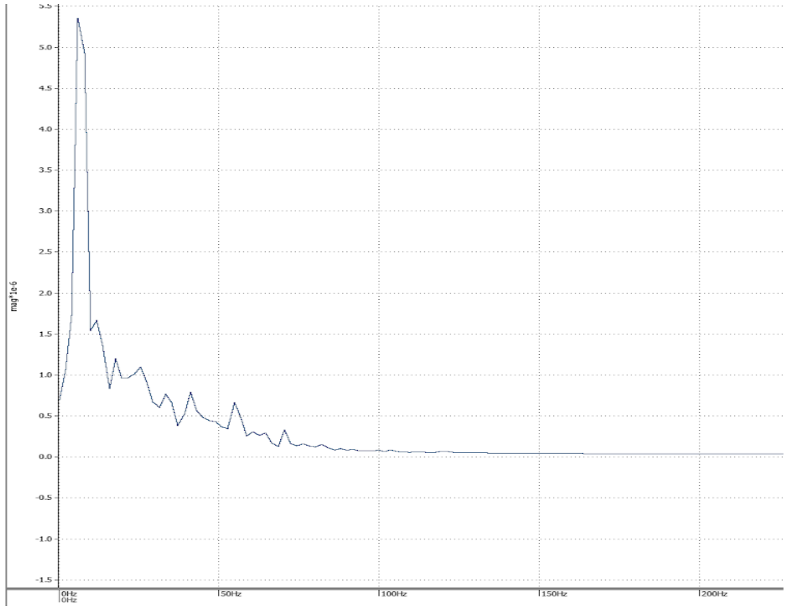


Image of the collected spectral data from one channel on the EEG headset

The notch filter ensured that the frequency values in or past that range were dropped, ensuring that any inputs to the system were only seen between the bandpass ranges themselves. There was also no Common Average Referencing, or CAR, for any of the channels. The common average referencing was a parameter that would take the average value from all channels, and subtract all values from this average value. This technique is beneficial for removing noise and artifacts within the data, but was determined to be effective only in offline analysis. Since the focus of this project was to utilize a real-time system, these constant calculations would add an inherent delay to the system, harming the responsiveness of the arm with respect to the user. Because of this, and the fact that other techniques can be used to reduce noise that do not harm the computational delay, the CAR parameter was turned off. The sampling rate was set to 500 hertz, which allowed for more data points to measure from. Decreasing the frequency would have reduced the number of data points, which may not help in capturing those critical features that help with classification, and 500 hertz was the highest frequency value that could be selected. Because of this reasoning, the 500 hertz operating frequency was chosen. One final setting that could be modified before any data recording was the operating mode for each node. In this case, each node worked in a monopolar manner. A monopole node meant that all of the values recorded were averaged to the reference node. For the headset, the reference node was the node attached to the neck, and the ground node was the other node attached to the neck. These two nodes helped to average out the values to the body's natural electrical signals, and were able to operate in a similar fashion to the CAR parameter.

Channel	Acquire	Input range	Bipolar channel	Bandpass filter	Notch filter	CAR	Noise reduction
CZ	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
2	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
3	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
4	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
5	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
6	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
7	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No
8	☑	2250 mV	none	5 Hz - 60 Hz	58 Hz - 62 Hz	No	No

Image of the settings chosen for the g.Nautilus Sahara (8 channel)

All recorded EEG trials were then segmented into shorter time windows to prepare them for input into various classification algorithms. Specifically, each 20-second trial was chopped into segments of fixed lengths: 10-second segments, 5-second segments, 4-second segments, 2-second segments, and 1-second segments were created. This segmentation produced multiple datasets of different temporal resolution. The rationale for this step was to investigate the trade-off between classification accuracy and response time [10], [11], [16]. Longer segments contain more brainwave information and potentially lead to higher classification accuracy because the signal averages out noise over a longer period, but they also meant that the system would respond more slowly to user inputs. Conversely, very short segments would enable faster response but may also be less accurate due to limited data per prediction. By evaluating a range of segment lengths, a window size that offered a good balance for real-time control could be determined.

Once the data was segmented, it was fed into MATLAB's Classification Learner application to train and evaluate a variety of machine learning models. Prior to training, each dataset was partitioned into training, validation, and testing subsets in a stratified manner. Typically, 70% of the data from each class was used for training and 30% was reserved for testing; additionally, a portion of the training data was set aside for validation during the training process. This careful splitting ensured that model performance could be assessed on data it had not seen during training, providing a more realistic evaluation of how the model might perform on new EEG signals.

Algorithm Selection Process

A total of 23 different classification algorithms, including spanning decision tree ensembles, support vector machines, neural networks, discriminant analysis, and others, were initially applied to the EEG data to identify which was most effective at distinguishing the 15 classes. The performance of each model was recorded, with particular attention to how accuracy changed as the segment length varied. In these trials, a type of random forest model, MATLAB's TreeBagger ensemble, emerged as the top performer. For the longest recorded window, 20 s segments, the TreeBagger model achieved a 100% classification accuracy, as did one of the Support Vector Machine models, indicating that with sufficiently large time windows the EEG patterns for each class were highly distinguishable. However, for shorter segment lengths, the TreeBagger consistently outperformed other algorithms. For example, using 10-second segments, the TreeBagger model reached about 91% accuracy – the highest among all models tested – whereas the next best algorithms scored under 70%. This trend continued with decreasing window sizes: even at the 1-second segment length, which was the most challenging scenario due to minimal data per prediction, the TreeBagger model maintained the best accuracy. The TreeBagger model reached around 50% accuracy in correctly classifying 15 classes, which, although modest, was significantly better than chance level and higher than other models achieved. Given its superior performance across different time scales and its robustness to noise and variability, the TreeBagger random forest was selected as the final classification model for the system. While this test was rudimentary in terms of the data used, it provided a clear picture on the type of machine learning model that should be chosen moving forward.

After identifying the TreeBagger random forest as the most promising algorithm, the project focused on configuring and optimizing this model for real-time use. The first step was to enhance the feature extraction from the EEG data to improve the model's accuracy and efficiency. Although the EEG headset and Simulink environment applied basic preprocessing through band-pass filtering during data collection, additional features were engineered from the signals to better capture the distinguishing characteristics of each class. The EEG signal from each channel was decomposed into standard frequency bands commonly used in BCI research: delta (~5–8 Hz in this setup's definition), theta (~8–13 Hz), alpha (often called "mu" when referring to motor cortex rhythms, here ~13–30 Hz), and beta/gamma (~30–60 Hz) [19]. Using a Fast Fourier Transform (FFT), the power spectral density within each of these bands was computed for each channel, alongside temporal features extracted from the time-domain signal. In total, 86 numerical features were derived per channel, including band-specific power measures, temporal statistics, etc. With 8 EEG channels in use, this feature extraction resulted in a feature vector of 688 elements representing each 1-second window of data. This was a drastic reduction from the raw data dimensionality. As a comparison, a 1-second window sampled at 500 Hz with 8 channels contains 4,000 time-point values. By distilling the EEG into a set of relevant features, the model's training complexity was reduced and its focus directed to the most informative aspects of the signal. This feature extraction not only sped up the training process but also helped to generalize the model by filtering out high-frequency noise and other irrelevant variations that could otherwise lead to overfitting.

With the features defined, the TreeBagger model was trained and then further refined through hyperparameter tuning. Key hyperparameters for the random forest, such as the number of decision trees in the ensemble, the maximum depth of each tree, the minimum number of samples per leaf node, and the subset of features considered at each split, were optimized using Bayesian optimization. This automated process searches for hyperparameter values that minimize the model's prediction error. In each iteration of the search, a set of hyperparameters is chosen, the model is trained and evaluated, and then the hyperparameters are adjusted based on the observed performance by utilizing Bayesian inference to decide the next set of values to try. This approach efficiently navigates the hyperparameter space and can find a near-optimal configuration in a reasonable number of iterations. One caveat of this method is the possibility of converging to a local optimum – a set of hyperparameters that is the best among those tried so far, but not truly the best overall. To mitigate this, multiple runs of the optimization were performed from different starting conditions. Through this tuning process, the random forest's accuracy improved substantially over its default settings. After feature extraction and initial hyperparameter tuning, the model was able to achieve roughly 65% classification accuracy on 1-second segments in a consistent manner, measured on validation data. This was a 15% increase from the base model measured in the Classification Learner application within MATLAB. While this accuracy might appear moderate, it is notable given the challenge of 15-way classification on very short EEG segments, and it provided a baseline for further improvements.

To further boost performance, a post-processing strategy termed "dynamic thresholding" was applied to the model's output. In the context of a multi-class random forest, the model produces a set of confidence scores, or probabilities, for all 15 classes for each input segment. By default, the class with the highest confidence is chosen as the prediction. Initially, a simple majority rule was applied: if any class's confidence was above a fixed threshold, such as 0.5 or 50%, that class would be selected as the predicted command. This could be done because a technique called SoftMax was utilized. This operation ensures that the sum of the confidence values of the outputs are restricted to 1. This meant that no two values could be over the 0.5

threshold, demonstrating that the confidence values were representative of the model as a whole. However, if no class exceeded the threshold, the highest-confidence class would be selected anyway. This was done to ensure a decision is always made. However, this default threshold can be suboptimal, especially if the model tends to spread its confidence, or if certain classes are consistently predicted with lower confidence. Dynamic thresholding involves adjusting the decision thresholds for each class after the model is trained, in order to improve overall accuracy and balance the errors across classes. Using another Bayesian optimization routine, the threshold values were tuned: the optimizer iteratively tried different, customized threshold settings for each class, and measured the resulting accuracy. The outcome of dynamic thresholding was a set of refined decision criteria that improved classification reliability. While this was an improvement to the model as a whole, the issue with this was a balance between overfitting the data to the model and underfitting the data to the model. The tuning of the threshold values only aided in the data that was used for testing, and the testing dataset was only a small subset of all of the recorded data. This tuning process may have introduced bias toward minority data subsets, potentially reducing generalization across the full dataset. On the other hand, having too general of a model would not be able to capture the relevant values for the data to be accurately classified, resulting in a lower confidence and accuracy overall in the model. This balance was constantly being offset with each additional post-processing step, as the model became more and more tailored to the recorded values, and less and less tailored to the general trends of the data.

Another post-processing step that was taken after the model was built was feature reduction, which was enabled through the use of Bayesian optimization as well [12]. This process aids in reducing the overall number of features to highlight the most impactful value only. Every feature could have the importance value measured, where the higher the importance a feature was, the more discerning the feature was in the classification process. Higher importance values meant more discriminating features, and lower importance values meant less discriminating features. These lower-importance features could actually harm the accuracy of the model because too many values could be treated as noise, and reduce the effectiveness of the important features. As such, features that did not contribute meaningfully were effectively pruned by this process, because the optimization would favor threshold values that ignore those features. This step helped to remove some remaining noise and artifact influence by effectively zeroing out contributions from less informative features and requiring stronger evidence before a class is chosen. It also reduced the bias where a few dominant classes might otherwise always be selected on marginal signals. The iterative nature of this approach ensured that a near-optimal balance was found between sensitivity, picking up true intentions, and specificity, avoiding false alarms, for each class. One potential downside to this reduction is that while the importance of a feature helps with discerning classes, the amount of classes discerned is not provided. Many key important features may only be highlighting a specific class compared to the majority of classes, inflating the confidence of individual classes, and increasing the amount of times that class gets predicted. While the noise made from all of the features can dilute the overall accuracy of the model, they can also pull the extreme class values from being inflated too much to where the model suffers in generalizing the predictions.

With the model trained and optimized offline, the final stage of the methodology was integrating it into a real-time system that could take live EEG data and output robot movements continuously. This integration was achieved through a network communication protocol. The EEG data acquisition was done in Simulink by running on a PC connected to the EEG receiver, while the classification model and robot control were implemented in Python. To bridge these components, a User Datagram Protocol (UDP) connection was established. In practice, one UDP socket was configured within the Simulink model to act as a data transmitter, sending out the encoded, preprocessed EEG feature data at regular intervals in the form of bytes. A corresponding UDP socket was set up in the Python environment to receive that data in the form of bytes after decoding the values back into the double values. UDP was chosen for its simplicity and low overhead in sending small packets over the local machine's network interface. Each 1-second EEG segment, along with its computed features, was transmitted as a binary data packet to the Python-based classification program. Upon receiving a packet, the Python program decodes it and applies the exact same feature extraction pipeline that was used during training to ensure consistency, such as FFT and bandpass filtering. The resulting feature vector is then input to the pre-trained TreeBagger model, which produces a probability distribution across the 15 classes and, using the dynamic thresholds and reduced feature threshold, outputs a predicted class label.

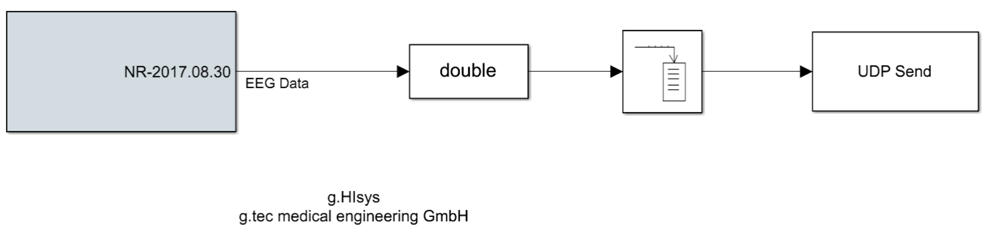


Image of the Simulink Workflow for real-time data recording

Because the UDP protocol is connectionless and transmits raw bytes, the predicted class label was converted into a suitable format for transmission to the robot controller. A second UDP connection was used for this purpose: the Python classification program acted as a UDP sender, and a separate Python process for the robot control program acted as the receiver. The predicted class was encoded as a string into a small binary message and sent to the robot control program. On reception, the robot control program decoded the message to retrieve the class label. It then executed the corresponding arm movement by activating the appropriate if condition in the control logic, which was the same logic that had been tested earlier with keyboard input. In this way, a continuous loop was established: the EEG system sends data to the classifier, the classifier sends commands to the robot, and the robot moves according to the user's inferred intent, all in near real-time.

To achieve a responsive yet stable control loop, the real time system operated on overlapping time windows of EEG data. Although the classification model was trained on 1-second segments, the implementation uses a sliding window with a 0.5-second step. Every half-second, a new classification is produced using the most recent 1.0 second of EEG data. This means consecutive predictions share 50% of their data with each other. For example, a window covering 0.0–1.0 s is used for one prediction, then 0.5–1.5 s for the next prediction. This would continue until the user manually turned the system off. Using a sliding window in this way effectively doubles the update rate of the system's outputs by providing a new command twice per second without requiring a shorter training window. The benefit of this approach is a smoother and more continuous control experience: if the user's brain signal maintains the same class for an extended time, the overlapping windows will repeatedly predict that class, resulting in sustained motion. If the user changes their thought, the system can detect the change within 0.5 s at most, rather than waiting a full second. However, there is a trade-off in terms of output latency and potential redundancy: because each data sample is analyzed twice in the two overlapping window segments, a brief artifact or spike in the EEG could influence two consecutive predictions. The overlap was carefully tuned and concluded that a 0.5 s step, or 50% overlap, was the optimal compromise between responsiveness and signal stability. This configuration yields a minor, manageable latency, as the system effectively has about half a second delay for new commands, while significantly mitigating jitter. The final result of the methodology was a fully integrated brain-controlled robotic arm system: one that preprocesses and classifies incoming EEG signals in real time and reliably translates them into the desired arm movements.

One main issue that was later discovered however, was that even though the model worked well for the currently recorded data, the model only worked with that session of the brain. One main issue with BCI implementations is the fact that the brain itself constantly evolves over time, making current predictions from the specific areas of the brain potentially useless in a static environment. This is a phenomenon called mental drift, where the thoughts of the user can alter neural patterns over time, causing different neural pathways to be developed, structurally changing the way the data gets recorded [8]. The way around this in the beginning was to record all of the sessions at the same time of day over various batches. Each class had a total of 10 files recorded, so each day was one file for each class. This was repeated over the multitude of days required, as an attempt to mitigate the drift. This helped with the way the data was collected statically, but even a session lasting more than 15 minutes was different from the start to the end. As such, one work around was to implement some dynamic calibrations every 5 minutes. These calibrations help to ensure the data stays on the same relative scale as the recorded data, removing some potential noise in the analysis stage with the standardization. Then, once the calibration was done, a dynamic threshold updating system was developed. This was a system that recorded individual actions from the user, and based on what the class should have been and the results from the prediction, the thresholds for each class would be modified to further support the individual class. This would be done for all 15 of the action classes, and afterwards, the user would be able to control the actions of the robot arm as normal. This would need to be done every 5 minutes to ensure the drift is controlled and ensures a sustained accuracy throughout the usage.

Results

In regards to the construction of the pipeline, the data was first filtered in order to remove any noise in the system. This was done through the use of a butterworth bandpass filter of order 4 within the defined bandpass filter range of 5 - 60 Hz.

	1	2	3	4	5	6	7	8	9
a	1.0	-6.0560983208873 19	16.189391776864	-25.0424187724717 77	24.5833527844497 12	-15.703133219063 606	6.37286734984521 35	-1.5012168452926 33	0.15725756202529 526
b	0.0066613367263 02997	0.0	-0.0266453469052 1199	0.0	0.0399680203578 17984	0.0	-0.0266453469052 1199	0.0	0.0066613367263 02997

After the data went through a bandpass filter, a variety of data transformers were tested to determine the best accuracy in terms of scaling the data. The four main scalers chosen were StandardScaler, MinMaxScaler, RobustScaler, and QuantileTransformer. Each one of these scalers provided a different way to scale the data after filtering, to help clamp all of the data in a more consistent manner. For example, the StandardScaler would transform the data to have a mean of 0 and a standard deviation of 1. The MinMaxScaler would ensure that the minimum and maximum values were scaled to -1 and 1, respectively. The RobustScaler would utilize the median and interquartile range (IQR) values to help eliminate outliers in the dataset. The QuantileTransformer would convert the data into a uniform distribution, reducing the impact of marginal outliers by balancing all of the values of the data. A base random forest model was built with each scaler utilized for the data, and the accuracy of the model was compared using the same cross validation techniques as mentioned for the base random forest model. The scaler with the highest accuracy led to the best transformation for the data, and this was determined dynamically. In the end, the model was best suited with the RobustScaler, with a mean cross validation score of 0.7553, an improvement to the StandardScaler and MinMaxScaler with a mean cross validation score of 0.7551, and all three performed better than the QuantileTransformer, who achieved a mean cross validation accuracy score of 0.753

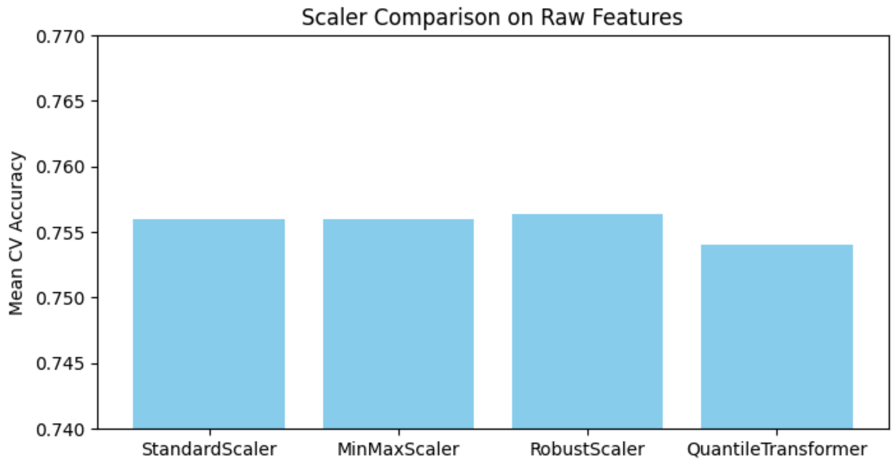
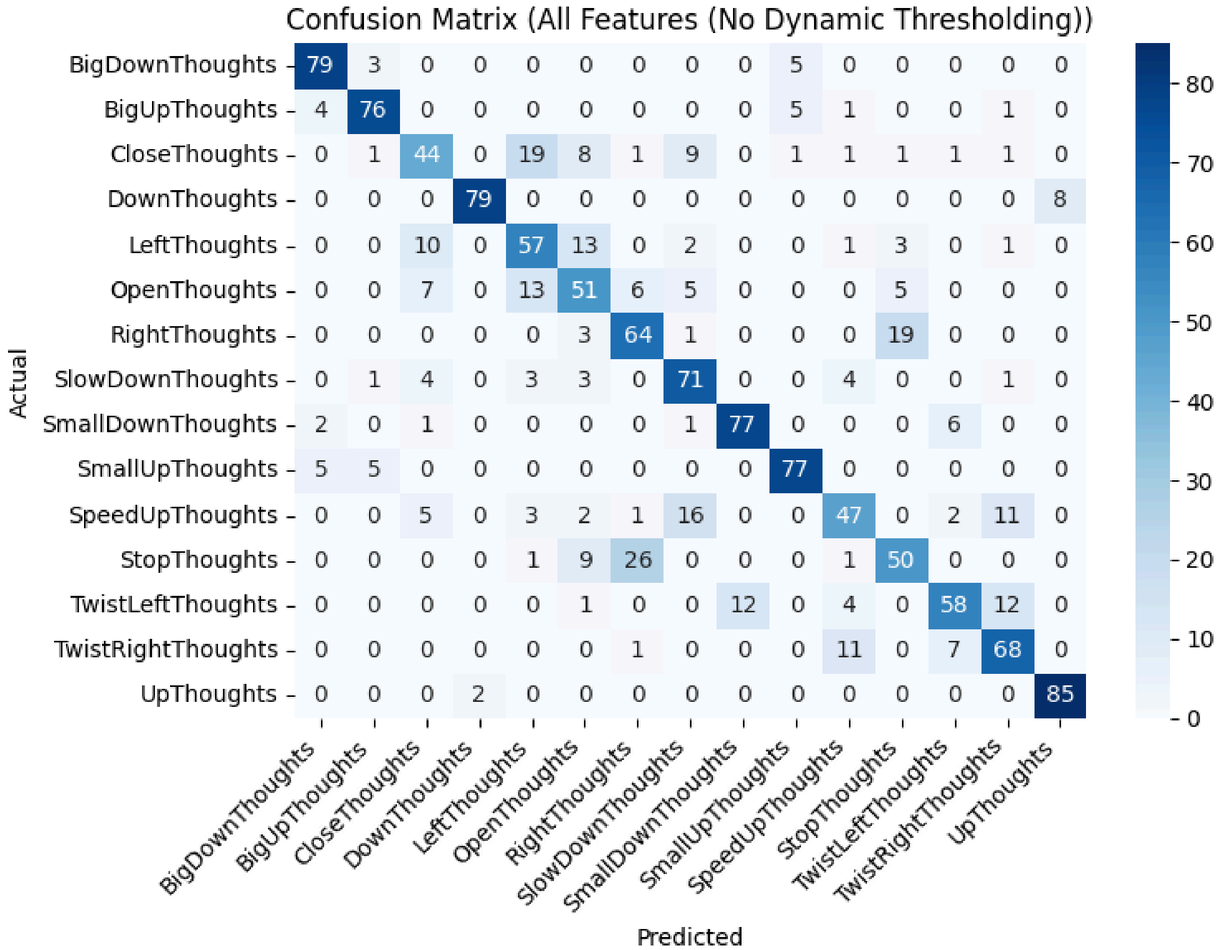


Image of the comparisons between the 4 data scaling techniques

Once the RobustScaler was chosen for the random forest model, the Bayesian optimization for the random forest model began. The random forest model underwent a total of 50 iterations on the 70% training data, and in the end, the hyperparameters that generated the best accuracy score was a max depth of 105, the maximum number of features as 0.7736687792246868, the minimum number of samples per leaf being 3, and the number of estimators in the random forest model being 1460. Upon developing the machine learning model with the determined parameters and optimizations, the random model was able to achieve a 75.33% accuracy on the recorded data. The optimization values were checkpointed at various stages of the process. With the model looking at all of the raw data for classification only, the accuracy score was 75.33%.



Confusion Matrix generated from Random Forest Model

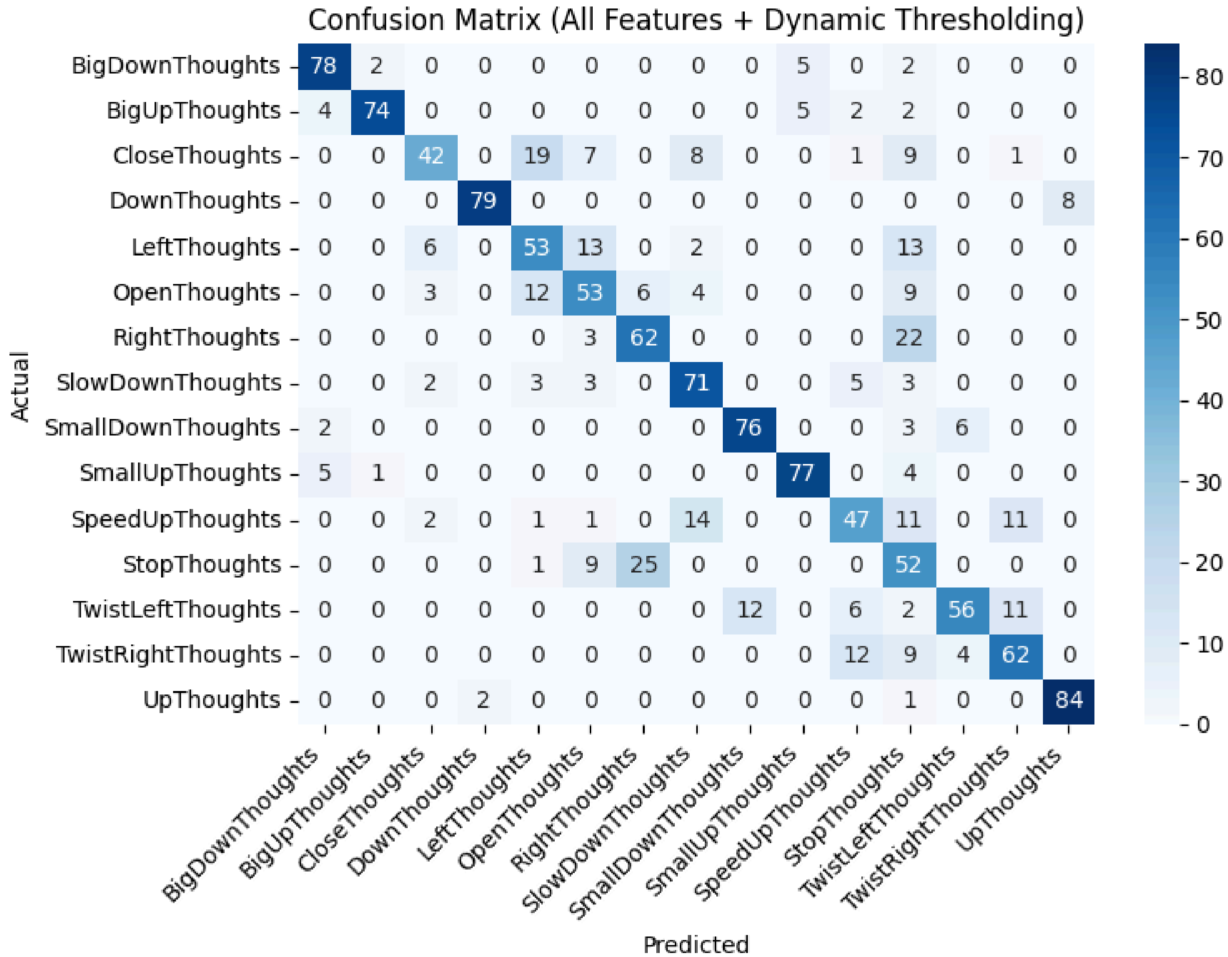
As seen by the confusion matrix however, the model was unable to predict some of the classes well at all, specifically CloseThoughts and SpeedUpThoughts. This was due to the fact that certain features from the extraction process are better classifiers for those more dominant actions. This can be seen by the increased predictions for the DownThoughts, UpThoughts, SmallDownThoughts, and SmallUpThoughts. Once the confusion matrix was generated, a classification report was also generated. In the table, the threshold values were generated, as well as the precision, recall, and harmonic mean, also known as an F1 score. The precision category demonstrates the individual class accuracy in relation to all of the files for that class. The recall category demonstrates the individual class prediction in relation to all of the files for that class.

The harmonic mean is a mathematical formula that combines the precision value and recall value to demonstrate the overall balance within the class in terms of specificity and sensitivity.

1. $TP = True\ Positive$
2. $FP = False\ Positive$
3. $FN = False\ Negative$
4. $Precision = \frac{TP}{TP + FP}$
5. $Recall = \frac{TP}{TP + FN}$
6. $Harmonic\ Mean = 2 * \frac{Precision * Recall}{Precision + Recall}$

Classification Report (All Features, No Dynamic Thresholding)					
Class	Balanced Thresholds	Precision	Recall	Harmonic Mean	Support
Big Down	0.5	0.878	0.908	0.893	87
Big Up	0.5	0.884	0.874	0.879	87
Close	0.5	0.620	0.506	0.557	87
Down	0.5	0.975	0.908	0.940	87
Left	0.5	0.594	0.655	0.623	87
Open	0.5	0.567	0.586	0.576	87
Right	0.5	0.646	0.736	0.688	87
Slow Down	0.5	0.676	0.816	0.740	87
Small Down	0.5	0.865	0.885	0.875	87
Small Up	0.5	0.875	0.885	0.880	87
Speed Up	0.5	0.671	0.540	0.599	87
Stop	0.5	0.641	0.575	0.606	87
Twist Left	0.5	0.784	0.667	0.720	87
Twist Right	0.5	0.716	0.782	0.747	87
Up	0.5	0.914	0.977	0.944	87

After obtaining the base machine learning model, the post-processing stages were used. Instead of the balanced 0.5 threshold values previously used, the dynamic thresholding modified these thresholds to increase the accuracy of these more reserved classes. This did hurt the overall accuracy, but aimed to improve the class accuracy for those that were not predicted as much, so the accuracy changed to 74.02%.

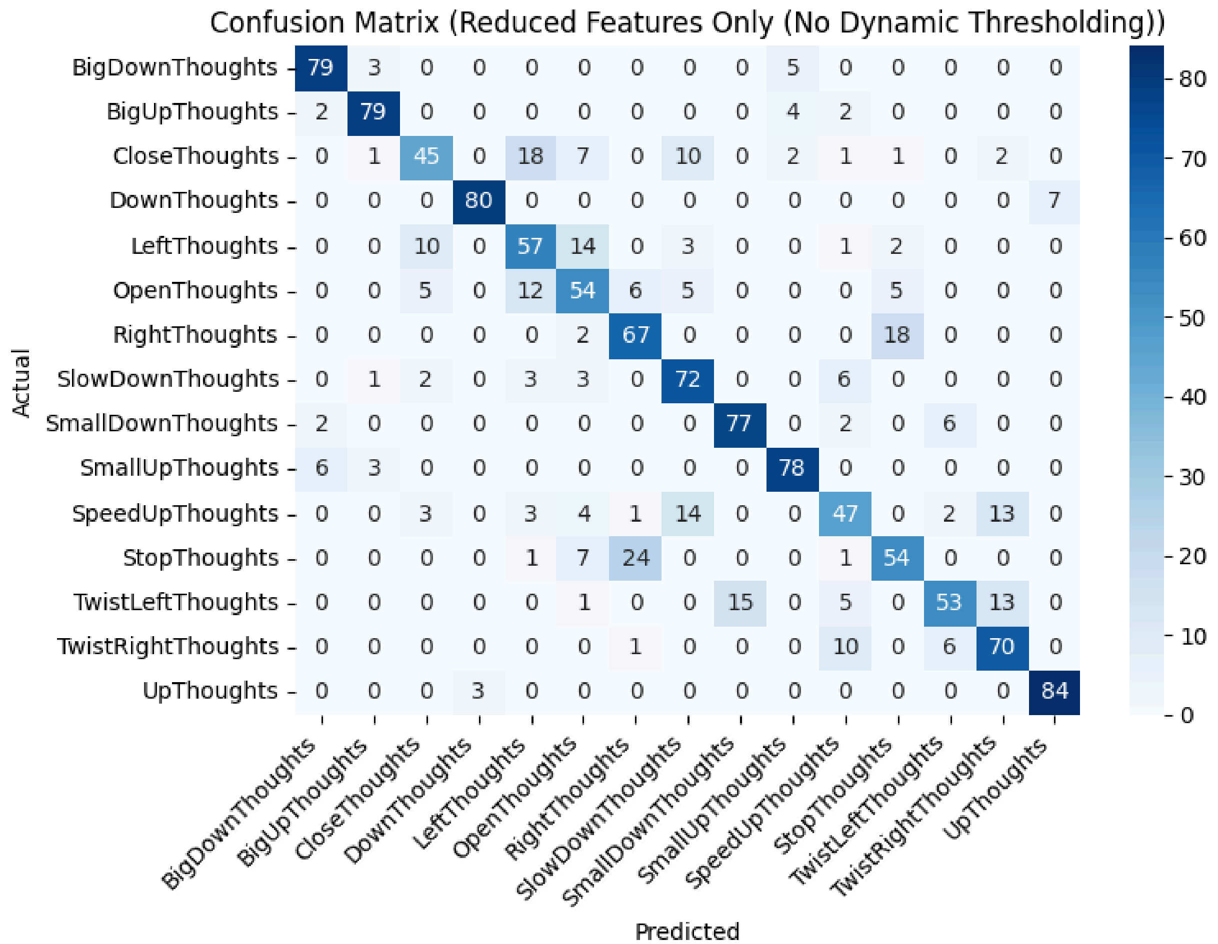


Confusion Matrix generated from Dynamic Thresholding Random Forest Model

As seen by the confusion matrix however, the model was still unable to predict some of the classes well at all, specifically CloseThoughts and SpeedUpThoughts. This was due to the fact that certain features from the extraction process are better classifiers for those more dominant actions. This can be seen by the increased predictions for the DownThoughts, UpThoughts, SmallDownThoughts, and SmallUpThoughts. Once the confusion matrix was generated, a classification report was also generated. In the table, the threshold values were generated, as well as the precision, recall, and harmonic mean, also known as an F1 score.

Classification Report (All Features, Dynamic Thresholding)					
Class	Dynamic Thresholds	Precision	Recall	Harmonic Mean	Support
Big Down	0.357	0.876	0.897	0.886	87
Big Up	0.481	0.961	0.851	0.902	87
Close	0.266	0.764	0.483	0.592	87
Down	0.285	0.975	0.908	0.940	87
Left	0.226	0.596	0.609	0.602	87
Open	0.241	0.596	0.609	0.602	87
Right	0.197	0.667	0.713	0.689	87
Slow Down	0.244	0.717	0.816	0.763	87
Small Down	0.341	0.864	0.874	0.869	87
Small Up	0.402	0.885	0.885	0.885	87
Speed Up	0.291	0.644	0.540	0.588	87
Stop	0.220	0.366	0.598	0.454	87
Twist Left	0.382	0.848	0.644	0.732	87
Twist Right	0.380	0.729	0.713	0.721	87
Up	0.423	0.913	0.966	0.939	87

Because of the large number of features, it was believed that reducing the feature size would highlight only the most important features for classification, aiding in boosting accuracy for the lesser-represented classes. With this feature reduction in place, the accuracy overall increased to 76.32%. A total of 255 features were selected out of the 688 features through the Bayesian optimization approach with the importance threshold values.

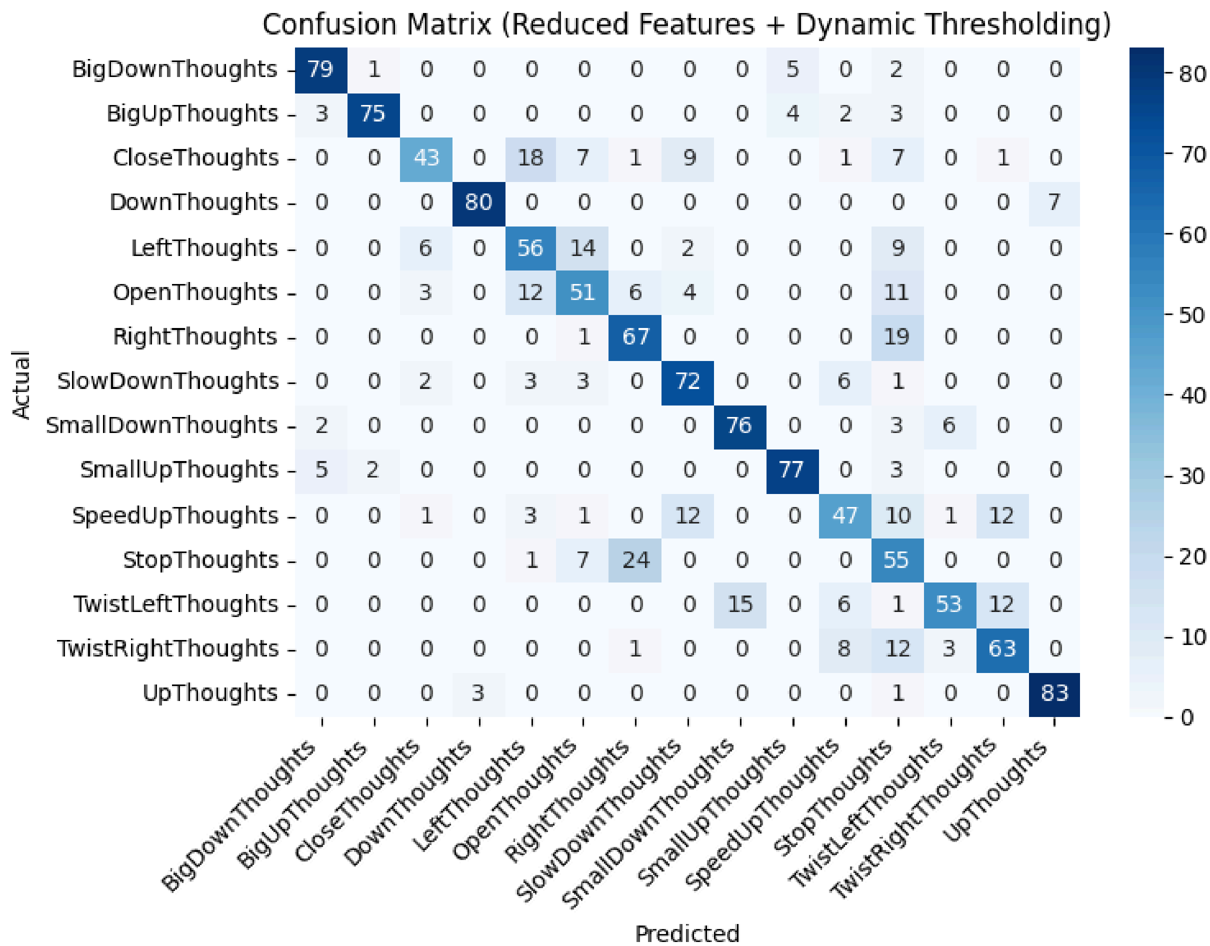


Confusion Matrix generated from Feature Reduction Random Forest Model

As seen by the confusion matrix however, the model was unable to predict some of the classes well at all, specifically CloseThoughts and SpeedUpThoughts. This was due to the fact that certain features from the extraction process are better classifiers for those more dominant actions. This can be seen by the increased predictions for the DownThoughts, UpThoughts, SmallDownThoughts, and SmallUpThoughts. Once the confusion matrix was generated, a classification report was also generated. In the table, the threshold values were generated, as well as the precision, recall, and harmonic mean, also known as an F1 score.

Classification Report (Reduced Features, No Dynamic Thresholding)					
Class	Balanced Thresholds	Precision	Recall	Harmonic Mean	Support
Big Down	0.5	0.888	0.908	0.898	87
Big Up	0.5	0.908	0.908	0.908	87
Close	0.5	0.692	0.517	0.592	87
Down	0.5	0.964	0.920	0.941	87
Left	0.5	0.606	0.655	0.630	87
Open	0.5	0.587	0.621	0.603	87
Right	0.5	0.677	0.770	0.720	87
Slow Down	0.5	0.692	0.828	0.754	87
Small Down	0.5	0.837	0.885	0.860	87
Small Up	0.5	0.876	0.897	0.886	87
Speed Up	0.5	0.627	0.540	0.580	87
Stop	0.5	0.675	0.621	0.647	87
Twist Left	0.5	0.791	0.609	0.688	87
Twist Right	0.5	0.714	0.805	0.757	87
Up	0.5	0.923	0.966	0.944	87

Finally, it was believed that combining both the feature reduction effects with the dynamic thresholding would further improve the accuracy of the model. That was not the case however, as the accuracy dropped to 74.87% instead. While this was an improvement as compared to all of the features with dynamic thresholding, the lack of dynamic thresholding provided a larger benefit than the inclusion of dynamic thresholding.



Confusion Matrix generated from Feature Reduction and Dynamic Thresholding Random Forest Model

As seen by the confusion matrix however, the model was unable to predict some of the classes well at all, specifically CloseThoughts and SpeedUpThoughts. This was due to the fact that certain features from the extraction process are better classifiers for those more dominant actions. This can be seen by the increased predictions for the DownThoughts, UpThoughts, SmallDownThoughts, and SmallUpThoughts. Once the confusion matrix was generated, a classification report was also generated. In the table, the threshold values were generated, as well as the precision, recall, and harmonic mean, also known as an F1 score.

Classification Report (Reduced Features, Dynamic Thresholding)					
Class	Dynamic Thresholds	Precision	Recall	Harmonic Mean	Support
Big Down	0.357	0.888	0.908	0.898	87
Big Up	0.481	0.962	0.862	0.909	87
Close	0.266	0.782	0.494	0.606	87
Down	0.285	0.964	0.920	0.941	87
Left	0.226	0.602	0.644	0.622	87
Open	0.241	0.607	0.586	0.596	87
Right	0.197	0.677	0.770	0.720	87
Slow Down	0.244	0.727	0.828	0.774	87
Small Down	0.341	0.835	0.874	0.854	87
Small Up	0.402	0.895	0.885	0.890	87
Speed Up	0.291	0.671	0.540	0.599	87
Stop	0.220	0.401	0.632	0.491	87
Twist Left	0.382	0.841	0.609	0.707	87
Twist Right	0.380	0.716	0.724	0.720	87
Up	0.423	0.922	0.954	0.938	87

Once all of the models were generated and evaluated with the classification reports, the final model was determined from the results. With the highest overall accuracy, the random forest model that utilized feature reduction, while also maintaining the balanced threshold values, was the dominant model. However, while this accuracy of 76.32% looked promising, this only dealt with the testing data, and may not translate over well in real-time data analysis.

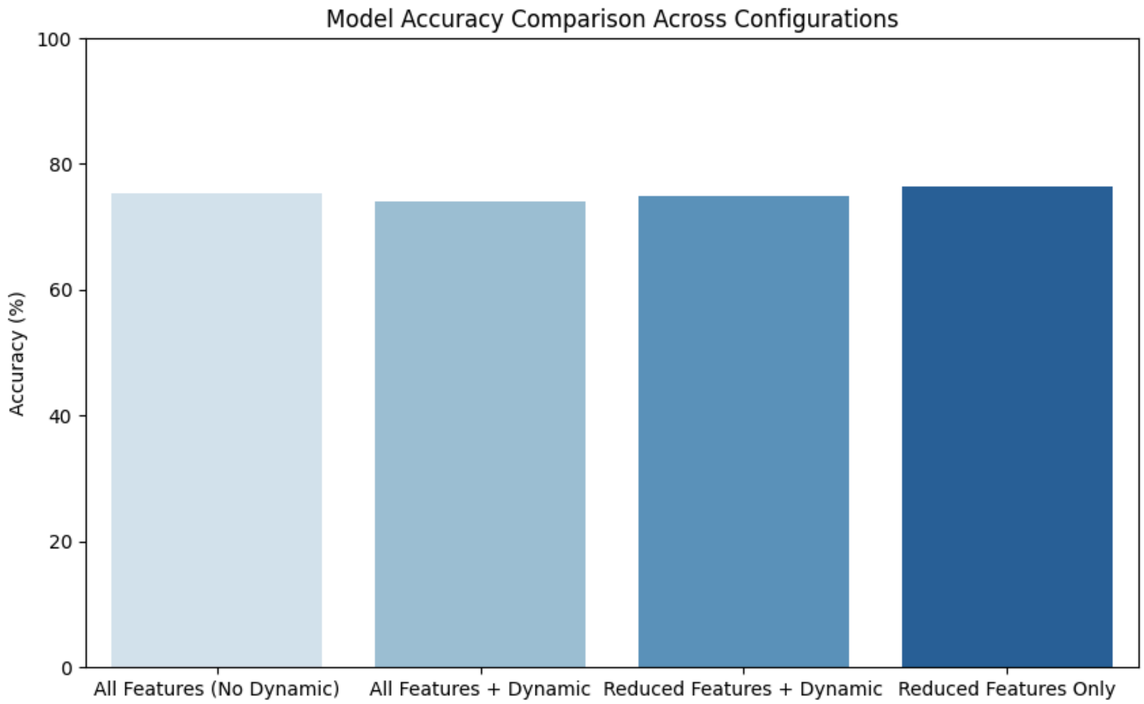


Image of Random Forest Model Accuracy Comparisons

Model	Accuracy
All Features (No Dynamic Thresholding)	75.33%
All Features and Dynamic Thresholding	74.02%
Reduced Features (No Dynamic Thresholding)	76.32%
Reduced Features and Dynamic Thresholding	74.87%

In the end, it was determined that utilizing feature reduction provided a boost to the overall accuracy of the machine learning model, while the utilization of the dynamic thresholding actually harmed the machine learning model. It may be due to the fact that some features are just too difficult to learn from if too many features overshadow its importance. The use of feature reduction was able to highlight only the most discerning of features that helped classification as much as possible, while removing the noise that some of the features may have added to the classification of the model itself. The dynamic thresholding may have been too aggressive with the values for the individual thresholds, so while some classes did benefit from the changes, other classes may have struggled. This tradeoff is a very difficult area to optimize, as too much could lead to overfitting, where the results would only be good to the specific training set, where the real-world analysis of the system would be unable to be accurately predicted. However, too little in terms of changing the threshold values could lead to further overshadowing of the dominant classes, such as UpThoughts, DownThoughts, SmallUpThoughts, and SmallDownThoughts. Because of this difficulty, it is possible the model was too aggressive, and ended up hurting the overall accuracy by trying to support the lesser predicted classes.

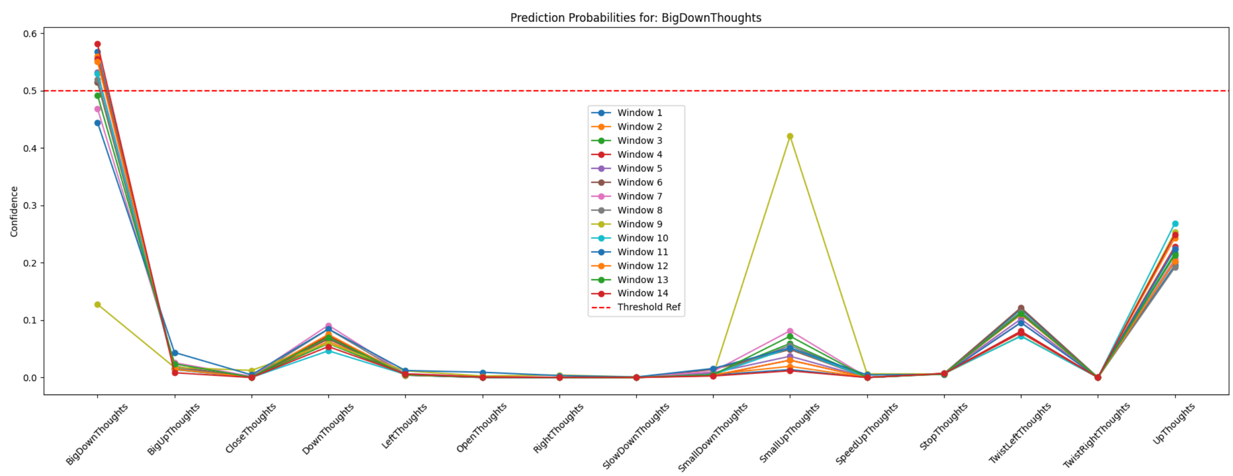


Image of Classification for BigDownThoughts confidence values for each second for the 15 seconds of data

After the static model was trained and tested for accuracy validation, the model was implemented in the real-time environment. In the beginning, the accuracy of the model dropped drastically from 76.32% to 45.30%. This drop was accounted for by the mental drift, and the proportions of the confidence values were recorded per class. It was observed that the dominant classes in the training data were also the dominant classes during the live data stream. This confirmed the idea that the model was better tuned to these classes because of their consistent features during the training process, as even with the mental drifting, the classes were consistently predicted.

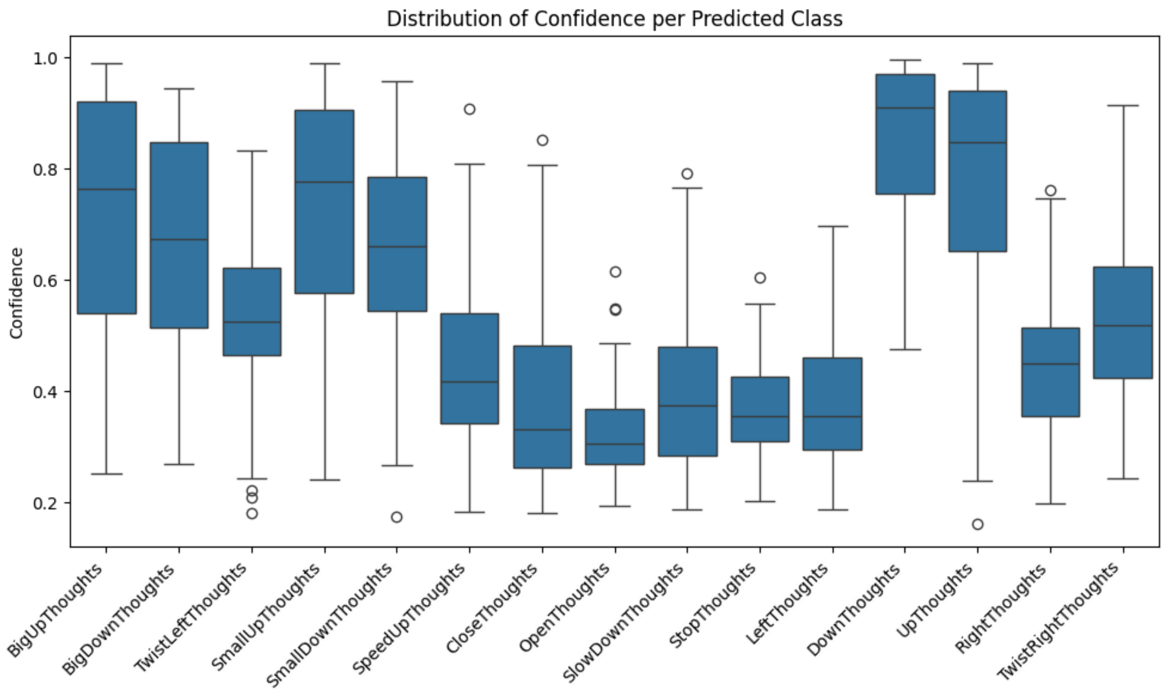


Image of Confidence Distribution Per Class in Live Data

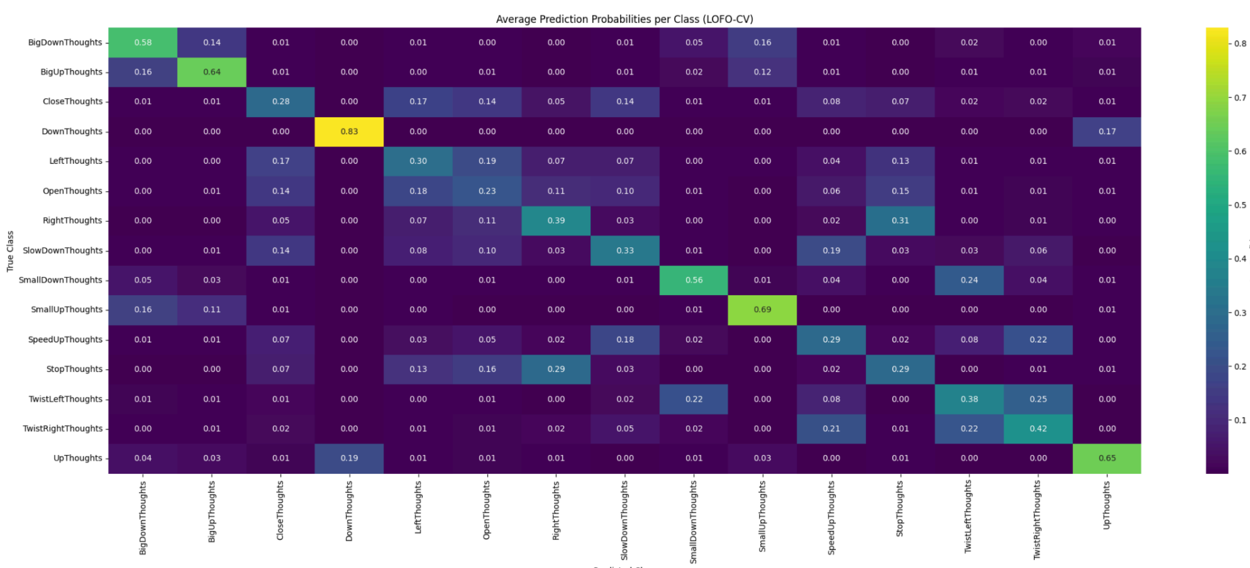


Image of Heat Map for Confidence Per Class Values on Live Data

After applying the dynamic threshold calibration, the model improved the accuracy back to 68%, but was no longer near the expected 76.32%. As the model continued to run, the accuracy of the model continued to slowly drop, and the need for recalibration continued.

Discussion

Challenges and Limitations

Throughout the development of this brain-controlled interface system, several challenges were encountered that required ongoing refinement of the approach. One significant challenge was related to the consistency and quality of the EEG data used for training. Early in the project, the data recording protocol was not standardized, resulting in trials of varying lengths and quality. These inconsistencies complicated data preprocessing steps such as segmentation and normalization. This was addressed by enforcing uniform trial lengths and ensuring each of the 15 classes had the same number of trials and total recording time. Even with standardized recording procedures, the data collection occurred in a highly controlled environment that does not perfectly represent real-world usage. The user was instructed to remain perfectly still, in silence, with eyes closed during imagination of movements, in order to minimize artifacts in the EEG signals [10]. While this controlled setting yielded cleaner data for initial model training, it introduced a limitation: the system's performance may degrade in practical scenarios where such strict conditions cannot be met. Any user motion, such as slight head movements or eye blinks, or external noise could introduce EEG artifacts that the model was not explicitly trained to handle, potentially leading to incorrect classifications. Indeed, during preliminary tests, it was observed that even minor movements by the user registered as significant spikes in the EEG readings. This was one of the reasons why every 20 second recording had to have the first 5 seconds removed. It was because the spikes saturated the data so much that prediction models were incredibly poor when including these areas of the data. For the remaining parts of the data, various signal processing techniques were applied to mitigate this issue—such as band-pass filtering to exclude frequencies outside the 5–60 Hz range of interest and scaling the input features to a fixed range to limit the impact of outliers. The EEG features were effectively constrained to a range of ±225, corresponding to the headset's calibrated output range. The choice of a random forest classifier also conferred some robustness, as ensemble tree methods can handle noisy features better than simpler linear models. These steps collectively reduced the frequency of false classifications caused by artifacts. However, they did not eliminate the problem entirely; the system remains susceptible to occasional misclassifications if the operational environment is not as controlled as the training environment. In summary, there is an inherent limitation in the generalizability of the current model when moving from a quiet, motionless setting to more dynamic real-life conditions.

Another major limitation involved the computational aspects of real-time signal processing. Initially, the machine learning pipeline was implemented in MATLAB due to the dependency on the EEG device's Simulink interface. This caused an unexpected compatibility and performance issue: the EEG acquisition ran in MATLAB R2020a, whereas the classification algorithms were prototyped in MATLAB R2022a. The newer code had to be refactored to work with the older version, and even after integration, it became apparent that MATLAB's execution speed was insufficient for real-time inference with the chosen feature set. In practice, computing the features and making a prediction often took longer than the length of the EEG window itself, leading to a backlog where new EEG data arrived before the previous window's processing had finished. This resulted in accumulating latency that would make real-time control infeasible. The resolution to this issue — and thus a critical turning point in the project — was to transition the classification and control components from MATLAB to Python. By using Python, along with optimized libraries and a compiled random forest model to handle feature processing and classification, and by streaming data out of Simulink via UDP, the prediction computation time was able to be reduced dramatically. In fact, it was reduced on the order of tens of milliseconds per window, down from several seconds in MATLAB. This change eliminated the processing bottleneck, but it introduced additional complexity in terms of system integration and communication. Fortunately, the lightweight UDP protocol proved sufficient for transmitting data between MATLAB/Simulink and Python without notable packet loss or delay on a local machine. After this restructuring, real-time operation was achieved — a clear illustration of how software architecture decisions can impact the feasibility of BCI systems. Nonetheless, the reliance on two different platforms, MATLAB for data acquisition and Python for prediction, is itself a complexity that future iterations might streamline. This could be done, for example, by moving EEG acquisition entirely to a Python-supported library or using a unified development environment.

Model training and optimization presented another set of challenges. The random forest model, while accurate, was computationally intensive to train, especially after the feature space was expanded through signal processing techniques. Each training iteration on the full dataset with hundreds of trees in the ensemble could take on the order of hours to even days to complete on a standard CPU. This long training time significantly slowed the process of experimentation and fine-tuning. The number of times the model was retrained was limited, given that each adjustment to the feature set of hyperparameters could require a fresh training run. Compounding this, careful attention had to be paid to preserving the trained model state for use in real-time testing. The model's performance depended not only on the learned tree parameters but also on the preprocessing steps, such as filter coefficients and feature scaling factors, and post-processing thresholds. If any of these components were not saved and reapplied exactly, the real-time predictions could deviate or fail altogether. Ensuring that the entire pipeline from raw signal to final decision was encapsulated and reproducible was a non-trivial task. There were instances during development when a small change or an oversight in saving a parameter necessitated re-training the model or recalibrating parts of the system, costing valuable time. In an attempt to reduce training time and potentially improve accuracy, an alternative modeling approach using deep learning was explored [11], [14]. A convolutional neural network (CNN) architecture was devised to take EEG time-frequency representations as input, with the hope that the CNN might automatically learn effective features and offer faster inference on a GPU. Training the CNN was indeed much faster, due to leveraging parallel processing on a graphics card, compared to the random forest on CPU. However, despite extensive experimentation, the CNN's accuracy did not reach the level achieved by the TreeBagger random forest. The CNN consistently underperformed in classifying the 15 EEG classes, possibly due to the limited amount of training data, or the complexity of patterns that the random forest was better suited to capture after feature engineering. As a result, the neural network approach was set aside in favor of the more accurate random forest model. This experience highlighted a limitation in the project: the ability to quickly iterate on model improvements was hampered by computational constraints, and some advanced approaches were not fruitful given the time and data available.

Once the model was trained, one limitation was the actual use case. As mentioned prior, the model would only be beneficial for the data set that was saved. In the real-time application, the model struggled to accurately predict the motions of the user, as the model was unable to support the data collected in real time. The calibration was a path to fix this issue, but the accuracy for training was not a real comparison for the accuracy in real-time data streaming. Because of this, the need for constant calibrations can not only indicate a potential issue with the user, but also a fundamental issue with the machine learning model as a whole. This may also indicate improper training, and only with the collection of even more data could this model be improved.

In terms of the control interface for the robotic arm, a limitation became evident in the smoothness and intuitiveness of the arm's movements. The control strategy implemented in this project drives each joint of the arm incrementally based on classified user intentions as discrete classes. The arm's motion, therefore, tends to be somewhat stepwise or jerky, especially when the user's thought pattern changes sequentially to move multiple joints one after another. In natural human arm movement, multiple joints coordinate simultaneously to reach a target position fluidly. By contrast, the current system might move, for example, the shoulder joint for a few cycles, then the elbow, then the wrist, in separate steps, to accomplish a single overall reach. This sequential approach increases the time required to position the arm as desired and can feel less smooth to the user. While the 0.5-second update window helps to mask some of the discontinuity since it updates more frequently than a longer window would, the underlying issue is that the system does not inherently plan multi-joint motions — it simply reacts to one classified command at a time. This is indeed a limitation in the control method: the robot arm executes exactly what it is told for each short interval, without knowledge of a broader goal or end position, leading to inefficiencies in movement.

Finally, a fundamental limitation of the BCI design is the absence of an explicit "idle" or "no command" state in the classification scheme. Because the system must always choose one of the 15 movement classes, including the stop command, each cycle, there is no mechanism for it to remain truly inactive when the user is not intending any movement. In practice, this means the user is required to maintain concentration at all times to keep the system from erroneously activating the arm. If the user's mind wanders or they need to pause thinking about the task, the EEG signals during that period may not clearly correspond to any of the trained movement classes. Yet, under the current design, the classifier will still output whichever class has the highest confidence, even if none of them reach a clear and telling prediction, causing the robotic arm to move potentially without the user's intention. The fixed threshold scheme implemented provides only a partial safeguard: if no class exceeds the threshold, the system defaults to the highest-scoring class anyway. Thus, in the absence of a strong intentional signal, the arm might execute a random or undesired movement. This limitation significantly affects usability — a user cannot simply relax or think about something else without risking an unintended action by the robot. It introduces mental fatigue as the user must continuously either focus on a stop command or otherwise consciously prevent stray thoughts from being misinterpreted by the system.

Potential Optimizations

While the current implementation demonstrates the core functionality of an EEG-driven robotic arm, there are several avenues for improving its performance, reliability, and user-friendliness in future work. One key area of potential optimization is enhancing the robustness of the system to real-world conditions. To address the gap between the controlled training environment and practical use, future iterations of the project could incorporate additional noise-handling and artifact-rejection techniques. For example, more sophisticated filtering or signal processing methods such as adaptive filters or real-time artifact removal algorithms could be employed to dynamically suppress noise from muscle movements or environmental electrical interference. Another complementary approach is to enrich the training data with more diverse scenarios: the user could perform data collection in slightly less restricted conditions, allowing a bit of movement or background noise so that the machine learning model learns to distinguish true intention signals from common artifacts. By broadening the training dataset in this way, the classifier may generalize better and become less sensitive to deviations from ideal conditions.

Improving the smoothness of the robotic arm's movements is another priority for optimization. The current system's joint-by-joint control could be replaced or augmented by an inverse kinematics approach. By implementing inverse kinematics, the system would interpret the user's intent in terms of a desired end-effector position or trajectory, rather than individual joint rotations. The control software could then compute all the necessary joint angles concurrently to move the arm's end-effector directly to the target position in a straight or natural path. This would likely result in more fluid and efficient motions, as multiple joints would move in unison towards the goal, reducing the “stop-and-go” behavior observed with sequential joint commands. Integrating inverse kinematic control would require additional computation and a different interpretation of the user’s EEG commands, such as mapping certain mental commands to spatial directions or target points, but it holds the promise of making the arm’s responses more intuitive and faster from the user’s perspective. In essence, rather than the user having to mentally break down a complex movement into a sequence of smaller motions, they could think of the movement as a whole, such as “move hand to the right”, and let the control algorithm figure out the joint coordination. This change would align the system’s operation more closely with natural human motor behavior and greatly enhance the user experience.

Another important improvement would be to introduce an explicit idle state or no-command class in the brain-computer interface. Incorporating an idle state could involve training the model on examples of “no movement” brain activity. For instance, having the user relax or think of something neutral so that the classifier can recognize when the user is not issuing a command. This way, the system can output a genuine “do nothing” decision in those moments, keeping the robotic arm stationary by design. Alternatively, adaptive thresholding mechanisms could be set such that if the confidence for all movement classes stays below a certain dynamic level, the system interprets it as the absence of a deliberate command and refrains from moving. By allowing the system to occasionally output a null action, the user would gain the freedom to rest or think unrelated thoughts without fear of triggering an unintended arm movement. Such an improvement would significantly reduce user fatigue and make the interface more practical for longer-term use. It would also bring the system a step closer to how humans naturally operate; people do not continuously move limbs—there are many idle moments even during focused tasks, and the technology should be able to accommodate that reality.

From a machine learning standpoint, further optimizations might involve exploring advanced classification algorithms or hybrid models that could potentially offer higher accuracy or faster training/inference times [14], [15]. While the current random forest approach was the most effective among the tested options, the field of BCI is rapidly evolving, and techniques such as deep learning could improve as more data becomes available. For example, a more complex neural network might, with enough training data or computational power, learn features that were not apparent through manual feature engineering, thereby improving accuracy. Future researchers could experiment with architectures like long short-term memory (LSTM) networks or convolutional neural networks explicitly tailored to time-series EEG data, or even hybrid models that combine the strengths of decision trees and neural networks. It would be important, however, to ensure that any increase in model complexity does not reintroduce prohibitive processing delays. Optimizing the code to run on high-performance hardware, such as GPUs or dedicated neural processing units, or employing techniques like model quantization and distillation can help maintain real-time performance even with more complex models. Lastly, refinements in system integration would be beneficial. The current use of separate platforms for EEG acquisition and classification could be unified for simplicity and reliability. For instance, if a future version of the EEG hardware provides a Python API or if an alternative EEG device is used, the entire pipeline could reside in one environment, reducing inter-process communication overhead and potential points of failure.

In conclusion, while the thesis project successfully demonstrated a working brain-controlled robotic arm, these potential optimizations highlight the pathways to transform the prototype into a more reliable, efficient, and user-friendly system. By addressing noise robustness, movement fluidity, idle-state handling, algorithmic improvements, mental drift, and system integration, future work can significantly enhance both the performance metrics and the practical usability of EEG-based robotic control. This would bring such BCI technologies closer to real-world applications, whether in assistive devices for impaired users or in novel human-computer interaction paradigms.

Acknowledgements

I would like to thank my brother Bryan Maus for being a listening ear when discussing issues throughout the research of the project. Talking the problem out with someone drastically improved the thought process on the task at hand, and helped solve some pretty difficult sections of software.

I would also like to thank Dr. George for being a general advisor for the research project, and to keep up to date on the current developments on the research project. This helped keep me motivated to keep making progress on the development side, and to consistently work on the research project.

I would also like to thank the numerous members within the BCI computing lab at California State University - Fullerton for asking questions. If I couldn't answer a question that they asked about a step or stage in the research project, it meant that I had not done enough research to understand it myself. This kept me going in terms of researching answers and really developing an understanding with every step of the process.

References

1. J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain–computer interfaces for communication and control," *Clin. Neurophysiol.*, vol. 113, no. 6, pp. 767–791, Jun. 2002, doi: 10.1016/S1388-2457(02)00057-3.

2. R. Majeed *et al.*, "Brain–computer interface: Trend, challenges, and threats," *Brain Inform.*, vol. 10, Art. no. 20, Aug. 2023, doi: 10.1186/s40708-023-00195-3.

3. M. A. Lebedev and M. A. L. Nicolelis, "Brain-machine interfaces: From basic science to neuroprostheses and neurorehabilitation," *Physiol. Rev.*, vol. 97, no. 2, pp. 767–837, Apr. 2017, doi: 10.1152/physrev.00027.2016.

4. L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, Jan. 2012, doi: 10.3390/s120201211.

5. F. Fang, M. Liu, S. Zhao, K. Guo, J. Xu, and H. Wu, "Control of a robotic arm with an optimized common template-based CCA method for SSVEP-based BCI," *Front. Neurorobot.*, vol. 16, Mar. 2022, Art. no. 838258, doi: 10.3389/fnbot.2022.838258.

6. J. R. Wolpaw and E. W. Wolpaw, *Brain-Computer Interfaces: Principles and Practice*. Oxford, U.K.: Oxford Univ. Press, 2012, doi: 10.1093/oso/9780195388855.001.0001..

7. J. Clausen, "Conceptual and ethical issues with brain–hardware interfaces," *Curr. Opin. Psychiatry*, vol. 24, no. 6, pp. 495–501, Nov. 2011, doi: 10.1097/YCO.0b013e32834bb8c5.

8. J.-H. Jeong, K.-H. Shim, D.-J. Kim, and S.-W. Lee, "Brain-controlled robotic arm system based on multi-directional CNN-BiLSTM network using EEG signals," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 28, no. 5, pp. 1226–1238, May 2020, doi: 10.1109/TNSRE.2020.2981685.

9. N. M. Erhardt *et al.*, "Comparison of dry and wet electroencephalography for the assessment of cognitive evoked potentials and sensor-level connectivity," *Front. Neurosci.*, vol. 18, Nov. 2024, Art. no. 1441793, doi: 10.3389/fnins.2024.1441793.

10. G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: A general-purpose brain–computer interface (BCI) system," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1034–1043, Jun. 2004, doi: 10.1109/TBME.2004.827072.

11. G. Dornhege, J. del R. Millán, T. Hinterberger, D. J. McFarland, and K.-R. Müller, "The Berlin Brain–Computer Interface: Machine learning-based detection of user specific brain states," in *Toward Brain–Computer Interfacing*, MIT Press, 2007, pp. 85–102.

12. J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 2951–2959, 2012.

13. B. Abibullaev, A. Keutayeva, and A. Zolllanvar, "Deep learning in EEG-based BCIs: A comprehensive review of transformer models, advantages, challenges, and applications,"

IEEE Access, vol. 11, pp. 138163–138185, Nov. 2023, doi: 10.1109/ACCESS.2023.3329678.

14. J. Arshad *et al.*, "Intelligent control of robotic arm using brain computer interface and artificial intelligence," *Appl. Sci.*, vol. 12, no. 21, Art. no. 10813, Oct. 2022, doi: 10.3390/app122110813.

15. J. S. Brumberg, S. D. Lorenz, B. V. Galbraith, and F. H. Guenther, "The Unlock Project: A Python-based framework for practical brain–computer interface communication 'app' development," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2012, pp. 2505–2508, doi: 10.1109/EMBC.2012.6346473.

16. V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "EEGNet: A compact convolutional neural network for EEG-based brain–computer interfaces," *J. Neural Eng.*, vol. 15, no. 5, Art. no. 056013, Oct. 2018, doi: 10.1088/1741-2552/aace8c.

17. C. Jeunet, B. N'Kaoua, S. Subramanian, M. Hachet, and F. Lotte, "Predicting mental imagery-based BCI performance from personality, cognitive profile and neurophysiological patterns," *PLoS One*, vol. 10, no. 12, Art. no. e0143962, Dec. 2015, doi: 10.1371/journal.pone.0143962.

18. A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch, "A survey of signal processing algorithms in brain–computer interfaces based on electrical brain signals," *J. Neural Eng.*, vol. 4, no. 2, pp. R32–R57, Jun. 2007, doi: 10.1088/1741-2560/4/2/R03.

19. K. K. Ang, C. Y. Chin, C. Wang, C. Guan, and H. Zhang, "Filter bank common spatial pattern algorithm on BCI competition IV datasets 2a and 2b," *Front. Neurosci.*, vol. 6, Mar. 2012, Art. no. 39, doi: 10.3389/fnins.2012.00039.