ELEC490 Final Report

FacialPC

Group 31:

Brandon Caron, Chris Gritter & Alan Pan

Queen's University Kingston

Executive Summary

There are many different conditions that may prevent an individual from operating a mouse and keyboard with their hands. Being unable to operate a computer could cause many challenges in today's climate where technology is rapidly developing. To rectify this problem, FacialPC has been developed. FacialPC is an inexpensive user driven solution that simulates mouse and keyboard operation with the user's face. FacialPC utilizes facial recognition, voice recognition and neural network technologies to convert simple everyday actions into computer input. This includes the use of face and eye detection algorithms, speech-to-text algorithms and eye classification from a trained convolutional neural network. The purpose of this report is to demonstrate the project and thoroughly analyze the research and design process, implementation and evaluation of the FacialPC program. The results of this analysis are used to evaluate the successfulness of the project and create goals for this project moving forward.

Table of Contents

List of Figures

List of Tables

1 Project Background and Goals

1.1 Problem Description

There are many different conditions that may prevent an individual from operating a mouse and keyboard. These conditions include loss of upper-limbs, people with fine motor control restrictions such as Parkinson's and severe arthritis, and individuals with large motor control issues such as Multiple Sclerosis (MS). Being unable to operate a computer could cause many challenges in today's climate where technology is rapidly developing. The U.S. Bureau of Labor Statistics found that 63% of all jobs require use of mouse and keyboard [1]. Additionally, 99.4% of all jobs require the use of hands. By enabling the physically disabled to effectively use a computer, we could expand their potential in today's job market.

A study published in Occupational Medicine found that 18% of upper-limb amputees did not make a return to work [2]. We believe that by providing a cost-effective method for these individuals to operate a PC, it could improve their resume and skillset to help this group make a return to work.

Another study published in the Annals of the Rheumatic Diseases about "which patients stop working because of rheumatoid arthritis" found that the prevalence of work disability in their studies varied from 29% to 50% by around five years of disease duration [3]. The researchers noted that people with Rheumatoid Arthritis (RA) holding manual jobs were at the most risk for RA-related work disability. However, moving to a desk job wouldn't seem like a viable option for these individuals with RA. By providing them the ability to effectively transition to a desk job with the hand-free operation of a PC, this could not only help them stay in work, but take away the requirement for them to use their hands all day, and potentially alleviate symptoms.

In addition to the benefit of maintaining employment working with a PC, the physically disabled would also benefit from personal use. A Pew Research Center fact sheet states that 78% of all Americans use social media to stay connected [4]. However, physically disabled Americans are less likely than those who don't have a disability to report using the internet on a daily basis (50% vs. 79%) [5]. We think that by making computer use more intuitive, it could help the physically disabled join social media with their friends and benefit from the endless entertainment provided by the internet.

In conclusion, the problem to be solved is the ability to operate a mouse-cursor and keyboard-input without the user needing to use their hands in order to do so. This would allow the physically disabled to benefit from the technology that the rest of the world is using, and rapidly evolving.

1.2 Project Impact

1.2.1 Economic Impact

The purpose of solving the problem is to help the physically disabled increase their skillset in today's job market. With 63% of all jobs in the United States requiring use of mouse and keyboard [1], the team believes that by allowing the physically disabled to be able to operate a mouse cursor and keyboard input as effectively as their competition in the job market, this could bring down the unemployment among the people in these markets.

It is difficult to estimate the exact economic impact of enabling the physically disabled who could not otherwise operate a PC to operate one. Based on the statistics outlined in the previous section, it is believed that there is a sizable amount of people in the USA alone that could benefit from the effective use of a mouse and keyboard without the use of their hands. In the following subsection, the addressable market will be estimated using USA Occupational statistics. The personal use of this software will not be included as currently there are limited statistics involving how many physically disabled individuals are unable to effectively operate a PC.

1.2.2 Addressable Market

There are two main categories that will be the focus of the total addressable market (TAM) for solving the problem outline in Section 1.1. The first will be individuals with upper-limb loss, and the second will be individuals with motor restrictions – specifically looking at arthritis. The following critical assumptions will be used:

- People with upper-limb loss and motor restrictions currently have trouble operating a PC.
- Half the people who are not returning to work could make a return if they were able to operate a PC.
- Only 25% of people who will benefit, will be willing to give the program a try and return to work.

Table 1 displays the statistics that will be used to calculate the TAM.

Table 1: Statistics used to calculate TAM.

Upper-Limb Loss	Motor Restrictions
50,000 new amputations every year in USA based on information from National Center for Health Statistics ratio of upper limb to lower limb amputation is 1:4 [6].	Rheumatoid arthritis (RA) studies varied from 29% to 50% not working by around five years of disease duration [3].
A study published in Occupational Medicine found that 18% of upper-limb amputees did not make a return to work [2].	The lifetime risk of developing RA is 3.6 percent for women and 1.7 percent for men [7].
	About 50% of all women and 25% of all men will experience the stiffness and pain of osteoarthritis (OA) of the hands[8].
	83,460,000 men and 74,078,000 women in workforce in 2019 [9].

Focusing first on the Upper-Limb loss category; by using the statistics in Table 1 and following the critical assumptions outlined, the TAM can be calculated to be:

*A_TAM = 50,000 amputees/year * 25% upper-limb/amputee * 18% do not return to work * 25% try software*

A_TAM = 563 users/year

Just focusing on upper-limb amputees that are struggling to make a return to work, the software will attract 563 users per year.

Next, the team looked at individuals with motor restrictions. The most major market would be people who develop rheumatoid arthritis. Rheumatoid arthritis occurs when the body's immune system attacks healthy tissue that protects the joints. The resulting symptoms can be similar to those of osteoarthritis, including pain, inflammation, and redness; usually attacking the hands first.

*RA_TAM = 29% do not return to work * (3.6% * 74E6 women + 1.7% * 83E6 men) * 25% try product*

RA_TAM=295,000 users

The team also looked at cases of osteoarthritis (OA) in the hands which is when the protective cartilage that cushions the ends of your bones wears down over time. For this market the team did not necessarily consider the option of people returning to work as the pain is often manageable, however the team considered them as potential customers because not needing to type and operate a mouse all day will alleviate the symptoms of OA. Because of the mentioned considerations, we will modify the percentage of users in our critical assumptions from 25% down to 1%.

*OA_TAM = (50% * 74E6 women + 25% * 83E6 men) * 1% try product*

OA_TAM = 577,000 users

By considering that 1% of people with OA will be willing to try the product if it helps operate a PC hands-free, the software will gain an additional 577,000 users.

Overall, by only considering the cases of people with upper-limb loss returning to work and individuals with arthritis, it is estimated that the product will attract 872,000 users and an additional 1000 every two years.

Note that there are plenty of other groups of people who currently have trouble operating a mouse and keyboard. To get a more accurate estimate of the TAM, a study should be conducted considering the percentage of physically disabled Americans that currently have trouble operating a PC due to their limited use of their hands and believe that they would benefit from the solution.

1.2.3 Social, Cultural & Ethical Impacts

Although there are solutions that already exist for the discussed problem, the idea behind our solution is that it will be affordable, completely non-invasive and easily accessible for all. The purpose is not to net a large sum of money off the product, but to provide a service. Current solutions such as the Quha Zono – a hands-free head mouse utilizing gyroscope sensors to move a mouse cursor – cost a minimum of $1000. The team hopes to make the product affordable such that people can try to utilize the product risk-free.

1.2.4 Risk Analysis

There were many risks associated with the project at various points in the development cycle. When developing any product, there’s always a risk that the product already exists in the market. To mitigate this risk, extensive market research was conducted to ensure the feasibility of the project. When conducting market research there were many questions that need to be answered. What similar products are already on the market? What are their strengths and weaknesses? Which product has the best hold on the current market? Can we improve on this product and seize control of the market with the resources we have? The answers to these questions helped determine the overall risk of entering the current market and helped establish a baseline for the design phase.

There are also several risks related to the technical aspect of the project. The most significant technical risk would be software failure. Despite the rigorous testing of the program, there is always a chance that the final product could be shipped or updated with a bug or an error that was not caught by the design team. This would heavily impact the team’s reputation and could severely lower the user’s technical capabilities. To reduce the probability of this risk as much as possible, extensive tests were carried out during and after the design phase. If the design team is ever made aware of a post launch bug or an error, the issue would be identified and fixed as soon as possible. This would restore the functionality of the program and minimize the impact to the team’s reputation. Another technical risk to consider, is the risk of hardware incompatibility. If a hardware failure occurred, such as a crash or webcam failure, the fault would not lie with the design team. However, steps have be taken to ensure that the program would be compatible with as many webcam models as possible. This makes things easier for the consumer and does not force them into spending more money to buy a specific webcam. To achieve this, extensive tests have been done with webcams from many different brands with varying specifications.

An additional risk to consider is related to the operation of the program. Due to the disabilities faced by many of the consumers, the program must be quick and easy to learn. If the program is difficult to learn and use, the user may get frustrated or upset. This could lead the user to give up on the continued use of the program. To mitigate this, the final product should be tested by people from various age groups with differing amounts of experience with computers. This would give the team a good idea of how quickly people learn to use the program. If the usability needs to be improved, an interactive tutorial could be introduced to teach the user in a quick and comprehensive way.

Another aspect to consider is how the program affects the human body. The continued use of this program will involve long intervals focused on a computer screen. Prolonged periods of staring at computer screens can lead to Computer Vision Syndrome (CVS) [10]. Common symptoms of CVS include eye strain, headaches, blurred vision, dry/itchy eyes and neck and shoulder pain [10]. Consumers may discontinue their use of the program if it gives them CVS, and it may negatively affect the team’s reputation. This could also stop companies from integrating the program into their environment if it causes their employee's harm. To reduce the harm CVS can cause, the program will have a built-in function reminding users with periodic alerts to take a break and rest their eyes. This should help reduce the odds that a user gets CVS, but the decision to take a break falls to the user.

1.3 Project Goals

The goal of the project was to create a completely non-invasive software that would allow a user to operate a mouse-cursor and keyboard with the use of their hands.

1.3.1 System Specifications

Table 2 outlines the project specifications set before beginning the project.

Table 2: Project goals set in September 2020.

1	**Functional requirements**
1.1	Move cursor to anywhere on screen using only head movement: • Tilt head left/right to move cursor left/right • Tilt head up/down to move cursor up/down With error checking for: • User turning away from screen
1.2	Being able to click anywhere on screen: • Left click by closing only your left eye for designated amount of time then opening • Left click and hold by keeping left eye closed • Double left clicks by winking your left eye twice in succession • Right click by closing only your right eye for designated amount of time then opening With error checking for: • Closing right eye for too long does not cause multiple clicks • Blinking does not cause any clicks on the screen
1.5	Operate the keyboard hands free using voice recognition software: • Audio cue to activate the voice recognition for typing ("Hey Computer") • Detection of when user is finished talking • Input the spoken string as a keyboard input
1.6	Operate the keyboard hands free through use of an on-screen keyboard
1.7	Voice recognition audio cue in order to active/deactivate the software • "Hey Computer, disable the program" for example
2	**Interface requirements**
2.1	Seamless interface on side of screen which includes:
2.2	• Activate/deactivate software
2.3	• Adjust cursor sensitivity
2.4	• Bring up on screen keyboard
3	**Performance requirements**
3.1	Seamless operation of cursor, allowing for a person to operate the computer without a mouse
3.2	Being able to left-click and right-click anything on the screen
3.3	93% accuracy in classification of whether each eye is open or closed
3.4	93% accuracy in classification of head position (tilted left/right/up/down)
3.5	Voice recognition recognizes audio cue ("Hey Computer") 90% of the time
3.6	Voice recognition will successfully identify 95% of words in a string when used for keyboard input

2 Design Process & Solution

The project was divided into 4 main stages: planning, research and development (R&D), integration, and testing. In the planning stage the team discussed the goals of the project, the necessary features to fulfil those goals, and how each feature would be implemented. In the research and development stage, each of the three team members took lead on a specified subtopic. Each team member conducted research on their subtopic in order to gain a knowledgebase and then shared their findings with the team. During development, all members collaborated with each other on the subtopics, however the lead on a given subtopic was responsible for ensuring deliverables are met and all tasks were assigned. During this development stage, features were implemented in standalone scripts as a proof of concept. During integration, all team members' scripts were compiled in a modular manner such that the code could be easily modified and reused in future projects. More details on each of these stages will be provided in the following sub-sections.

2.1 Planning

During the planning phase of the project, the team first identified the functional, interface, and performance goals of the project. These have been presented in Section 1.3.1 in Table X. After the requirements were established, the team developed a methodology in two stages: approach and design tools.

2.1.1 Approach

The following is the proposed design approach from September 2020.

The designed software is intended to set people's hands free from the keyboard and mouse by utilizing the technologies of computer vision and natural language processing. The software should be programmed as a computer embedded program, which would use the user's face and voice to simulate a mouse and keyboard. The users would be able to control the mouse's movement by tilting their head up, down, left or right. They would also be capable of performing specific actions, such as a double click or right-click, through some facial actions. Furthermore, the users should also be able to use voice-to-text input or an onscreen keyboard to replicate a real keyboard.

2.1.2 Design Tools

In September 2020, the team did preliminary research to decide which software tools would be required in order to realize the goals. The team set on python as the programming language that would be used to create the software because all team members were familiar, and it contains plenty of useful artificial intelligence and image processing libraries. The team then looked for python libraries that would be beneficial to the design of the program.

The team determined that *OpenCV* would be used for the facial recognition aspects of the solution. *OpenCV* has a lot of built-in functionality related to classification of different facial features. This would allow the design team to get a quick start on facial feature detection and allow for a larger focus on detecting and distinguishing between the positions of different facial features. *OpenCV* also contains methods for live video capture, which was of vital importance to the program.

The team set on using the *Speech Recognition* and *PyAudio* libraries for the voice recognition aspects of the solution. The *Speech Recognition* library is very user friendly and contains built in methods for converting an audio files to string by utilizing any of a wide variety of respected API's including, *Google Web Speech*, *Google Cloud voice-to-text*, *CMU Sphinx* and *IBM Speech to text*. The *PyAudio* library allows for the design team to take live audio in as an input, which can then be used in the *Speech Recognition* library.

The *PyAutoGUI* package was established as the core package be used to simulate the mouse and keyboard. This package contains built in methods for moving and clicking the mouse, as well as simulating various key presses. *PyAutoGUI* can also interact with elements of various windows applications which would allow for increased compatibility in the future.

The team decided that the *TensorFlow and Keras* library would be used for the neural network development. *TensorFlow* is a very extensive machine learning library that contains a lot of built-in functionality for creating neural networks. It allows for simple implementations of classification algorithms, which would be used to distinguish between the different positions of the facial features.

2.2 Research and Development

Once the team had a preliminary plan, they moved onto the research and development phase. During this the overall project was divided into subtopics and then distributed amongst the team members. Each team member conducted research into their subtopic and shared their findings with the team. Then, each team member began developing scripts that would fulfill certain features related to their subtopic. The subtopics were divided as follows:

- Chris would be responsible for *feature detection*,
- Alan would be responsible for *neural network development for facial feature classification*,
- Brandon would be responsible for *voice recognition* and *windows control*.

The *feature detection* subtopic consisted of creating a methodology for detecting different features on the face and determining what these features could be used for. The *neural network development for facial feature classification* subtopic consisted of developing neural networks to classify chosen features, for example, classifying whether an eye is open or closed. The *voice recognition* subtopic consisted of developing a methodology for detecting speech, converting it to a string, and then executing commands based on that string. *The windows control* subtopic consisted of developing a methodology for controlling the mouse, keyboard and anything related to controlling the operating system (e.g., opening programs). Each of the subtopics and their respective implementation will be discussed in more details in the following sub-sections.

2.2.1 Feature Detection R&D

For the facial recognition section of the FacialPC project, the *OpenCV* library was used. *OpenCV* is a cross-platform open-source library containing many programming functions related to computer vision. For FacialPC, *OpenCV* was used for all image operations and facial feature detection functions. For feature detection, the cascaded face detection method by Viola and Jones was utilized with pretrained Haar Cascade classifiers. More details about cascaded face detection algorithms and Haar cascades can be found in Appendix I. The pretrained classifiers that were utilized were the “haarcascade frontalface default” and “haarcascade eye” classifiers found in the data/haarcascades folder in OpenCV. These classifiers are prepared upon program initialization using the *OpenCV* command *cv2.CascadeClassifier*.

For the sake of discussion, the facial recognition section of FacialPC can be separated into three modules:

1. Face detection
2. Eye detection
3. Eye classification

For the face detection section, the image to run the algorithm on is a captured frame from a webcam. This frame is then run through the face detection classifier using the *OpenCV* command *detectMultiScale*. This command runs the image through the face detection classifier and outputs an array containing the coordinates of the box that surrounds all detected faces. This command takes in the image, and two other parameters: *scaleFactor* and *minNeighbors*. These parameters adjust the cascaded face detection algorithm and dictate the effectiveness of the face detection. The optimal values for face detection were found to be a *scaleFactor* of 1.14 and a *minNeighbors* of 5. The *scaleFactor* parameter controls how much the image size is reduced at each image scale in the Viola-Jones detection algorithm. For example, a scale factor of 1.05 reduces the size of the image by 5% for each rescaling step. Lower values such as 1.05 are more reliable, as the chance of finding a matching model is higher, however this makes the algorithm much more computationally expensive [11]. The *minNeighbors* parameter specifies how many neighbors each candidate rectangle should have [11]. This parameter affects the quality of detected faces. Higher values result in less detections, but the detections are higher quality [11]. To find the most optimal parameter values for the FacialPC program, multiple experiments were carried out. A script was created that runs the face detection algorithm for a set number of frames, on images captured through a webcam at initial parameter values. The accuracy of the algorithm was assessed by comparing the number of frames where the face was captured, to the total number of frames captured. These parameter values were then incremented and retested. The parameters chosen were the parameters that give the highest accuracy. Multiple filtration functions are applied to the output to remove false face detections. In order to accomplish this, the largest, most central face was passed through to the next stage. Faces that were too small are discarded. This filtration system also accounts for when there are multiple people on camera, by filtering the output to only include the most central face. The output image is then resized to become an image of the face that was detected, and this image is input to the eye detection stage.

For the eye detection stage, the image of the face output from the face detection algorithm is run through the eye detection classifier using the same *detectMultiScale* command that was used for the face detection stage. The output is an array containing the coordinates of the box that surrounds all detected eyes. The optimal parameters for eye detection were found to be a *scaleFactor* of 1.05 and a *minNeighbors* of 9. The optimal parameter values for eye detection were obtained using the same experimental process that was used for the face detection algorithm. Multiple filtration functions are applied to the output to remove false eye detections. This is accomplished in two ways. First, all detected eyes are checked to see where they are on the y-axis of the face. If they eyes do not fall within the top three fifths of the face, they are discarded. Secondly, the remaining eyes are checked for size. If the size of the eye is not within the range of 35 to 60 pixels in both the x and y directions, it is discarded. These filtration functions work together to reduce the classifiers output to the two desired eyes. Output images are created for each of the detected eyes, and these images are input to the classification phase.

In the eye classification phase, the images of the detected eyes are reformatted and classified as open or closed. Firstly, the coordinates of the eyes are used to determine which side of the face they fall on. If the eye falls on the left side of the face, it is classified as a left eye, and if the eye falls on the right side of the face, it is classified as a right eye. The images of the eyes are then run through a function called *frameEye*. This function resizes the eye images to match the input specifications of the eye state classifier and centers the eye in the new image. The coordinates of the eye image are then updated by the coordinates saved in the *prev_eyes* array. This array functions as a backstop in case an eye is not detected on a specific frame. Every time an eye is detected in each frame, the *prev_eyes* variable is updated with the detected eye's coordinates. If an eye is not detected in a specific frame, the coordinates of that eye from the previous frame are used instead. For example, if the right eye of a frame is not detected by the algorithm, the coordinates of the right eye that was detected on the previous frame are used to get an image of the right eye in the current frame. Next, the images of the eyes are reformatted into a *NumPy* array and run through the eye state classifier. The eye state classifier takes in the image of the eye and outputs if it is in the open state or closed state.

After the three phases for the frame are completed, the program moves on to the next frame that was captured. Multiple variables are used to keep track of the classification output from previous frames. For example, a counter is incremented every time an eye is detected as closed. If the eye is detected to be in the closed state for 8 consecutive frames, the command is given for the computer to simulate a click for that eye's side (Ex. Left eye closed for 8 consecutive frames simulates a left click). If there is a frame where the eye is detected as open, the counter is reset to zero.

The feature detection phase also implements the mouse movement for the program. This was accomplished using what the design team calls a "dead zone". During the face detection phase, the face that was detected is run through the calibrate function. This function calculates the center of the face and creates a box that surrounds the calculated midpoint. This box is the dead zone. On subsequent frames, the center of the face is recalculated. If the center of the face is calculated to be outside of the dead zone's coordinates in any direction, a command is given to move the mouse in that direction. This is accomplished using the *checkCenter* function. At any point in the program, the user can activate the calibrate function and remake the dead zone to fit to their current orientation.

2.2.2 Neural Network Development R&D

When training the neural network, data preprocessing such as normalizing, centering, and standardizing was performed on each image in the dataset. Normalizing often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1. By bringing all the values of numeric columns in the dataset to a common scale it would eliminate bias placed on larger values. The normalized data would allow the training process to become more efficient, which results in a more optimized set up.

Standardizing often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, one might subtract the mean and divide by the standard deviation. The result is a standard normal random variable with mean of 0 and standard deviation 1.

Standardizing the features around the mean and dividing by the standard deviation is important when we compare measurements that have different units. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. For example, different lighting conditions contribute to different pixel values. If the model was trained on well-lit images, it would be prone to miss classify input images that were not taken in the same conditions. By standardizing and normalizing training images as well as input images, we can remove dependencies on lighting conditions and increase robustness of the model.

For model selection, various architectures were evaluated for suitability with considerations to the dataset, classification accuracy, complexity, and training method. The multi-layered perceptron model, convolutional neural network model, recurrent neural network model, and k nearest neighbor model.

K nearest neighbor model is an unsupervised neural network model that does not fit the training method that we are seeking to perform which resulted in the elimination of this model.

Multilayer Perceptron (MLP) is a class of feed-forward artificial neural networks. An MLP consists of three main layers of nodes: an input layer, a hidden layer, and an output layer [12]. In the hidden and the output layer, every node is considered as a neuron that has a nonlinear activation function. MLP uses backpropagation to adjust its weights which results in incrementally minimizing the cost functions. MLPs are most ideal for tabular datasets, classification prediction problems, and regression prediction problems [12].

Convolution neural network (CNN) model processes data that has a grid pattern such as images. It is designed to learn spatial hierarchies of features automatically [12]. CNN consists of three types of layers the convolution, pooling, and fully-connected layers. The convolution and pooling layers perform feature extraction, and these extracted features are mapped into the final output by the fully connected layer. Some of the application areas of CNN are in image recognition, image classification, object detection, and face recognition.

In Recurrent Neural Networks (RNN), the output from the previous step is fed back as input to the current step. The hidden layer in the RNN enables this feedback system. This hidden state can store some information about the previous steps in a sequence. RNNs are most suitable for natural language processing, machine translation, video tagging, and text summarization [12]. Since the architecture of the recurrent neural network allows the output of previous steps to be stored it is more suitable for natural language processing. Since the architecture of machine learning models have a strong relationship to how well it performs depending on the task. Recurrent neural networks architecture would not be suitable for eye classification.

A MLP model and a CNN model were built to compare their effectiveness using the same dataset. The metrics of comparison were training time, accuracy, and model complexity.

Various optimizers and loss functions were tested by re-runs of the of model in order to determine which loss function and optimizer suited the model best. Sparse categorical Cross entropy loss function was selected with an Adam optimizer. A SoftMax activation function was used as the output function for the sequential model.

The CNN model obtained higher accuracy with the training and testing set than the MLP model. Although the CNN model is more complex than the MLP model, it required the same training time. The MLP model requires 100 epochs with a learning rate of 0.00001 to achieve an accuracy of 95.41% on the training set and 90.75% on the testing set. While the CNN model requires 30 epochs with a learning rate of 0.0001 to achieve 98.56% accuracy on the training set and 94.61% accuracy on the testing set. These results can be seen below. Due to the CNN performing better than the MLP in all aspects, the CNN was used in classifying open and closed eyes. More detailed data of the two models is shown in Appendix II.

```
Model: "sequential" training results
Epoch 100/100
12/12 [==============================] - 0s 13ms/step - loss: 0.1615 - accuracy: 0.9541
Model: "sequential" test results
35/35 [==============================] - 0s 2ms/step - loss: 0.2942 - accuracy: 0.9075

Model: "CNN training"
Epoch 30/30
113/113 [==============================] - 18s 159ms/step - loss: 0.0349 - accuracy: 0.9856
Model: "CNN testing"
35/35 [==============================] - 1s 32ms/step - loss: 0.1446 - accuracy: 0.9461
```

The model parameters were adjusted and retested to increase model accuracy. Parameters such as the activation function, loss function, and optimizer were varied to evaluate the optimal parameters of the model. The model architecture was tested by varying the complexity of the architecture. Starting with the basic CNN model, it took more epochs to learn the relationship between the training images and labels with low accuracy. As the model complexity increases, it becomes more accurate at fitting the training images. However, due to increasing model complexity overfitting the training data set becomes a concern. If the model were to overfit the training data it would lose its generalizing ability which would render it inaccurate at classifying input images. Complexity was reduced by removing the nodes and layers of the architecture when overfitting was observed.

In order to convert images of eyes to clicks, sequence of images is passed into the CNN model for classification. In order to separate eye winks from normal blinks, 10 consecutive frames must be classified as a closed eye in order to click. When the model classifies that both eyes are closed, the number of frames is reset. This allows the user to blink without receiving false clicks. The left and right eye have individual frame counters that tracks the number of consecutive frames the model classifies as closed. This allows the user to close their right eye for a right click or a left eye for a left click and both eyes without clicking. The number of consecutive frames required to click can be changed, which would result in more clicks performed. However, by reducing the number of consecutive frames required to click the number of false positive clicks increase. When the number of required frames increase it will reduce the number of false positives, but the duration of a wink or effectively the waiting time is increased. The considerations to the number of frames required is a matter of balancing false positives and waiting time. By testing, 10 frames were found to eliminate 95% of false positives, while keeping the theoretical wait time to be 1/6 to 1/3 of a second. This would depend on the hardware of the machine since some cameras vary from 30 to 150 frames per second on commodity machines.

2.2.3 Voice Recognition R&D

When facing the problem of voice recognition, the python library *Speech Recognition* was used. *Speech Recognition* is a library for performing speech recognition that has support for several engines and APIs, online and offline [13]. The package can quickly be installed using pip. From the various engines/APIs the library supports, the two of interest are the *CMU Sphinx* engine and the *Google Cloud Speech API*. *CMU Sphinx* is an open-source library which uses state of art speech recognition algorithms for efficient speech recognition. It is of interest because it works while offline and since there is no need to access the network, the engine can convert the audio signal into a prediction very quickly. *Google Cloud Speech API* is an API provided by Google within in their clous services. It uses Google's massive database and state of the art algorithms to transcribe audio files or create audio files from strings. It is of interest due to Google's high accuracy when transcribing an audio file. Although *Google Cloud Speech API* is a paid service, it is preferred over the free *Google Speech Recognition API* due to lower delays, more reliable results and the ability to specify keywords. Additionally, because the voice recognition portion of the program must always be listening for voice inputs, it was implemented using the thread class from the python threading library. A thread is a separate flow of execution that allow for multiple processing to be running concurrently. Although the processes are not actually running at the same time, they work well when a process spends most of its time waiting for external events, which is the case with the hot-word detection.

The voice recognition portion of FacialPC can be divided into three stages:

1. Hot-word detection
2. Convert voice to a string
3. Command encoding

In the first stage, the software must listen for a hot-word in order to know when the user is talking to the program. In the implementation of the program, "hey computer" was chosen as the hot-word. The pseudocode for the algorithm developed in order to implement the hot-word detection is shown in Algorithm 1. In order to implement the listening of the microphone input, the *listen* function within the *Recognizer* class within the *Speech Recognition* library was used. This function takes a timeout time (1 second), a phrase limit (3 seconds) and an audio source (the microphone). It then outputs audio data either once it detects the user is done talking, or once the phrase limit is met, whichever comes first. In order to detect a hot word from the audio, the *CMU Sphinx* engine was used. This was chosen because it is quick – due to being locally installed – and it is accurate when given a preferred phrase to look for. The *Speech Recognition* library contains the framework for using *CMU Sphinx* so all that is needed is a call to the *recognize_sphinx* function within the *Recognizer* class. This function takes the audio data and any preferred phrases, then outputs a string of the most-likely estimate of what was said. If the string contains the hot-word, then the program continues to stage 2.

Algorithm 1: Hot-word detector.

```
Loop:
        Listen to microphone input and timeout after 1 second if no sound is detected. If sound is
        detected, then record audio for 3 seconds, or until the input sound stops, whichever is first.
        If: timeout, go to Loop
        Else: Check audio for hot-word
                If: hot-word is detected, go to stage 2
                Else: go to Loop
```

In the second stage the user says what they want to the program, and the audio is converted to a string. The implementation of this stage is very straightforward:

1. Play an audio cue to notify the user that the program is listening for an audio input
2. Listen to the audio input using the *listen* function within the *Recognizer* class
3. Play an audio cue to notify the user that the program is done listening for an input
4. Send the audio to the *Google Cloud Speech API* to convert it to a string. During this step the possible commands are specified as preferred phrases.
5. Send the received string to the encoder (stage 3).

In the final stage, the string is encoded and sent for execution. This stage is implemented through a series of logic. The purpose of the logic is to take the string and encode it in a list as *[command, prefix, [modifiers]]* which makes it clear to the program what the user wants to accomplish. Table 3 provides a description of the possible commands and their respective prefixes or modifiers. First the program splits the string into a list of words and then parses through this list looking for commands. Since some phrases may contain multiple commands, certain commands are given a higher priority in situations where this may happen. For example:

- If the user says: “press and hold Ctrl”, then they want to hold the Ctrl key, which is classified as a hold command not the press command, so hold is given a higher priority than press.
- If the user says: “click and hold”, then they mean they want to click, which is the click command and ‘hold’ would be a modifier, so click is given a higher priority than hold.
- If the user says “recalibrate the dead zone” then that means they want to calibrate, not adjust the size of the dead zone, so calibrate is given a higher priority than dead zone.
- Type is given the highest priority. If we detect ‘type’ we immediately set it as the command, then everything after the word ‘type’ is consider the string that the user wants to type.

After deciding on a command, the logic parses the list of words for possible prefixes or modifiers. Prefixes and modifiers provide context to the command. For example:

- If the command is open, then the modifier would be the program the user wants to open.
- If the command is click, the prefix is the button to click, and the modifiers may be ‘hold’ or how many times they want to click.
- If the command is sensitivity, the prefix may be a specific direction, and the modifier would be the amount and type (absolute, relative, percent). Note that in this case, the user must not implicitly say the type of change as it can be inferred. If the user says “**set** sensitivity **to** ____” then this is clearly

an absolute change. If the user says "**increase/decrease** sensitivity by ____" then this is either a relative or percent change, depending on if they give the value as a percent or not.

Once the string has been passed through the logic, and the data is encoded in the form of *[command, prefix, [modifiers]]*, it is sent to either: the windows control program (Section 2.2.4) to execute the appropriate function, or it is sent to the GUI handler (Section 2.3.4) in order to adjust the appropriate settings.

Table 3: Possible commands and their corresponding prefixes and modifiers.

Priority	Command	Description	Possible Prefixes	Possible Modifiers
1	Type	Type out a string	N/A	- The string to be typed
A-2	Click	Click the mouse	Left (default), right, center	- Number of clicks (double, 3, 4…) (Default is 1) - Hold.
A-3	Hold	Hold a keyboard key	N/A	The key(s) to be held
A-4	Press	Press a keyboard key	N/A	-The key(s) to be pressed -Number of presses
B-2	Calibrate/ recalibrate	Recalibrate the face dead-zone	N/A	N/A
B-3	Dead zone	Adjust the size of the face dead-zone	both (default), X, Y	- An integer value (required). - Absolute, relative or percent change. Default is absolute. - If a relative or percent change, an increase or decrease should be specified.
-	Open	Open a program	N/A	The name of the program
-	Close	Close a program	N/A	The name of the program
-	Release	Release a keyboard key or mouse button.	N/A	The key(s) or mouse button to be released
-	Sensitivity	Adjust the program's sensitivity	both (default), X, Y	- An integer value (required). - Absolute, relative or percent change. Default is absolute. - If a relative or percent change, an increase or decrease should be specified.
-	Start	Enable the program's mouse movement	N/A	N/A
-	Stop	Disable the program's mouse controls	N/A	N/A
-	Invert	Invert the camera view in the GUI	N/A	N/A

2.2.4 Windows Control R&D

For *windows control*, all the commands from the previous section (excluding the ones that adjust program settings) must be correctly mapped to a function which will execute the appropriate actions within the OS.

The program was only designed to work on Windows OS. The *pyautogui* library provides all the controls necessary in order to perform a wide variety of actions within the operating system. In order to open and close programs, the preinstalled python library *subprocess* is used. Table 4 shows the mapping of each command to a function.

It was also necessary to develop a method to move the mouse in order to supplement the facial detection section. The library *pyautogui* has a function called *pyautogui.move(x,y)* which moves the mouse cursor relative to the current position by the specified amount (x,y). This function was used within the facial detection in order to move the mouse.

Table 4: Mapping of each command to a function.

Command	**Functions and Descriptions**	
Type	Function:	pyautogui.write(string, interval)
	Description:	the function takes in the string to be typed, and then sends that to the keyboard input within the OS.
Click	Function 1:	pyautogui.click(button=[‘left’,’right’,’middle’], clicks=num_of_clicks)
	Description 1:	This function clicks the designated mouse button the specified number of times.
	Function 2:	pyautogui.mouseDown(button=[‘left’,’right’,’middle’])
	Description 2:	This function is for when the hold modifier is present. It presses the mouse button down without releasing it.
Hold	Function:	pyautogui.keyDown(key)
	Description:	Presses the designated key without releasing it.
Press	Function 1:	pyautogui.press(key, presses)
	Description 1:	Presses the designated key the specified number of times
	Function 2:	press_keys(keys, presses) which uses: pyautogui.keyDown(key) and pyautogui.keyUp(key)
	Description 2:	If multiple keys are specified, a function was created to press all the keys down, then release all the keys, and it does this the designated number of times.
Open	Function:	app = subprocess.Popen(path)
	Description:	Opens a program from its path. A reference to the process is saved so it can later be closed.
Close	Function:	app.kill()
	Description:	Closes the app where app is the saved reference to a previously opened program.
Release	Function 1:	pyautogui.keyUp(key)
	Description 1:	When the thing to be released is a keyboard key, this function is used.
	Function 2:	pyautogui.mouseUp(button=button)

Description 2:	When the thing to be released is a mouse button, this function is used.

2.3 Integration

In order to integrate all components of the program together, it was compiled in a modular way using object-oriented-programming. The code was divided into 4 modules: facial recognition, speech recognition, windows handler and GUI handler. A UML of all the classes involved in the program can be found in Appendix III.

2.3.1 Facial Recognition Module

The facial recognition module, named facialrecog.py, contains all code related to facial feature detection and classification. Within the module is an object called Watcher. Table 5 summarizes the Watcher's properties and Table 6 summarizes the Watcher's methods.

Table 5: The Watcher object's properties.

Property	**Description**
sens_x, sens_y	Integer in the range [1,100] to control the mouse movement sensitivity in the x- and y-directions.
dead_x, dead_y	Integer (number of pixels) to control the size of the dead zone in the x- and y-directions.
flipped	Set to 1 if the camera is not mirrored, and 0 if the camera is mirrored in the horizontal direction.
start	Set to 1 to enable mouse movement and mouse clicking, and 0 to disable mouse movement and mouse clicking.
limit_x_left	Holds the value of the leftmost x pixel value in the dead zone in the x-direction
limit_x_right	Holds the value of the rightmost x pixel value in the dead zone in the x-direction
limit_y_up	Holds the value of the uppermost y pixel in the dead zone in the y-direction
limit_y_down	Holds the value of the lowermost y pixel in the dead zone in the y-direction
center_x	Holds the value of the current center of the face in the x-direction
center_y	Holds the value of the current center of the face in the y-direction
cap	A VideoCapture object from OpenCV used to capture an image from the webcam.
face	Holds data of the latest face detected in a list: [x, y, w, h] where x is the left_most pixel in the x-direction, y is the uppermost pixel in the y-direction, w is the width of the face, and h is the height of the face.
framed_eyes	A list [left_eye, right_eye] where left_eye and right_eye are images of the most recently captured left and right eye respectively.
prev_eyes	Holds data of the latest eyes detected in a list [left_eye, right_eye] where each left_eye and right_eye hold the data in the form [x, y, w, h].

left_closed_count	A counter keeping track of how many left eyes in a row were detected as closed
right_closed_count	A counter keeping track of how many right eyes in a row were detected as closed
img, img_gray, img_draw, roi, roi_gray	Images captured from the webcam in different formats. Any image with the suffix '_gray' means it is in gray scale. The img_draw is the image in which rectangles are drawn on to show the detected features. The roi images are the webcam image cropped to show just the face.
face_cascade	An OpenCV CascadeClassifier Object holding the face classifier
eye_cascade	An OpenCV CascadeClassifier Object holding the eye classifier
eye_model	The tensor flow model used to classify whether eyes are opened or closed

Table 6: The Watcher object's methods.

Method	**Description**
detect_face(self)	1. Takes an image capture from the webcam 2. Detect any faces. If more than one face is detected take the one closest in size to the previous face 3. If a face is detected, save it in the face property, update the center_x and center_y properties, save all img properties and draw rectangles on img_draw Else, set the face property to an empty list
detect_eyes(self)	1. Detect any eyes from the roi_gray image 2. For each detected eye, check to ensure they are actual eyes by checking the position and size compared to the face 3. If the eye makes it pass the check, if it's on the right side of the face it is a right eye, else it's a left eye. Update the proper index of the prev_eyes property
check_eyes(self)	1. Get the position of the most recently detected left and right eyes from prev_eyes. 2. Crop the roi_gray to single out each of the eyes using the frame_eye function 3. Center and normalize each of the eye images 4. Send each of the normalized eye images through the eye_model to determine each is opened or closed 5. If either the left or right eye is closed, increment their respective counter.
frame_eye(self, x, y, w, h)	A function that takes in the position of an eye, and crops the roi_gray image to output an image of just the eye
calibrate(self)	A function that updates the left, right, upper, and lower dead zone limits based on the current face.
move_mouse(self)	A function which checks if the current center of the face is outside the deadzone. If it is, the function moves the mouse in the direction of the face, at a distance proportional to how far the face is from the deadzone.

click_mouse(self)	The function first checks that the face is within the dead zone. If it is, then it checks each the left and right eye counters. If either counter is greater than some threshold, while the other is less than some threshold, it will click the appropriate mouse button.
draw_face_rectangle(self)	A function to draw a rectangle around the current face on the img_draw image.
draw_dead_rectangle(self)	A function to draw a rectangle around the current deadzone on the img_draw image.
draw_eye_rectangles(self)	A function to draw a rectangle around each eye on the img_draw image.
set_sens_x(self, val)	A function to set the sensitivity in the x-direction
set_sens_y(self, val)	A function to set the sensitivity in the y-direction
set_dead_x(self, val)	A function to set the dead zone in the x-direction
set_dead_y(self, val)	A function to set the dead zone in the y-direction
flip_cam(self)	A function to toggle the flipped property
toggle_running(self)	A function to toggle the start property
face_in_dead(self):	A function to check if the center of the current face is within the dead zone.

2.3.2 Speech Recognition Module

The speech recognition module, named speechrecog.py, contains all code related to speech detection. Within the module is an object called Listener which automatically listens in the background on initialization. The background listener will wait for the hot-word, prompt the user with a sound once it detects the hot-word, listen to the user's command, convert the input audio to a string, and then encode the string in the *[command, prefix, [modifiers]]* format. Table 7 summarizes the Listener's properties and Table 8 summarizes the Listener's methods.

Table 7: The Listener object's properties.

Property	Description
recognizer	Holds an instance of the speech_recognition.Recognizer class
microphone	Holds an instance of the speech_recognition.Microphone class
running	Boolean variable that controls whether the background listener is running
output	A list which holds the output string from the Google Cloud API after the user inputs a command. This is so the encoder can pick it up
print_output	A list which holds print outputs for the GUI
cmds_output	A list which holds encoded commands for the GUI to pick up and adjust program settings (sensitivity, dead zone, etc.)
listener_thread	Holds reference to the thread which is listening in the background
handler_thread	Holds reference to the thread which is doing the encoding
keyword	The hot word that the program is listening for. It is set to "computer"

Handler	Holds reference to a Handler object from the windows handler module
last_cal	The time of the last noise-floor calibration of the microphone
auto_cal_time	The time between noise-floor calibration of the microphone

Table 8: The Listener object's methods.

Method	**Description**
calibrate(self, duration=1, source=None)	A function to calibrate the noise-floor of the microphone
sync_google_cloud(self)	A function to sync to google cloud on initialization of the Listener
stop_listening(self,wait_for_stop=False)	A function to stop the background listener
listen_in_background(self, phrase_time_limit=3)	A function to start the background listener. Called on initialization of the Listener
callback(self, audio, s)	Function called once the hot-word is detected. It gives an audio prompt to the user, converts their input to a string using Google Cloud API, then puts the string in the output list for the encoder to pick up
decoder(self)	The function that takes the users input string and encodes it in the [command, prefix, [modifiers]] format

2.3.3 *Windows Handler Module*

The windows handler module was designed to support speech recognition. It contains an object called the Handler. On initialization, the Handler parses through the windows computer using windows registry keys looking for any installed applications. It saves the name and path of these applications in a dictionary named SOFTWARE_DICT. The function that the Listener calls in order to perform an action is the execute function. The execute function takes in the *[command, prefix, [modifiers]]* format and calls to the appropriate functions. When the user is attempting to open a program, it will take the modifier and find close matches to the keys in SOFTWARE_DICT. If there exist any matches, it takes the closest match and opens that program while saving a reference to the process in the open_applications dictionary, under the same name it had in the SOFTWARE_DICT, so it can be later closed. For all other commands, the execute function calls the proper method within the class based on the command, prefix and modifiers. Table 9 contains a description of all the Handler object’s properties and Table 10 contains a description of all the Handler object’s methods.

Table 9: Handler object’s properties.

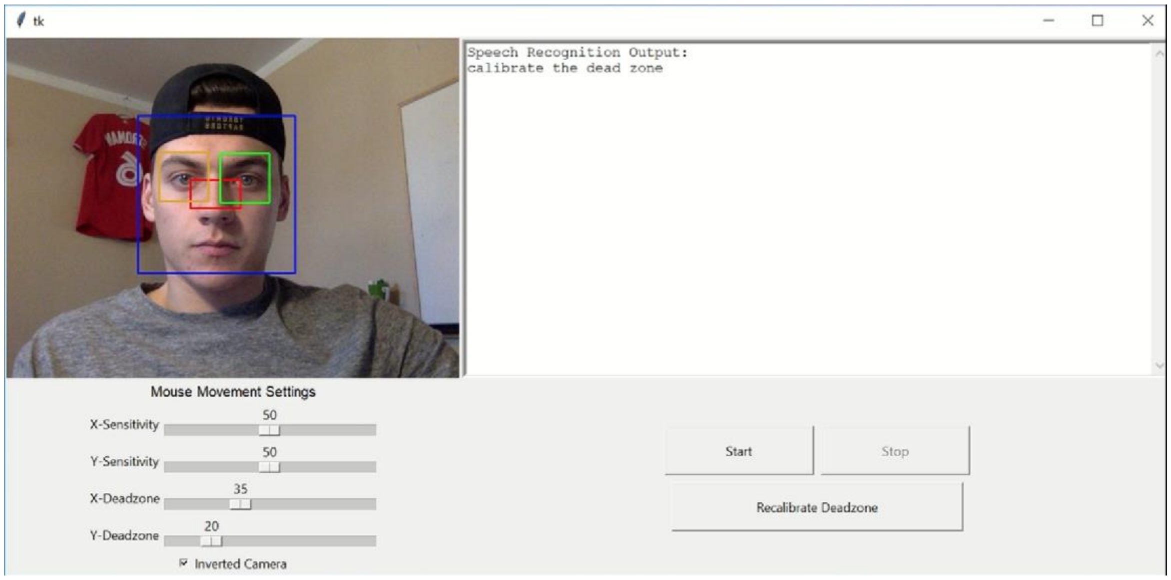
Property	**Description**
COMMANDS	A list holding the possible commands the user can give
SOFTWARE_DICT	A dictionary holding all installed applications. The keys are the program names, and the items are the programs’ path.
KEY_LIST	A list of all keyboard keys
open_applications	A dictionary of currently opened applications. The keys are the program names, and the items are the reference to the program’s process.

Table 10: Handler object's methods.

Method	Description
execute(self, cmd, prefix, modifiers)	Takes in the [command, prefix, [modifiers]] data format and executes the proper function.
click(button='left', clicks=1)	Clicks the designated button the specified number of times.
mouseDown(self, button='left')	Press the designated mouse button down.
mouseUp(self, button='left')	Releases the designated mouse button.
type_string(self, string, interval=0)	Inputs the designated string to the OS as a keyboard input.
press_key(self, key, presses=1)	Presses the designated key the specified number of times.
press_keys(self, keys, presses=1)	Pushes down the specified keys in order, then releases them. This repeats the specified number of times.
keyDown(self, key)	Presses the designated key down without releasing.
keyUp(self, key)	Releases the designated key.
open_application(self, path, name='unknown')	Opens the specified application and saves a reference to the process in the open_applications dictionary with 'name' as the key.
close_application(self, name)	Closes the specified application by accessing 'name' in the open_applications dictionary and terminating the process.

2.3.4 GUI Handler Module

The GUI Handler module, called guihandler.py, contains an object called App. This object is a subclass of threading so that the GUI can run in the background. On initialization, the App takes reference to an instance of the Listener and an instance of the Watcher. This allows the GUI to access the properties of the Watcher and Listener such that it can change their properties or read any outputs or images from the objects. The GUI is built using the *tkinter* library, which comes preinstalled with python. A screenshot of the GUI is shown in Figure 1. The image in the top left of the GUI is taken from the img_draw property of the Watcher. The sliders on the bottom left of the GUI change the properties of the watcher in order to adjust the sensitivity or dead zone size. The check box at the bottom left will change whether the camera image gets mirrored. The space in the top right is designated for print outputs from the Listener. The buttons on the bottom right of the GUI will start/stop the facial mouse control and recalibrate the dead zone. Additionally, all settings and buttons can be controlled using voice commands.



```
Speech Recognition Output:
calibrate the dead zone
```

X-Sensitivity 50
Y-Sensitivity 50
X-Deadzone 35
Y-Deadzone 20
☑ Inverted Camera

Figure 1: FacialPC's GUI.

Every 100ms, the App does the following:

1. Check the print_output property of the Listener. If there is something to print, add it to the text box in the top right of the GUI.
2. Update the image in the top left of the GUI from the img_draw property of the watcher.
3. Check the cmds_output property of the Listener. If there is a command waiting, adjust the appropriate settings.

After creating an instance of the App, the GUI will open and begin updating immediately.

2.3.5 The Main Program

Due to the object-oriented-programming approach taken, the main program was kept to 28 lines of code. It is shown in Algorithm 2.

Algorithm 2: Main program.

```
from facialpc import speechrecog
from facialpc import facialrecog
from facialpc import guihandler

if __name__ == '__main__':
    listener = speechrecog.Listener()
    watcher = facialrecog.Watcher()
    app = guihandler.App(listener, watcher)

    while watcher.limit_x_left == 0:
        watcher.detect_face()
        watcher.calibrate()

    while watcher.prev_eyes[0] == 1 or watcher.prev_eyes[1] == 1:
        watcher.detect_face()
        watcher.calibrate()
        watcher.detect_eyes()

    while True:
        # Read the frame
        watcher.detect_face()
        if (len(watcher.face) > 0):
            if (watcher.start):
                watcher.move_mouse()
            watcher.detect_eyes()
            watcher.check_eyes()
            if watcher.start:
                watcher.click_mouse()

        if not app.isAlive():
            break

    # Release the VideoCapture object
    watcher.cap.release()
    # Stop the listener
    listener.stop_listening()
```

2.4 Testing and Product Evaluation

The testing phase of FacialPC consisted of many iterations. Each component of the design was tested individually upon completion and the final build of the program was tested after all the components were combined.

The face detection algorithm was tested for general accuracy by running the algorithm on a set of pictures with varying numbers of faces in the image. The number of faces expected to be detected was compared to the number of faces detected in the image. The test results showed that the algorithm had an 87% accuracy for this test. Next, the new and improved algorithm including the filtering functions was tested by having a single tester sitting in front of a webcam. The tester moved and tilted their head in different directions. The total number of frames where a face was detected, was compared to the total number frames in the test session. The test results showed that the algorithm had a face detection accuracy of 98%.

The eye detection algorithm was tested in the same format as the face detection algorithm. First, many images with many faces were run through the eye detection algorithm. The expected output of detected eyes was compared to the number of eyes detected in the image. The accuracy of the eye detection algorithm was 72%. Next, the improved eye detection algorithm with the filtration functions was tested by having a single tester in front of the webcam. The tester moved and tilted their head and opened and closed their eyes. The total number of frames where both eyes were detected was compared to the total number of frames in the test session. The algorithm was calculated to have an eye detection accuracy of 95%.

The eye classification model was tested and evaluated after the training of the model by running the model on the test set. The accuracy of the classifier was assessed to be 95.36%. The classifier was then tested with the input from the eye detection algorithm. The program was modified to output the “eye stream”. The eye stream output the states of both eyes for each frame. This allowed the team to count the number of false positives and false negatives produced by the classifier over a fixed period. The classifier was found to correctly classify the state of the eye 96% of the time. Next, the clicking function was tested. The test utilized the modified program that outputs the eye stream. The tester tested the clicking functions by closing their eyes for different periods of time. The number of clicks and when the clicks occurred was recorded and compared to the expected output of the program. The accuracy of the clicking function was assessed to be 92%. The clicking function was also tested for response time. The program was modified to output the latency between the first frame where the eye is detected as closed, and the click commands execution. The total time between the eye being closed and the click command was 157ms. Considering that the webcam runs at 60 frames per second, each frame takes 1/60 = 16.67ms. The eye must be detected as closed for 8 frames in a row, which is 16.67x8 = 133.36ms. 157 - 133.36 = 23.64ms or 1.42 frames of delay. Therefore, the latency between eye detection and command execution is 23.64ms.

The mouse movement function was tested by having a tester move their face around the screen. It was assessed, that as long as a face was detected, the dead zone function could move the mouse in the correct direction 100% of the time. This means that the effective accuracy of the mouse movement function is the same as the accuracy of the face detection algorithm, as it is completely reliant on the detection of the face. Therefore, the accuracy of the mouse movement function is 98%. The mouse movement function was also tested for response time. The program was modified to track the latency between the center point leaving the dead zone and the mouse moving. The latency recorded was 70ms.

The voice recognition was tested for both accuracy and latency. The accuracy was assessed for both hot-word detection, and command decoding. The hot-word detection accuracy was assessed by having different testers say the hot word “Hey Computer” to the program. The results were recorded, and it was found that the program successfully identified the keyword 92% of the time. The same test process was used for the command decoding, except many different commands were given to the program by different testers. The accuracy of the command decoding test was calculated to be 95%. The response time was tested to observe the latency between entering a command and having the command executed. The latency for the command execution was calculated to be 18ms.

After all individual components were tested, they were integrated together into the final build of the program. This final build was thoroughly tested by the design team for catastrophic errors that could lead to a critical program failure. All three members of the design team used the program for long periods of time, and stress tested the program by applying as many input scenarios as possible to the program. This approach to testing allowed the design team to identify many error points in the software, and error checking and handling was added where necessary.

When evaluating the final design against the target system specifications, the design meets almost all the proposed goals. The user can move the mouse and left or right click anywhere on the screen using head movement and eye winks. The double click and click and hold functions were not implemented with double eye winks and holding the eye shut as originally planned but were instead implemented using voice commands. This decision was made as the voice recognition was a more reliable option. Many error checks were implemented for when the user turns away from the screen, closes an eye for a long period of time or blinks. The user could also use voice recognition to simulate the operation of the keyboard and enter full sentences and commands to the system. While the on-screen keyboard was not directly integrated into the GUI as originally planned, the onscreen keyboard can still be opened using a voice recognition command. The program can be both opened and closed using a voice command as listed in the system specifications. The interface allows for the users to activate and deactivate the software, change both the mouse sensitivity and dead zone dimensions, and observe the detection of their facial features. As mentioned earlier the on-screen keyboard function was changed from being on the interface to being openable with a voice command. The program meets all the performance requirements set out in the system specifications. The mouse can be seamlessly operated, and the user can click anywhere on the screen. The numerical test results provided earlier in the section meet all the performance requirements for eye classification accuracy, head position accuracy and voice recognition accuracy. Table 11 summarizes the team's evaluation of the project goals.

Table 11: An evaluation of the project's goals.

1	Functional Goals	How it was Achieved
1.1	Move cursor to anywhere on screen using only head movement: • Tilt head left/right to move cursor left/right • Tilt head up/down to move cursor up/down With error checking for: • User turning away from screen	The goal of moving the cursor anywhere on screen was achieved using the center of the face. If the center of the face was within a dead zone, the mouse did not move. Otherwise, the mouse cursor would move in the direction of the head. The if the program loses track of the face, no mouse movement occurs.

1.2	Being able to click anywhere on screen: • Left click by closing only your left eye for designated amount of time then opening • Left click and hold by keeping left eye closed • Double left clicks by winking your left eye twice in succession • Right click by closing only your right eye for designated amount of time then opening With error checking for: • Closing right eye for too long does not cause multiple clicks • Blinking does not cause any clicks on the screen	Being able to click anywhere on screen: • Left click by holding the left eye closed for 8 frames • Right click by holding the right eye closed for 8 frames • Double click using voice commands Error checking: • Keeping either eye closed will not cause multiple clicks because the frame counter only resets once the eyes open • Blinking does not cause any clicks because the eye must be closed for 8 frames
1.5	Operate the keyboard hands free using voice recognition software: • Audio cue to activate the voice recognition for typing (“Hey Computer”) • Detection of when user is finished talking • Input the spoken string as a keyboard input	Operating keyboard hands free using voice recognition: • Program waits for “Hey Computer” audio cue • Program waits for voice input from user • Program decodes the sting and executes functions to accomplish what the user wants to do. This includes typing a string.
1.6	Operate the keyboard hands free through use of an on-screen keyboard	An on-screen keyboard can be brought up with the voice command “Open the on-screen keyboard”
1.7	Voice recognition audio cue in order to active/deactivate the software • “Hey Computer, disable the program” for example	The mouse movement and clicking can be enabled or disabled with the voice command: “start the program” or “stop the program”.
2	**Interface requirements**	
2.1	Seamless interface on side of screen which includes:	The interface is not transparent as the team wanted it to be. There is also not a button to bring up the on-screen keyboard. However, there is the ability to adjust sensitivity and dead zone size.
2.2	• Activate/deactivate software	
2.3	• Adjust cursor sensitivity	
2.4	• Bring up on screen keyboard	
3	**Performance requirements**	
3.1	Seamless operation of cursor, allowing for a person to operate the computer without a mouse	The mouse operation has a very low latency of 70ms. As a result, the design team feels this performance requirement is met.
3.2	Being able to left-click and right-click anything on the screen	The mouse movement and clicking functions simulate regular mouse movement. This includes the bounds of mouse movement.
3.3	93% accuracy in classification of whether each eye is open or closed	Tests showed an eye classification accuracy of 96%.
3.4	93% accuracy in classification of head position (tilted left/right/up/down)	Tests showed a 98% accuracy for face detection and mouse movement.

3.5	Voice recognition recognizes audio cue ("Hey Computer") 90% of the time	Tests showed a 92% accuracy for keyword detection.
3.6	Voice recognition will successfully identify 95% of words in a string when used for keyboard input	Tests showed a 95% accuracy for string detection, encoding and decoding.

3 Future Work and Conclusions

This project allows the physically disabled to increase their skillset in today's job market by providing a solution that is affordable, non-invasive and easily accessible for its users. As society becomes more technology driven, everyday activities will become dependent on computers. For example, in store shopping is gradually being replaced with online shopping. Webservices such as Amazon require the use of a computer in order to shop more conveniently. Enabling computer use to the physically disabled will bring an overall better quality of life. Allowing the physically enabled to use commodity machines will also have a positive economic impact on society. With 63% of all jobs in the United States requiring use of a mouse and keyboard, allowing the physically disabled to be able to operate a mouse and keyboard will increase their employment in these markets.

If this project were to be scaled to a commercial version, it would have updates which would allow for better usability as more development is done. Users would purchase a license to download this program with an account. A key encryption would be given in order to activate the account. A subscription model would be required because the program uses *Google Cloud API* which is paid for based on the number of accesses per month.

3.1 Next Steps

3.1.1 Facial Recognition

For the next stages of the project, more accurate and quicker facial recognition could be implemented. More accurate eye classification models could be built which would allow a reduction in the number of consecutive frames of a closed eye required for a click.

A calibration processes could be implemented where each user can run a script which takes 1200 images of their closed and open eyes. These images would be used to train and test the model that would be tailored to each user. Since each model is only trained on a single user, the model will have increased accuracy.

Further exploration of feature detection should be conducted in order to map double clicks and click and drag.

More research should be conducted in order to explore complex options for mouse movement such as non-invasive eye tracking.

3.1.2 Speech Recognition

External noise can cause increased latency when the voice recognition software doesn't detect if the user is finished speaking. The program currently adjusts the noise floor every 10 seconds when there is not noise input from the user. By implementing an adaptive noise filter, it would improve performance on the voice inputs.

3.1.3 GUI Development

The ideal GUI for the program would consist of a small (~20 by 100 pixels) rectangle at the top right of the screen. When the mouse is hovered over this rectangle, the GUI automatically drops down. This would allow the user to easily access the GUI at any point while using the program.

3.2 Other Existing Solutions

Previous solutions such as the eagle eye were unsuccessful due to the need for electrodes in order to map eye movement to mouse movements [14]. The improved Camera mouse was implemented by selecting a distinct feature on the face and tracking that feature's movement [15]. Clicking can be done by making the mouse pointer dwell over a spot on the screen [14]. Camera Mouse works best with application programs that require only a mouse and a left click and that do not have tiny targets [15]. It's easier to use Camera Mouse with application programs that do not require extreme accuracy [15].

The program was developed at Boston College to help people with disabilities use the computer. The main audience for this program is people who do not have reliable control of a hand but who can move their head. People with Cerebral Palsy, Spinal Muscular Atrophy, ALS, Multiple Sclerosis, Traumatic Brain Injury, various neurological disorders use this program and its predecessors to run all types of computer software [15].

FacialPC performs clicking and mouse movements significantly better than the Camera Mouse. Camera Mouse performs clicks by dwelling over a spot on the screen [15], which gives the user less control over mouse clicks. This method of clicking also makes Camera Mouse prone to false positives on clicks. FacialPC gives the user a specific method of clicking, that allows the user to have more control over when they click.

Another weakness of Camera Mouse is it does not give the user the option of right clicking. One of the fundamental functional requirements of a mouse is the ability to left and right click. Without this functionality the user will be limited to their ability to effectively use a computer. FacialPC can right click with a right eye wink.

The mouse movement of Camera Mouse is also weak when it comes to precision. By tracking features and moving the mouse accordingly, spastic movements of the face can cause rapid relocations of the mouse. The user would have to keep their head still and precisely move their head in order to click with high accuracy. This isn't realistic because micromovements of the head is normal and any restrictions on that would involve keeping the user's head still for a long duration of time. This would cause discomfort and fatigue in the neck. FacialPC tackles this problem by implementing a dead zone where movements of the head do not affect the movements of the mouse. The mouse moves proportionally to the distance of the tracked feature and the dead zone. If the head is just outside the dead zone the mouse moves slowly in the direction of the head. This allows the user to adjust their head proportionally to have precise control over the movements of the mouse. This method also reduces fatigue by allowing the user to adjust the size of the dead zone depending on the activity and needs of the user.

4 Self-Analysis

In Section 2 this report described the design approach taken by the team. Table 11 shows which goals the team did and did not meet. Although the team finished a large percentage of the goals, and the program could do everything the team wanted it to do, there were still some short comings. The team wanted the program to enable the user to double-click and click-and-drag using facial features, however it was only implemented using voice commands. The GUI did not turn out exactly how the team envisioned. The team wanted a GUI that would overlay all windows with automatic minimization and maximization when the mouse is not or is hovered over it respectively.

Table 12 shows the team's project plan from September 2020. Although deadlines very often got pushed back, the team did use the plan throughout the project. It provided a good way of working towards our final goals. There were, however, some milestones that were not met. Testing on a 3rd party user was not accomplished as a result of isolation. Additionally, testing on different quality webcams was not completed. The team only used the three webcams on each of their laptops.

Table 12: Project plan from September 2020.

No.	Milestones	Due date	Responsible members
1	Set up python environment to run all packages discussed in Section 3.2	Week 2	All members responsible for own system
2	Gain knowledge of Speech Recognition package to recognize users' voice and share with team	Week 4	Brandon
3	Gain knowledge of face detection functions within OpenCV and share with team	Week 4	Chris
4	Gain knowledge of pywinauto package to control the mouse and keyboard and share with team	Week 4	Alan
5	Test OpenCV functions for detecting and distinguishing between facial features	Week 5	Chris
6	Start developing neural network for facial recognition to detect different head positions with sets provided by team	Week 6	Chris
7	Develop and compile training and test sets for the neural network	Week 5	Alan
8	Help develop neural network and framework	Week 6	Alan
9	Start developing framework for the program	Week 5	Brandon
10	Finish developing framework for the program	Week 6	Brandon

11	Major mid Semester meeting to discuss the position of the project and what issues will arise moving forward	Week 7	Brandon Send Zoom Link
12	Finish training and start testing Neural Network on testing sets	Week 7	Chris
13	Link left eye wink with left click, double left wink with double click, and left wink and hold with left click and hold	Week 7	Alan
14	Finish and test left click functionalities	Week 8	Alan
15	Link right eye wink with right click and test functionality	Week 9	Alan
16	Implement moving mouse with head movement detection	Week 7	Brandon
17	Test mouse movement functionality	Week 8	Brandon
18	Add functions for changing mouse sensitivity	Week 9	Brandon
19	Implement and test an on-screen keyboard using pywinauto	Week 8	Chris
20	Test keyboard functionality with on screen mouse clicks	Week 9	Chris
21	Start to implement voice detection	Week 10	Brandon
22	Finish implementing voice detection	Week 11	Brandon
23	Implement on screen interface	Week 10	Chris
24	Test on screen interface	Week 11	Chris
25	Implement and test enable/disable software feature with both the interface and voice recognition	Week 12	Alan
26	Begin testing general operation of program (browsing internet)	Week 12	Chris
27	Begin 3rd party testing	Week 12	Brandon
28	Test operation of program with lower quality webcam	Week 13	Chris
29	Test operation of program with laptop webcam	Week 13	Brandon
30	Bug fixes found during testing	Week 13-16	Alan
31	Prepare for open house show case	Week 16	Alan
32	Final Deliverable	Week 23	Alan
33	Project report & presentation	Week 24	All Members

Overall, the team was very happy with the progress made with the project in the 24-week development. The main goal – operating a PC without the use of hands – was met in an effective way. From the results in Section 2.4, it is evident that the team did an exceptional job in meeting the project goals.

5 References

[1] U.S. Bureau of Labor Statistics, "OCCUPATIONAL REQUIREMENTS IN THE UNITED STATES – 2018." U.S. Department of Labor, 2019.

[2] M. Craig, W. Hill, K. Englehart, and A. Adisesh, "Return to work after occupational injury and upper limb amputation," *Occupational Medicine*, vol. 67, no. 3, pp. 227–229, Apr. 2017, doi: 10.1093/occmed/kqx012.

[3] A. Young *et al.*, "Which patients stop working because of rheumatoid arthritis? Results of five years' follow up in 732 patients from the Early RA Study (ERAS)," *Ann Rheum Dis*, vol. 61, no. 4, pp. 335–340, Apr. 2002, doi: 10.1136/ard.61.4.335.

[4] 1615 L. St NW, Suite 800Washington, and D. 20036USA202-419-4300 | M.-857-8562 | F.-419-4372 | M. Inquiries, "Demographics of Social Media Users and Adoption in the United States," *Pew Research Center: Internet, Science & Tech*. https://www.pewresearch.org/internet/fact-sheet/social-media/ (accessed Mar. 25, 2020).

[5] 1615 L. St NW, Suite 800Washington, and D. 20036USA202-419-4300 | M.-857-8562 | F.-419-4372 | M. Inquiries, "Disabled Americans less likely to use technology," *Pew Research Center*. https://www.pewresearch.org/fact-tank/2017/04/07/disabled-americans-are-less-likely-to-use-technology/ (accessed Mar. 25, 2020).

[6] ISHN, "Statistics on hand and arm loss," Jan. 27, 2014. https://www.ishn.com/articles/97844-statistics-on-hand-and-arm-loss (accessed Mar. 26, 2020).

[7] C. S. Crowson *et al.*, "The Lifetime Risk of Adult-Onset Rheumatoid Arthritis and Other Inflammatory Autoimmune Rheumatic Diseases," *Arthritis Rheum*, vol. 63, no. 3, pp. 633–639, Mar. 2011, doi: 10.1002/art.30155.

[8] Arthritis Foundation, "Osteoarthritis of the Hands." https://www.arthritis.org/diseases/more-about/osteoarthritis-of-the-hands (accessed Mar. 27, 2020).

[9] U.S. Bureau of Labor Statistics, "Labor Force Statistics from the Current Population Survey." https://www.bls.gov/cps/ (accessed Mar. 25, 2020).

[10] "Computer vision syndrome (Digital eye strain)," *American Optometric Association*. https://www.aoa.org/healthy-eyes/eye-and-vision-conditions/computer-vision-syndrome?sso=y (accessed Mar. 29, 2020).

[11] "Face Detection using Haar Cascade Classfiers," *Bogo to Bogo*. https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php (accessed Apr. 12, 2021).

[12] S. Goled, "Top 5 Neural Network Models For Deep Learning & Their Applications," *Analytics India Magazine*, Oct. 25, 2020. https://analyticsindiamag.com/top-5-neural-network-models-for-deep-learning-their-applications/ (accessed Apr. 12, 2021).

[13] A. Zhang (Uberi), *SpeechRecognition: Library for performing speech recognition, with support for several engines and APIs, online and offline.* .

[14] "Eagle Eyes." https://www.bc.edu/bc-web/schools/carroll-school/sites/eagle-eyes.html/ (accessed Apr. 12, 2021).

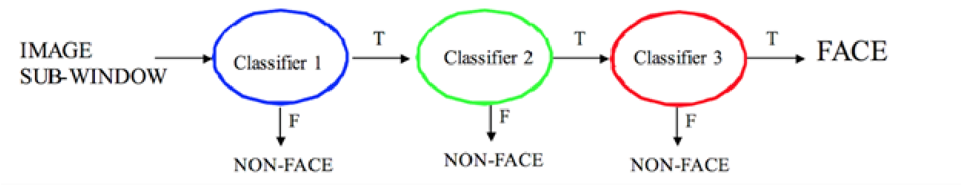
[15] "Camera Mouse." https://eyecomtec.com/3101-Camera-Mouse (accessed Apr. 12, 2021).

[16] C. Zhan, W. Li, P. Ogunbona, and F. Safaei, "Real-Time Facial Feature Point Extraction," in *Advances in Multimedia Information Processing – PCM 2007*, vol. 4810, H. H.-S. Ip, O. C. Au, H. Leung, M.-T. Sun, W.-Y. Ma, and S.-M. Hu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 88–97.

[17] E. Mäkinen, "Introduction to Computer Vision from Automatic Face Analysis Viewpoint," p. 28.

Appendix I – Cascaded Face Detection Algorithm and Haar Cascades

The Viola and Jones cascaded face detector algorithm is constructed of a cascade of classifier layers. Each layer has a set of classifiers trained with the Adaboost algorithm. The detection search starts from the top left corner to the bottom right, looking for a face. The image where faces are detected is searched through numerous times with a different sub image size each time. When the image is scanned through, the sub images are passed to the first layer of the cascade that contains a set of Haar-like features to determine whether the sub-image contains a face or not [16].



If the first layer classifies the sub image as a face, then it is passed on to the next layer. This is repeated until the sub image has passed through all the layers and is classified as a face or is discarded in a layer. Each layer is an Adaboost classifier that makes the detection decision based on the set of Haar-like features on that layer [16]. The main benefit of cascades is that it easily weeds out recognizable non face sub images in the lower layers that have fewer features compared to the later layers which have more features. This cascaded structure has a high efficiency because it takes less time to process sub images in the earlier stages compared to the later ones. While most of the non-face sub images are rejected in the earlier stages, sub images that will contain a face will be isolated faster.

The above approach has a low detection rate in identification of specific features, such as eyes nose mouth, in lower-level resolution. This is due to the lack of significant structural information of facial components which become less detectable as resolutions become lower [17]. To improve detection extended Haar-like feature set is used in the facial feature detection.

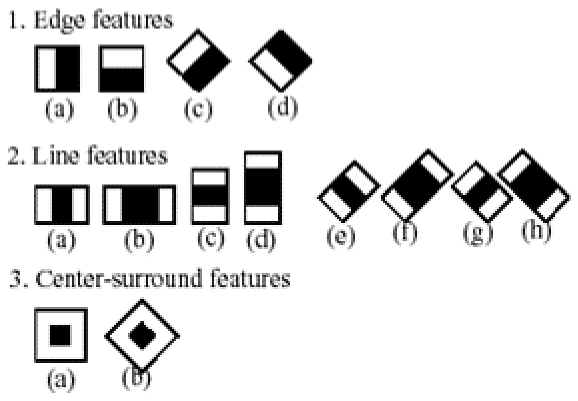


Figure 2: Haar-like features.

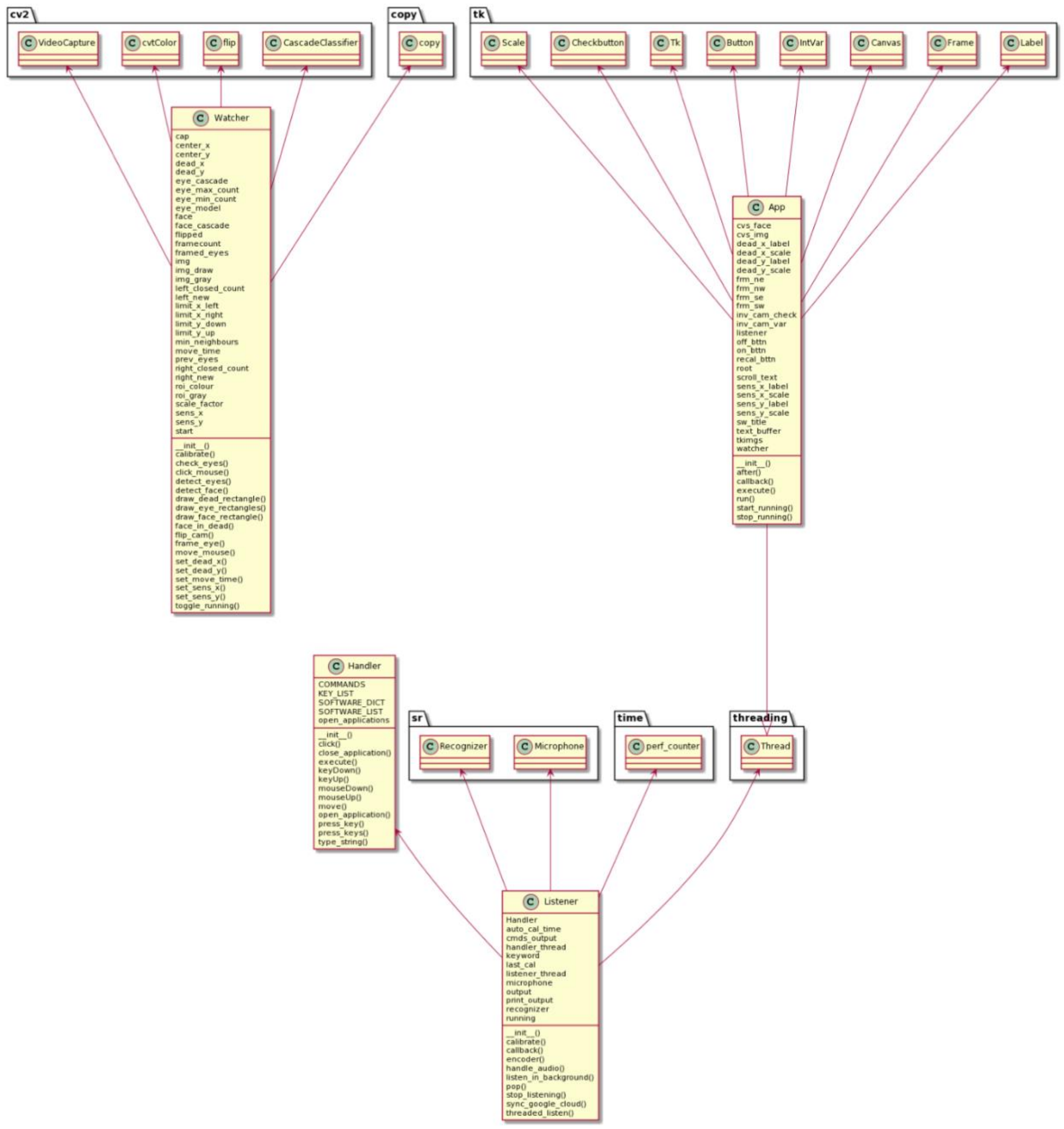
Appendix II – Neural Network Training Results

```
Model: "sequential"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
flatten (Flatten) (None, 4761)0
_________________________________________________________________
dense (Dense) (None, 128) 609536
_________________________________________________________________
dense_1 (Dense) (None, 128) 16512
 _________________________________________________________________
dense_2 (Dense) (None, 2) 258
=================================================================
Total params: 626,306
Trainable params: 626,306
 Non-trainable params: 0

Model: "CNN"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d (Conv2D) (None, 67, 67, 32) 320
_________________________________________________________________
 max_pooling2d (MaxPooling2D) (None, 33, 33, 32) 0
_________________________________________________________________
conv2d_1 (Conv2D) (None, 31, 31, 64) 18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 64) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 13, 13, 64) 36928
_________________________________________________________________
flatten (Flatten) (None, 10816) 0
_________________________________________________________________
dense (Dense) (None, 64) 692288
_________________________________________________________________
dense_1 (Dense) (None, 256) 16640
_________________________________________________________________
dense_2 (Dense) (None, 256) 65792
_________________________________________________________________
dense_3 (Dense) (None, 128) 32896
_________________________________________________________________
dense_4 (Dense) (None, 14) 1806
==================================================================
Total params: 865,166
Trainable params: 865,166
 Non-trainable params: 0
```

Appendix III – UML Diagram of Program

Below is a UML diagram of all classes involved in the program.


cv2
VideoCapture
cvtColor
flip
CascadeClassifier
copy
copy
tk
Scale
Checkbutton
Tk
Button
IntVar
Canvas
Frame
Label
Watcher
cap
center_x
center_y
dead_x
dead_y
eye_cascade
eye_max_count
eye_min_count
eye_model
face
face_cascade
flipped
framecount
framed_eyes
img
img_draw
img_gray
left_closed_count
left_new
limit_x_left
limit_x_right
limit_y_down
limit_y_up
min_neighbours
move_time
prev_eyes
right_closed_count
right_new
roi_colour
roi_gray
scale_factor
sens_x
sens_y
start
__init__()
calibrate()
check_eyes()
click_mouse()
detect_eyes()
detect_face()
draw_dead_rectangle()
draw_eye_rectangles()
draw_face_rectangle()
face_in_dead()
flip_cam()
frame_eye()
move_mouse()
set_dead_x()
set_dead_y()
set_move_time()
set_sens_x()
set_sens_y()
toggle_running()
App
cvs_face
cvs_img
dead_x_label
dead_x_scale
dead_y_label
dead_y_scale
frm_ne
frm_nw
frm_se
frm_sw
inv_cam_check
inv_cam_var
listener
off_bttn
on_bttn
recal_bttn
root
scroll_text
sens_x_label
sens_x_scale
sens_y_label
sens_y_scale
sw_title
text_buffer
tkimgs
watcher
__init__()
after()
callback()
execute()
run()
start_running()
stop_running()
Handler
COMMANDS
KEY_LIST
SOFTWARE_DICT
SOFTWARE_LIST
open_applications
__init__()
click()
close_application()
execute()
keyDown()
keyUp()
mouseDown()
mouseUp()
move()
open_application()
press_key()
press_keys()
type_string()
sr
Recognizer
Microphone
time
perf_counter
threading
Thread
Listener
Handler
auto_cal_time
cmds_output
handler_thread
keyword
last_cal
listener_thread
microphone
output
print_output
recognizer
running
__init__()
calibrate()
callback()
encoder()
handle_audio()
listen_in_background()
pop()
stop_listening()
sync_google_cloud()
threaded_listen()