Spring 2018

Improved Hands-Free Text Entry System

Gaurav Gupta
San Jose State University

Improved Hands-Free Text Entry System

A Project Presented to

The Faculty of Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements of Degree

Master of Science

By

Gaurav Gupta

May 2018

The Designated Project Committee Approves the Master's Project Titled

Improved Hands-Free Text Entry System

By

Gaurav Gupta

APPROVED FOR DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2018

ABSTRACT

Improved Hands-Free Text Entry System

By Gaurav Gupta

An input device is a hardware device which is used to send input data to a computer or which is used to control and interact with a computer system. Contemporary input mechanisms can be categorized by the input medium: Keyboards and mice are hand-operated, Siri and Alexa are voice-based, etc. The objective of this project was to come up with a head movement based input system that improves upon earlier such systems. Input entry based on head movements may help people with disabilities to interact with computers more easily. The system developed provides the flexibility to capture rigid and non- rigid motions. Unlike prior work, the organization of alphabet symbols in our design is based on the frequency of the characters in the English dictionary. We conducted experiments on our system and compared it to previous head movement systems.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my project advisor, Prof. Chris Pollett, for his guidance, motivation and immense knowledge during this project. Without his precious support, this project would not have been possible. I want to take this opportunity and extend my appreciation to everyone who helped me in this project secondarily.

I sincerely thank my committee members, Prof. Katerina Potika and Mr. Kevin Smith, for their valuable suggestions and encouragement. Thank you all for the support.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

An input device is a hardware device which is used to send input data to the computer. The contemporary input mechanism is not restricted to keyboard and mouse, there are several other ways depending on the medium the input can be provided, for example, audio-based input, video-based input, input in the form of images. Examples of such media are Siri or Alexa, touchscreens. The objective of this project is to come up with a solution that provides an input entry mechanism based on head movements. Input entry based on head movements could help people with disabilities to interact with computing devices easily.

Head movements can be continuously tracked with the help of images taken from a camera mounted on a computing device. Nowosielski [1] [2] suggests methods to generate text from head movements. In [1], an approach is given such that every alphabet symbol can be reached with the help of at most three head movements. Nowosielski mentions that there are four directions in which a person could move his head, so moving the head three times gives the flexibility to cover sixty-four possibilities. These possibilities are then mapped to alphabet symbols, digits, punctuation, special symbols, and backspace. In [2], its shown how to reach any particular alphabet symbol using two head movements. In order for the system to work, alphabet symbols are paired. This system also provided support for English word completion. The words are suggested based on the proximity to a dictionary word and the pairs of alphabet symbol selected.

In this project, the objective is to provide a novel method to generate text-based characters with the help of head movements. Our approach is different from the solutions proposed by Nowosielski [1] [2]. We focus on the frequency of alphabet symbols in an English dictionary. Based on the alphabet symbol frequency, placement of the alphabet symbol has been decided. Placement of the alphabet symbol makes the access to certain letters in two head movements whereas the symbols with the lower frequency counts can be reached in three head movements. The project captures non-rigid motions to support shortcuts to different actions of our system. The project also supports word guessing based on word completion. The number of words suggested is limited to four, one for each head direction.

The organization of this report is as follows: The background chapter provides information on techniques used by Nowosielski [1] [2], face recognition strategies, Tensorflow API, and Atom editor. This is followed by design and architecture chapter to provide a clear illustration of the proposed design for the implementation. The implementation chapter describes in detail the sub-components that are implemented as a part of this project. The experiments chapter explains the unit test-cases covered for the individual components and what was the observation based on these unit test-cases. Finally, conclusion and future work chapter concludes the project and shows the possibilities for future work.

2. BACKGROUND

In this chapter, we discuss the prior work implemented by Nowosielski [2]. We also discuss what are the possible strategies to detect a face in an image since to recognize head movements it is required to detect the face of the primary person (person of interest). We also look into the Atom editor. Atom is a configurable, and hackable editor with the flexibility to integrate pre-built extensions or to write one from scratch. Learning about Atom editor is important because we have chosen the Atom editor platform to display the text input by the head motions. It is used in the project to display the text generated as outputs.

2.1. <u>The Implementation of Nowosielski [1] [2]</u>

Nowosielski proposed two novel methods described in [1] and [2] for generating text characters with the help of head movements. He divides that motions into two groups: rigid motions, and non-rigid motions. Rigid motion captures the translational or rotational movements of the head. Non-rigid motion captures motions like mouth movements (opening, closing or stretching), cheeks twitch, or eye winks. Nowosielski captures rigid motions of head movements in [1] and [2]. Nowosielski captures translational motion for each of the four directions (up, down, left, and right). In other words, Nowosielski captured motion in the vertical and horizontal direction.

Nowosielski [1] quotes that head movement might be the only source of interaction with computers for people suffering from certain disabilities. Nowosielski suggests two techniques to generate text characters with the help of head movements.

2.1.1. Three Steps Keyboard [2]

To track head movements, Nowosielski [2] uses rigid motions only. The motion is constrained to four directions, thereby limiting the number of choices a user can make to four. It requires more movements to cover all the possible alphabet symbols, numbers, punctuation, and special symbols. To solve this issue, Nowosielski suggests that if a user moves the head in one of the direction, four more choices are further provided to select a character. This increases the total number of possibilities from four to sixteen since in each of the four directions there are four more choices. If flexibility to select more choices is repeated one more time (i.e. three times in total), the number of choices will increase from sixteen to sixty-four. Sixty-four choices would be sufficient to cover the required number of alphabet symbols, numbers, and other punctuations. Fig.1 shows the layout of the arrangement of the keys in the keyboard used by Nowosielski.

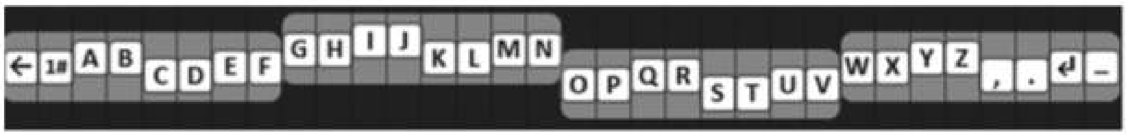


Fig. 1. The Layout of the reduced interaction 3 steps keyboard [2]

Nowosielski arranges the letters in four main groups. The groups on the extreme left and extreme right are on the same level. If the head is moved in the left or right directions, then the group of the same level in that direction gets selected. There are two more groups in the middle, one in an upward direction and other in downward direction. Moving the head in one of the two directions will select the character choices in that direction. After moving the head in one of the directions of the main groups, other possible four choices to select are available. Further moving head towards in one of directions will select the characters in that direction. At the lowest level, there are two choices available, one in the left direction and other in right direction. The character on the leftmost extreme provides a backspace capability and in the rightmost extreme provides a space capability. To select digits and other symbols "*1#*" could be selected.

A simple example is to select a character 'A' using three steps keyboard is illustrated in Fig. 2. From Fig. 2., the process of selecting the groups of characters can be easily understood. If the user moves the head in the left direction, then the extreme left group from the main group get selected. The selected group gets highlighted and rest of the choices gets disabled. Further moving the head in the upward direction selects the characters available in that direction. For this case, the available characters to select would be *'A'* or *'B'*. Finally, when the user moves in the left direction the character *'A'* get selected. If the user moves in the right direction instead of the left direction in the third step, the character *'B'* would have been selected. A clear illustration of the whole process is illustrated in Fig. 2.

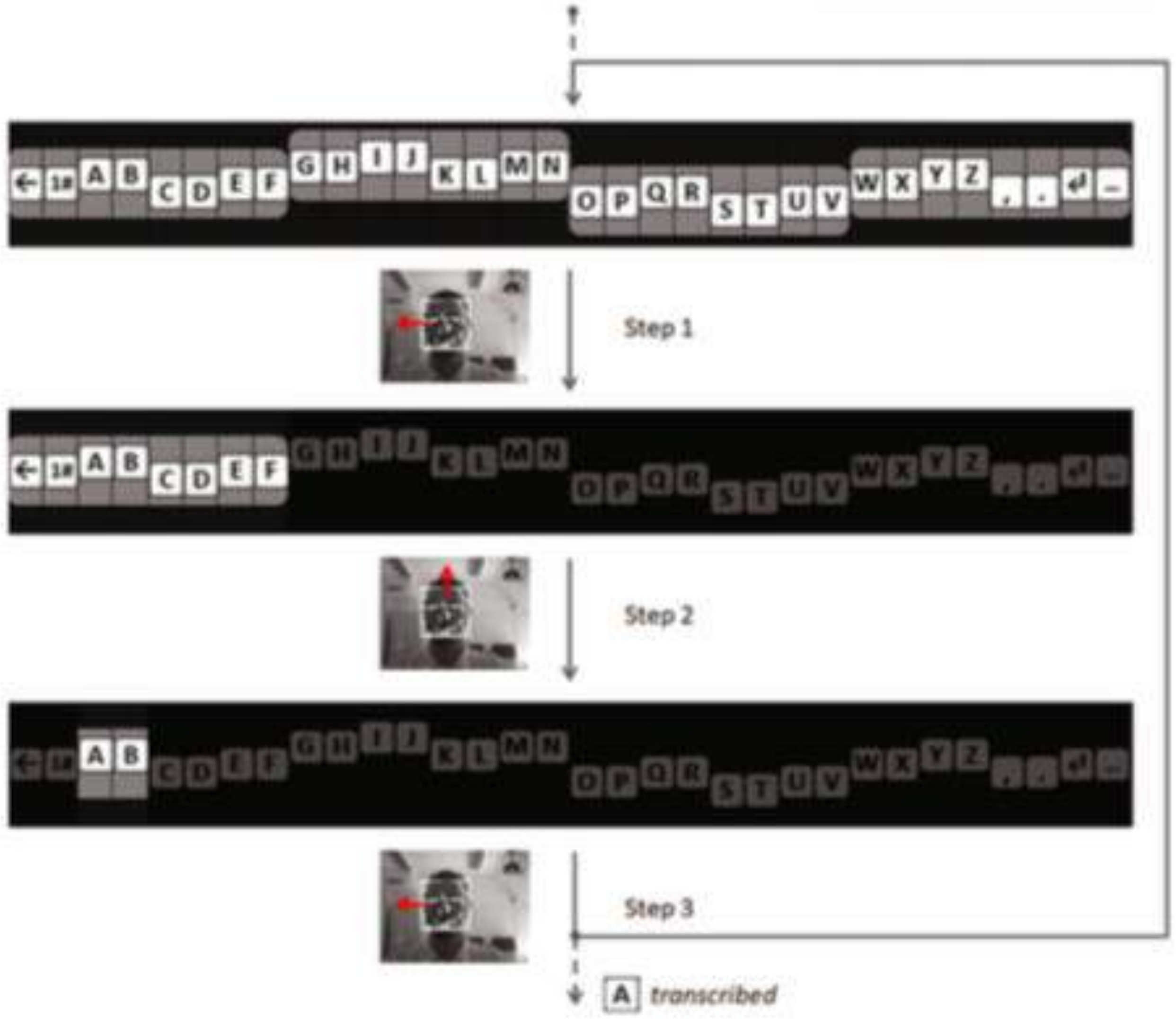


Fig. 2. Example of selecting character ‘A’ using 3-Steps Keyboard

2.1.2. Two Letters Key Keyboard [1]

The architectural design of the two letters keyboard resembles that of a 3-Step keyboard [2], but with major improvements. The layout of the two letters key is shown in Fig. 3. In this design architecture, two letters from alphabets are grouped together is also shown in Fig. 3. Two letter key keyboard also supported backward compatibility to the 3-Step keyboard. In order to access 3-Steps keyboard from two letter key keyboard, the user is required to select the characters ‘1#’.

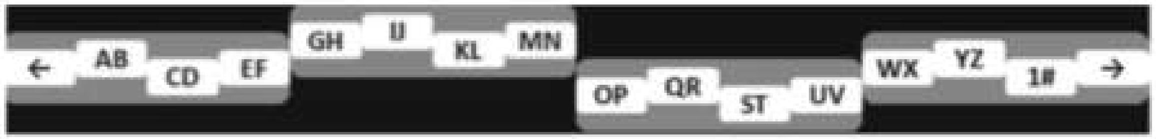


Fig. 3. Layout of two letter key keyboard [1]

Two letter key keyboard further added dictionary support features. When a user select combinations of letters using just two head movements. A suggestion panel appears which helps a user in selecting the right word from the combinations formed from 2 letters combined. In order to select the suggestion panel, a user is required to move in the right direction twice.

Nowosielski has mentioned that most of the words formed in the dictionary are of 7 or 8 letters. Author has provided word length statistic to support his analysis. The analysis is shown in Fig. 4.

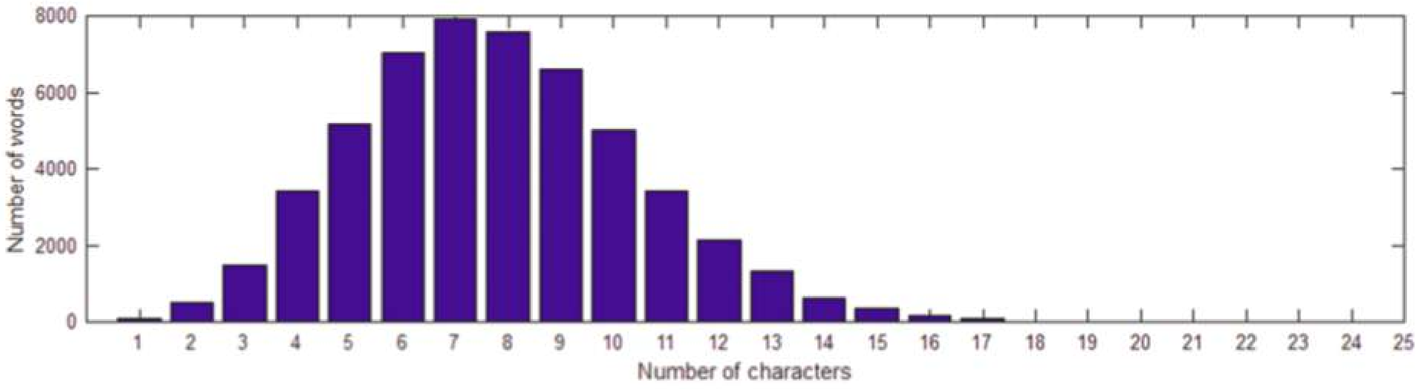


Fig. 4. Word Length Statistics

Nowosielski has provided a convenient example to show the usage of the two-step keyboard. The user could move the head in four directions left(L), up (U), Right(R), and Down(D) and to reach a combination of the alphabet it requires two head movements. Using these movements, the author has shown an example where the user moves the head couple of times “LD -> DL -> UR -> DL -> DR”. The arrow key in the explanation separates the combinations of letters. To explain the example more clearly, first LD movement enables the user to select character combinations "CD", further moving to the second selection "DL" will select the character combination "OP". Using these movements, the user makes five combined character combinations. The plausible word using the 5 combinations would be "compu." User provided with a list of word suggestions based on the combination of characters chosen earlier. The number of words in the list is kept to four, so that word suggestion panel is not flooded with words and suggestion could be selected faster. Fig. 5. represents the process in which the combination of characters "compu" is selected and suggestion panel showing the list of possible suggestions.

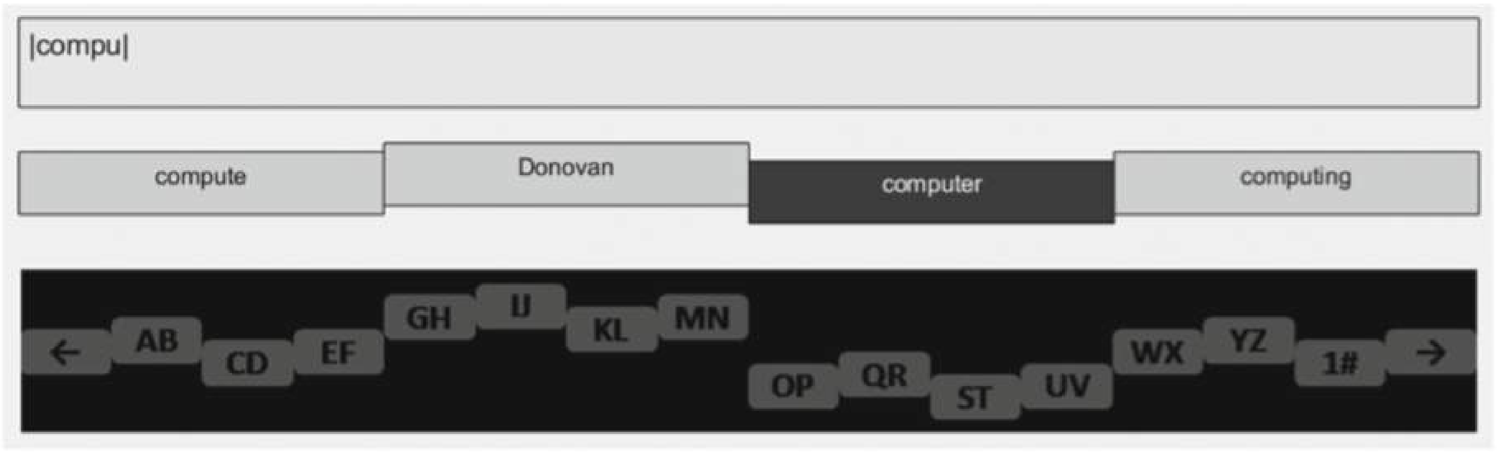


Fig. 5. Two-letter keyboard in action

2.2. <u>Face Recognition Strategies</u>

There are several ways a user could detect a face in an image. It is important to understand the pro and cons of using methods which allows detecting faces in the images. In the current sub-section, firstly, we will understand the use of the Adaboost algorithm and later, we would understand the few techniques that allow facial detection in an image.

2.2.1. Adaboost algorithm [3] [4]

In Adaboost algorithm, the main objective is to train several Haar-Based weak classifiers with the help of a lot of negative and positive images. Initially, the image is trained to recognize weak classifier. Once the image is trained to recognize for weak classifiers, a combination of weak classifiers is chosen to form a single strong classifier. Finally, a cascade of strong classifiers is used together to enhance the performance of the detection.

To understand the concept with the help of an example, let say if we would like to detect the images with a facial region. Then the example of weak classifier would be the presence of eyebrows, nose, hairs, cheeks, ears, lips, eyes, pupil of the eye and many more. After the several weak classifiers are trained, then we would select combinations of the weak classifier to build a single strong classifier for example presence of a pupil, eyes, and eyebrows could build one strong classifier. Finally, all the strong classifier are combined together to output a single result. Cascade of strong classifiers results in increased accuracy.

2.2.2. Face detection based on skin color and Adaboost algorithm [3]

In [3], the author explains that clustering features can be seen once the brightness is eliminated from the skin colors. With the help of this clustering of features, one can sort the region in the image where skin color is detected. The author specifies to remove the scatter of face region in the image obtained as a result of Binarization process.

Author mentions that skin color detection has the benefit of detecting the facial regions in the image but has a major drawback that using this approach, it results in many false positive detections with complex backgrounds. To enhance the accuracy to detect the facial region in the image, the author has combined the usage of the Adaboost algorithm and skin color based detection algorithm. Flowchart showing the steps of detection using the combined approach is shown in Fig. 6.

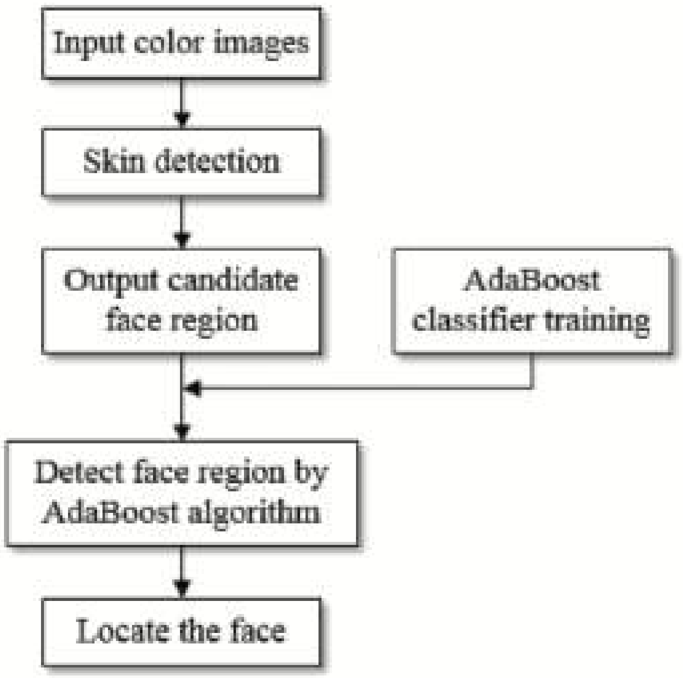

Input color images
Skin detection
Output candidate face region
AdaBoost classifier training
Detect face region by AdaBoost algorithm
Locate the face

Fig. 6. Flowchart for face detection using skin color and Adaboost algorithm [3]

Author has provided an analysis comparing the results achieved over two hundred and twenty images using the Adaboost algorithm, skin color approach, and combined approach which uses Adaboost approach to detect facial region over skin color detection approach. The performance comparison can be checked in TABLE I.

TABLE I. Analysis of comparison of the performance of skin color, Adaboost and combined approach [3]

Detection Method	Precision
Skin Color	84.1%
Adaboost	89.1%
Combined Algorithm	93.2%

2.2.3. Face detection algorithm based on Adaboost and new Haar-Like-Feature [4]

In [4], the author has introduced new Haar-Like features alongside the existing Haar based features available in Adaboost. Combination of both the Haar-Like features increases the accuracy in detecting the facial region in the image. Author mentions that the new Haar-Like features are more helpful in detecting the facial region because of the geometric shape of the feature designed. The new Haar-Like feature resembles the shape of letter 'T' as shown in Fig. 7. As per the author, the face could be divided into several combinations of 'T' shaped region. To support the analysis, the author has provided another figure depicting the facial features formed using the combination of T regions and can be checked in Fig. 8.

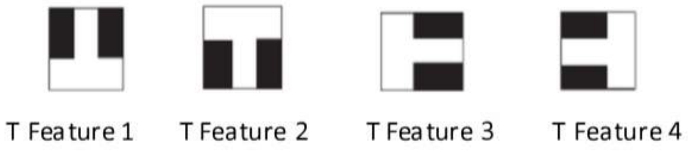


Fig. 7. New Haar-Like Features [4]

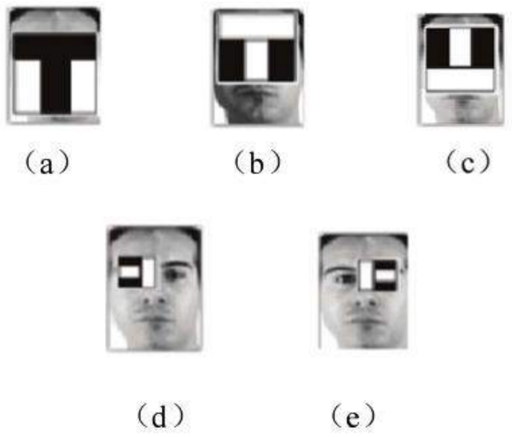


Fig. 8. Relationship between facial region and New Haar-Like features [4]

Author mentions that from Fig. 8., it can be clear that facial region could be divided into a combination of T-shaped regions as shown in Fig. 7. From Fig. 8. (a), it can be seen facial region using the T-shaped feature would cover humans mouth, nose eyebrows, and eyes. The white region in the Fig. 8. (a) would cover the cheeks regions of the face. Further, Fig. 8. (b) depicts that nose region and forehead region of a human could be detected using the T Feature 2 from Fig. 7. The relation between the eyes and the nose region could be obtained with from T Feature 3 and T Feature 4 from Fig. 7. as shown in Fig. 8. Author has provided a reference to algorithm depicted in the form of a flowchart as shown in Fig. 9.

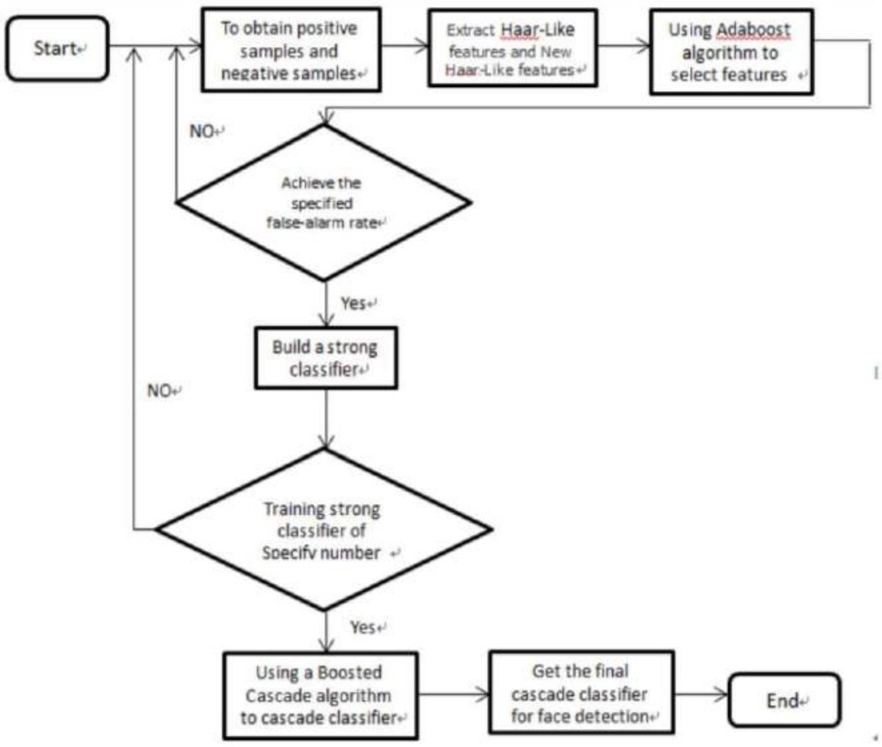

Start
To obtain positive samples and negative samples
Extract Haar-Like features and New Haar-Like features
Using Adaboost algorithm to select features
NO
Achieve the specified false-alarm rate
Yes
Build a strong classifier
NO
Training strong classifier of Specify number
Yes
Using a Boosted Cascade algorithm to cascade classifier
Get the final cascade classifier for face detection
End

Fig. 9. Flowchart of face detection algorithm using new Haar-Like features [4]

Finally, the author concludes that adding new Haar-Features to the existing Haar-features created a significant difference in performance of the Adaboost algorithm in terms of speed and accuracy.

2.2.4. Face detection using neural networks [5]

In [5], the author explains the approach to determine the presence of a facial region in the image with a simple approach. Author explains that the detection of the facial region is divided into two stages. In the first stage, several neural-network based filters are tried to determine whether a face is present in a frame or not. In the second stage, all the overlapping frames are merged together.

The algorithm used by the author is represented in the Fig. 10. Author initially creates a window of 20 x 20 frame capturing part of the image. Since the size of the face could be greater than the chosen size of the frame, author continuously reduces the size of the image at different scales. Once this window captures a small fragment of this images, they are passed to the neural network. The neural network consists of three types of hidden units to identify the facial region in the image. The hidden units are shown in the Fig. 10. This hidden unit produces the result whether the given fragment of the image contains face or not. If the prediction of the facial region in the fragment of the image is wrong, then the parameter that they are trying to learn are updated.

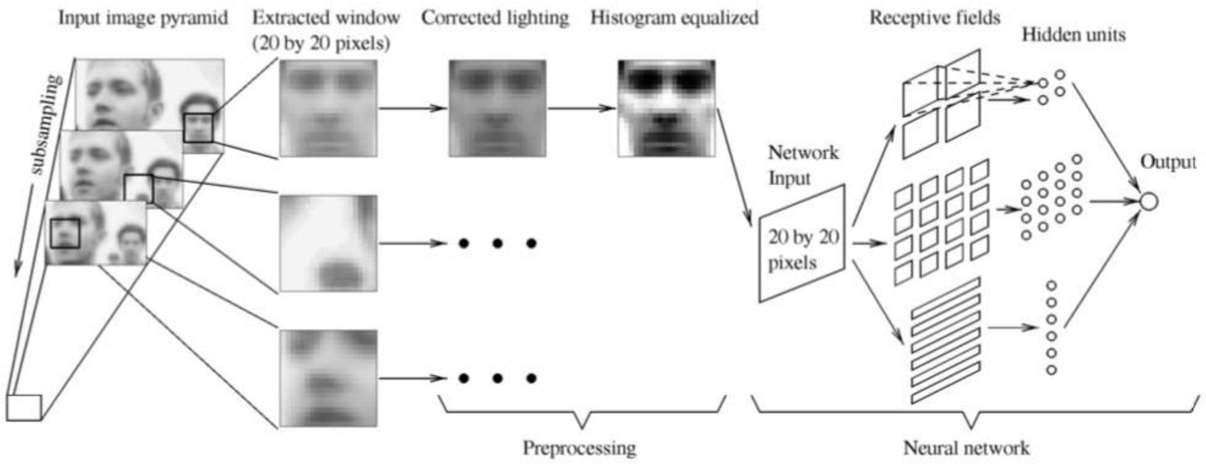


Fig. 10. basic algorithm to detect face detection in neural network [5]

There could be several false detections in this scenario, in order to prevent the detection system from false detection, the second stage of the technique helps. When several neighboring windows identify the face region at different scales, the author concludes that the current region represented by the window will either contain face or not. The process of identifying the face region in the window can be seen in the image Fig. 11.

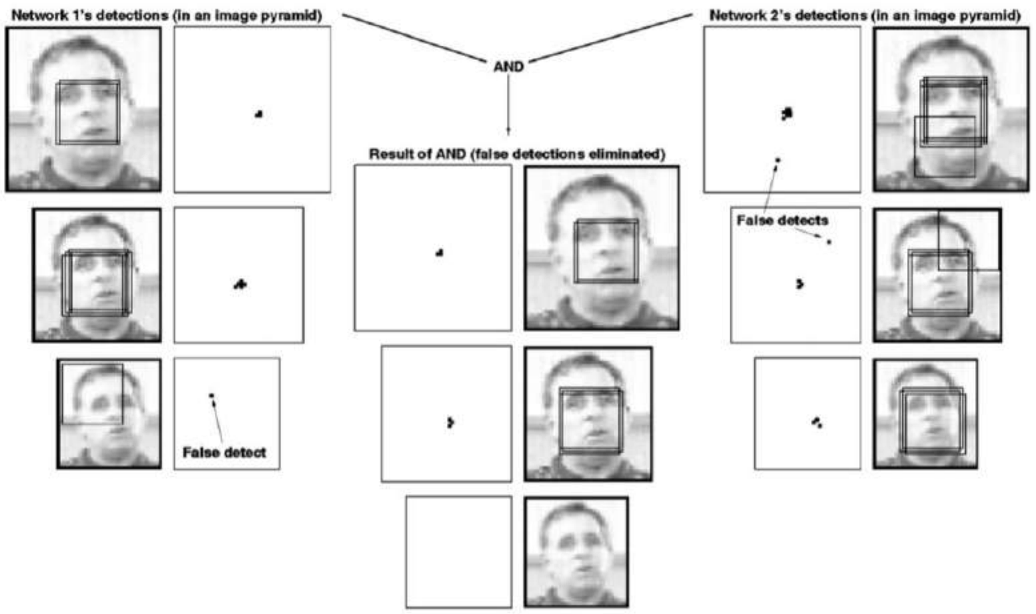

Network 1's detections (in an image pyramid)
AND
Network 2's detections (in an image pyramid)
Result of AND (false detections eliminated)
False detects
False detect

Fig. 11. Removing false detection with the help of overlapping stage [5]

Finally, the author claims that with different network models tested in [5], the accuracy of detecting the facial region in the image lies between 77.9% to 90.3% with complex background images. Author has also shared the results of the image detecting the face in different backgrounds as shown in Fig 12.

2.3. Tensorflow [6] [7]

Tensorflow is an open source platform developed by Google Brain Team. It was initially released in 2015. Tensorflow provides flexibility to develop complex machine learning applications with ease. There are several open source tutorials available to learn how to code and develop applications in machine learning using Tensorflow. Tensorflow also supports capabilities to run a machine learning tasks over Nvidia's Graphical Processing Unit (GPU), because of which training and testing a machine learning application takes significantly lesser time.

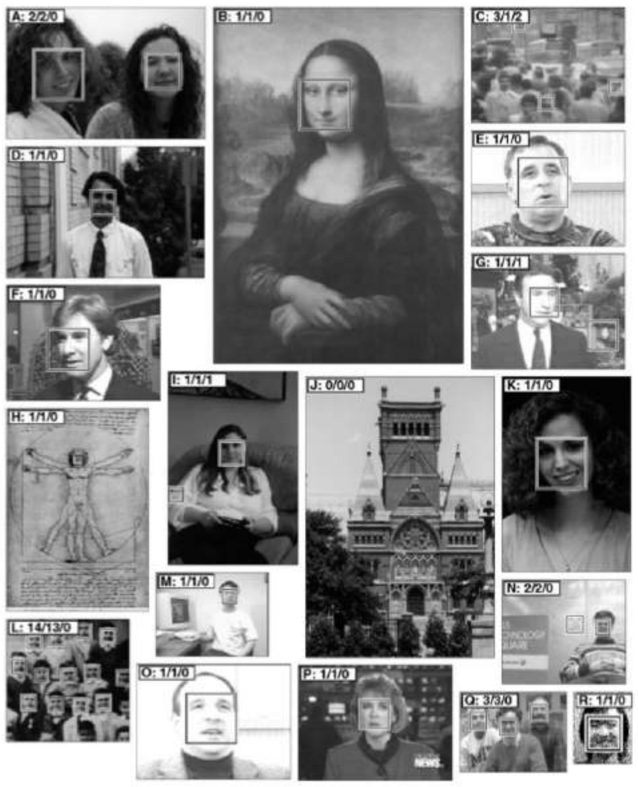


Fig. 12. Output of neural network model trying to identify faces in the images [5]

2.3.1. Tensorflow object detection API [8]

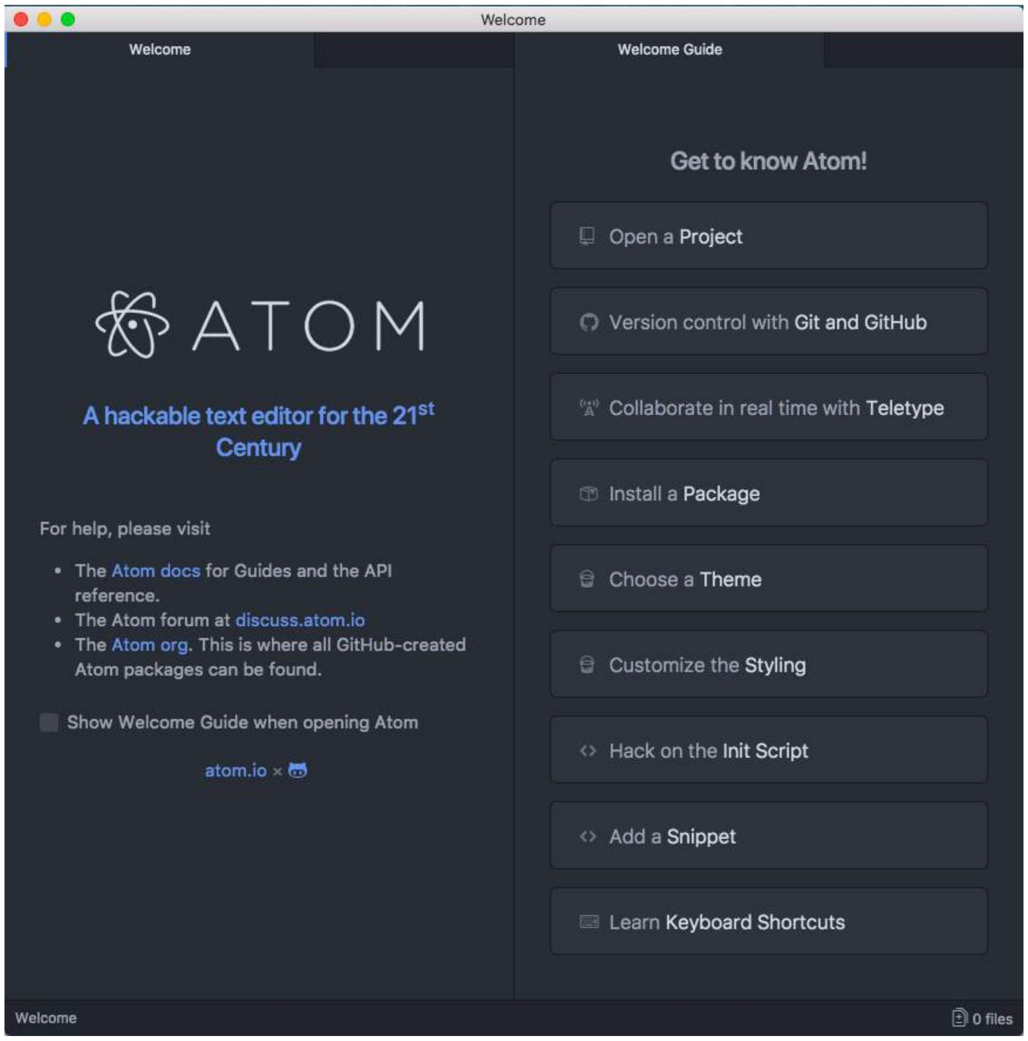
Tensorflow's object detection API provides the pre-trained models and configuration to detect several objects in advance. It also provides the flexibility to develop customizable models to detect the objects that are of interest. It also provides optimized configuration files to train and test to detect objects that are of interest, whose performance detail can be checked at [9]. Alternatively, the user could write their own configuration file as per there requirement.

For example, a user trying to train and test models to detect bowls containing Mac and Cheese using Tensorflow object detection API [10].

2.4. <u>Atom Editor [11] [12]</u>

Atom editor is an open source text-editor which provides support for Windows, Linux and OS X operating system. Atom IDE is a cross-platform editor. Atom editor provides flexibility to add an extension to the editor to enhance the experience of the editor. There are more than 7000 packages and also provides the flexibility for developers to develop their own extension as per user's requirement.

Look of Atom IDE is shown in the Fig. 13. which shows the basic layout that would be shown once the editor is launched in dark theme mode. In order to add a new extension, a user is required to search in the packages. Atom editor provides a couple of themes in advance to a user which could be configured or changed at a later point in time by the user as the user's requirement. User is also free to add in new themes from packages or build one from scratch.


Welcome

Welcome

Welcome Guide

ATOM

A hackable text editor for the 21st Century

For help, please visit

- The Atom docs for Guides and the API reference.
- The Atom forum at discuss.atom.io
- The Atom org. This is where all GitHub-created Atom packages can be found.

Show Welcome Guide when opening Atom

atom.io ×

Get to know Atom!

Fig. 13. Layout of Atom editor on launch of application

3. DESIGN AND ARCHITECTURE

The proposed design of the "Improved Hands-Free Text Entry System" (IHFTES) is novel. The current system is built keeping the important aspects of design that the author believed needed improvements. Author believes that frequency of the character plays an important role in placing the characters in groups and accessing them. Fig. 14. and Fig. 15. shows the frequency count of characters in English words.

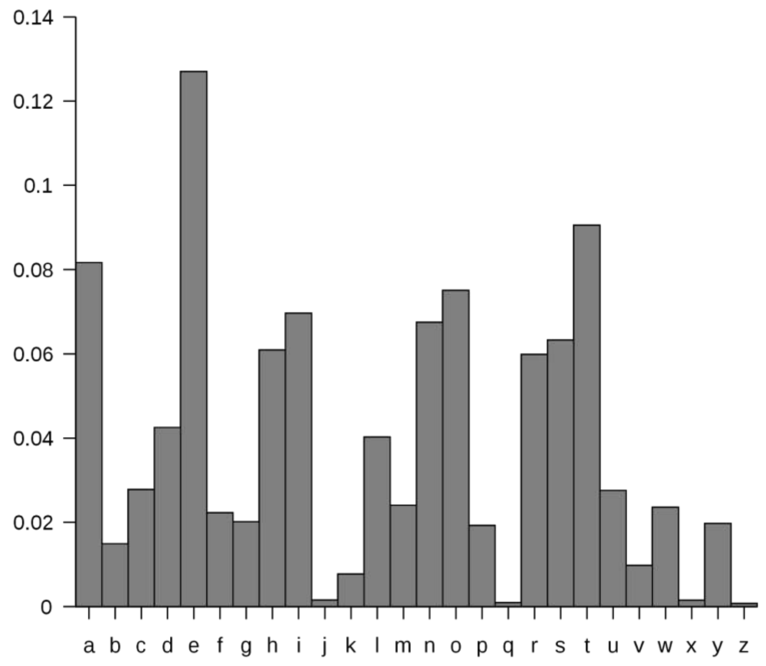


Fig. 14. Frequency Count of characters

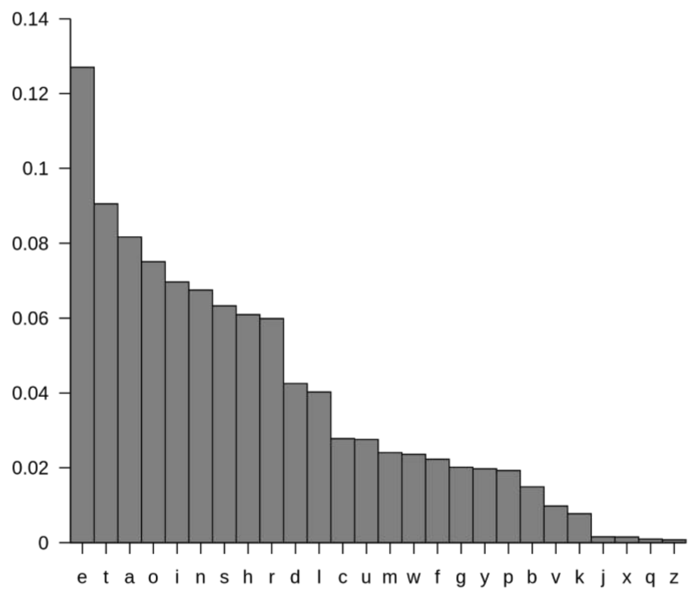


Fig. 15. Frequency count in decreasing order of use of character

The author continuously tracks rigid motions and non-rigid motions. Rigid motions are constrained to translation motions which are tracking four directional (up, down, left or right) movement like Nowosielski [1] [2] used. Non-rigid motions are tracking the facial emotions made by the user and provide different shortcut features illustrated in the menu on the bottom right corner. Shortcut feature available with the help non-rigid motions can also be accessed using rigid-movements but accessing them using rigid motions takes a number of moves.

The author provides an elegant user interface to interact with the user. Neither all the characters could be reached in three steps, nor all characters could be reached in two steps. The current design for IHFTES is modal. There are two modes, one is alphabetical, which helps in selecting the characters, punctuations, change of case, add space, tab space or newline characters. Second modes allow adding digits, operators, space and other useful symbols.

Fig. 16. provides the overview of alphabetic modal pane of the IHFTES. This is also the view that would be displayed once the character a user is trying to select has been selected. This pane is also visible when the user starts the system.

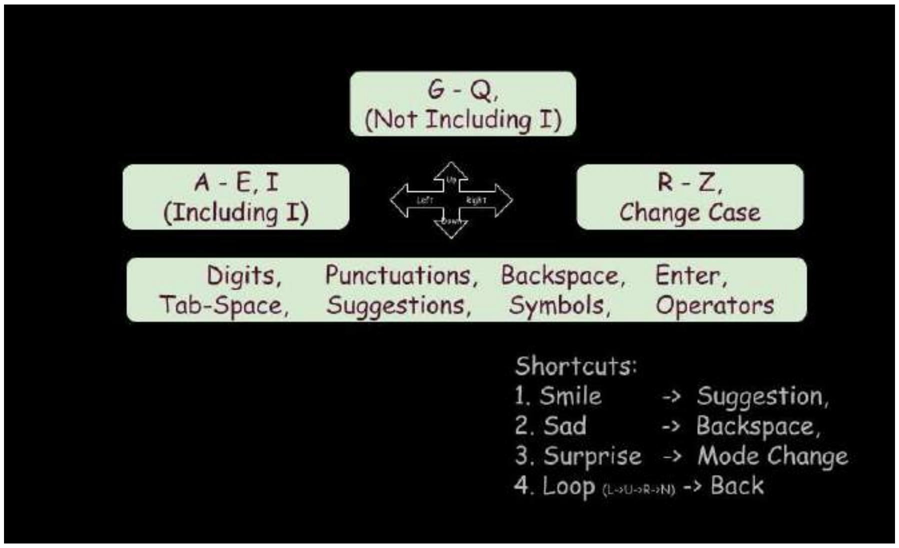


Fig. 16. Key Map Design Overview

As mentioned earlier there are shortcuts provided which could be selected using the non-rigid motions. If the user's non-rigid motion is not detected, then they could still be selected using the rigid motions by moving towards the appropriate direction. If the user moves the head towards the left direction.

If the user moves the head in the left direction in the main menu, then characters A-E and I are available to be selected. The arrangement of the characters can be seen in Fig. 17.

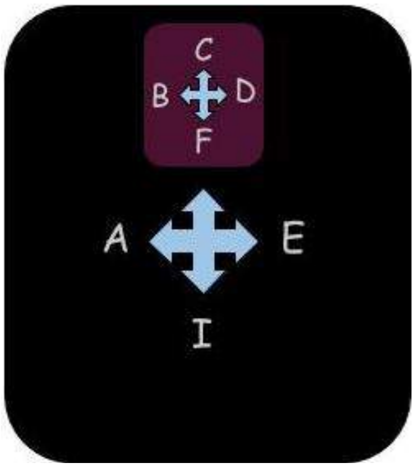


Fig. 17. Layout of characters A-E and I (left direction from main menu)

As it can be seen from the image that characters ‘A’, ‘E’, and ‘I’ can be selected in two steps. To select characters ‘B’, ‘C’, ‘D’, and ‘F’, a user has to further have move head in the upward direction. Finally, after moving the head in the upward direction, to select the character placed in the appropriate direction, the user needs to move the head in the corresponding direction. Which would mean that in order to select the characters ‘B’, ‘C’, ‘D’, ‘F’ user had to move head three times.

Fig. 17. helps in understand the important aspect, which that some of the characters can be reached in two steps while others could be reached in three steps. How did we decide which characters should be reached in three steps and which one should be allowed to get accessed in two steps?

The answer lies in the frequency count of the characters. It can be seen from the following images that top seven characters having the maximum frequency count are chosen to provide access in two steps while rest of the characters are provided to be get accessed in three steps. The top seven characters that based on frequency count which could be accessed in two steps in alphabetical order are ‘A’, ‘E’, ‘I’, ‘N’, ‘O’, ‘S’, and ‘T’. Rest of the characters could be accessed in three steps.

If the user moves the head in the up direction from the main menu displayed in Fig. 16., the characters ‘G’ to ‘Q’ not including the character ‘I’ are available to select. The arrangement of the characters can be seen in Fig. 18.

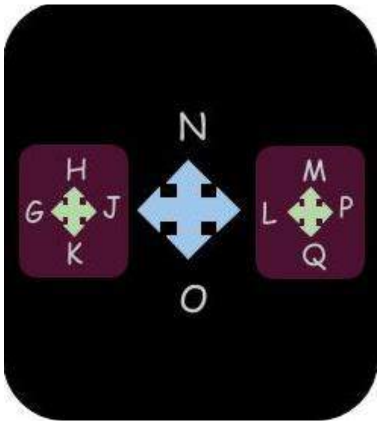


Fig. 18. Layout of Characters G-Q not Including character I (up direction from main menu)

It can be seen from Fig. 18. that the characters ‘N’, and ‘O’ can be selected in two steps by moving the head in the up or down direction. Whereas to select the characters ‘G’, ‘H’, ‘J’, ‘K’, the user is required to move the head in the left direction and further move in the direction in which user would like to select the character. Similarly, to select characters ‘L’, ‘M’, ‘P’, and ‘Q’ user is required to move the head in the right direction and then finally move in the direction as per the choice of the character.

To select a character from R to Z or change the case from lower to upper or from upper to lower, the user is required to move the head in the right direction from the main menu as illustrated in Fig. 16. The arrangement of characters R to Z and change case option is shown in Fig. 19.

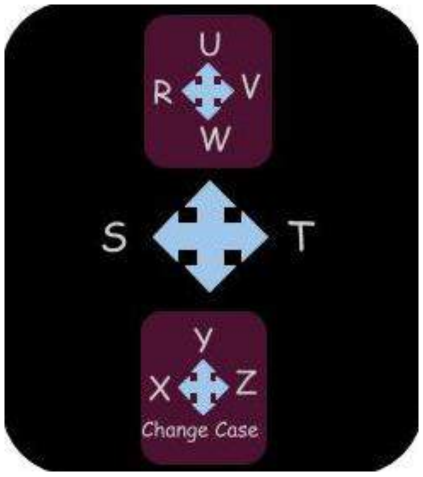


Fig. 19. Layout of characters R to Z and change case option (right direction from main menu)

It can be seen from Fig. 19. that characters ‘S’ and ‘T’ are available in two steps, and can be selected by moving the head in the corresponding direction. The characters ‘R’, ‘U’, ‘V’, and ‘W’ can be selected by moving the head in the upward direction and further moving the head in the direction of the character of interest. Similarly, characters ‘X’, ‘Y’, ‘Z’ and ‘Change Case’ option can be selected by moving the head in the downward direction and finally, moving in the direction of the interested character or option.

Moving the head towards the downward direction from the main menu illustrated in Fig. 16. provides the flexibility to choose from a number of options. By entering into this sub-menu, a user could either select a suggestion, or change modal plane from alphabetical to numerals and other symbols using the option ‘1#’, or select a correct punctuation or add a new line or space or tab space, or backspace. The representation of the options available when a user moves in the downward direction is shown in Fig. 20.

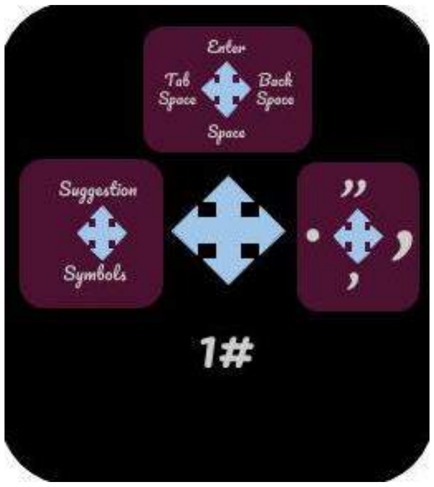


Fig. 20. Layout of options in downward direction from main menu

From Fig. 20., it can be seen that to change the mode from alphabetical to numerals and other symbols could be done in two steps by moving the head in the downward direction. Whereas to select a proper punctuation user need to move the head in the right direction in this current menu shown in Fig. 20. and then select the appropriate punctuation. To select the option to enter a new line character or add a space or tab space or remove the last character user need to move in the up direction from the menu illustrated in Fig. 20. and finally, move the head in the direction of the interested option. Finally, to select the option to check any suggestion or use any symbols users need to move the head in the left direction and finally, move the head in the direction of the interested option.

Once a user enters the numeral modes, the user stays in numerals mode until and unless the user chose to exit the numeral mode. Fig. 21. shows the representation of numeral keys overview and how they are placed.

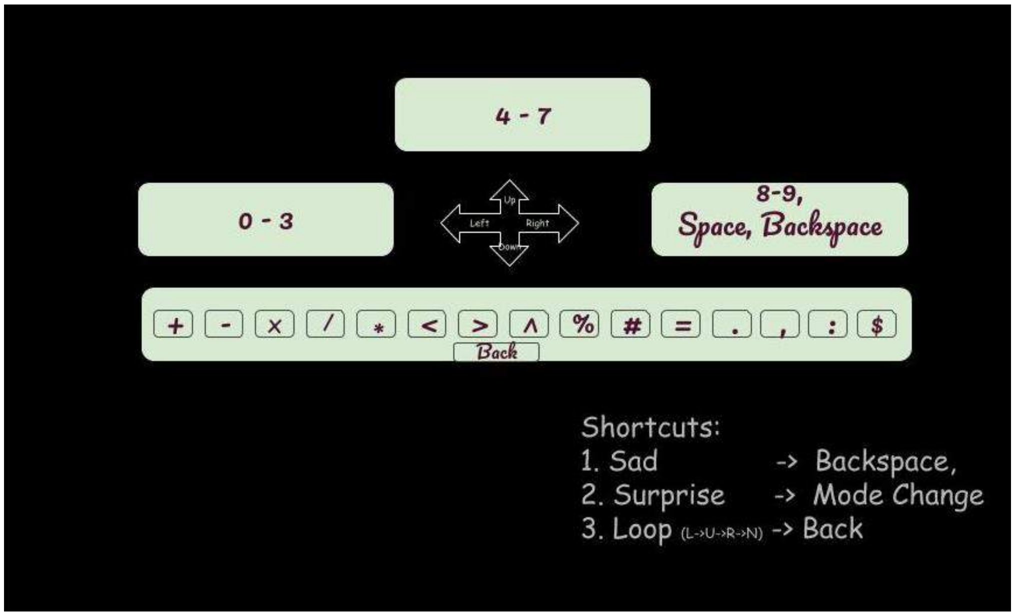

4 - 7
0 - 3
Up
Left
Right
Down
8-9,
Space, Backspace
+ - x / * < > ^ % # = . , : $
Back
Shortcuts:
1. Sad -> Backspace,
2. Surprise -> Mode Change
3. Loop (L->U->R->N) -> Back

Fig. 21. Layout of numeral keyboard

The layout depicts that numerals can be accessed from left, up or right direction. Down direction provides further options to select one of the operators or go back to alphabetical mode. There are few shortcuts also provided which would help the user select one of the options faster.

Fig. 22. Illustrates the layout of numerals 0 to 3. In order to select the numerals 0 to 3 the user is required to move the head towards the left direction.

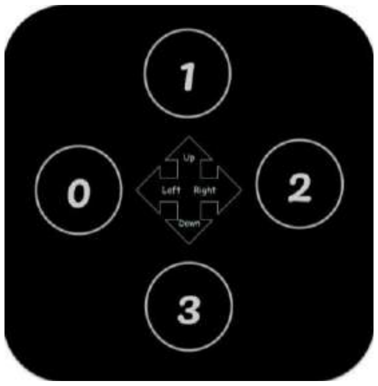


Fig. 22. Layout of the numbers 0 to 3 (Accessed from left direction from numeral mode)

As it can be seen from Fig. 22. the numbers 0 to 3 can be selected by moving the head in the direction in which the number is placed.

The layout of numerals 4 to 7 can be seen in Fig. 23. To select a number in the range from 4 to 7 the user needs to move the head in the upward direction from the numeral mode.

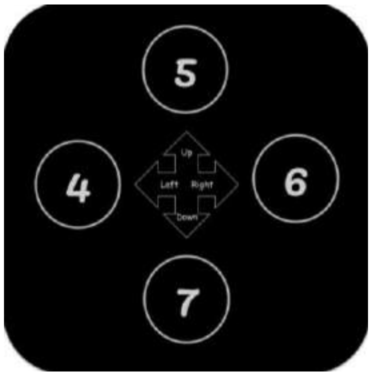


Fig. 23. Layout of numeral 4 to 7

Fig. 24. shows the layout arrangement of numbers 8 and 9 also provides the user the options of space or backspace. This arrangement could be accessed by moving the head in the right direction from the numeral mode.

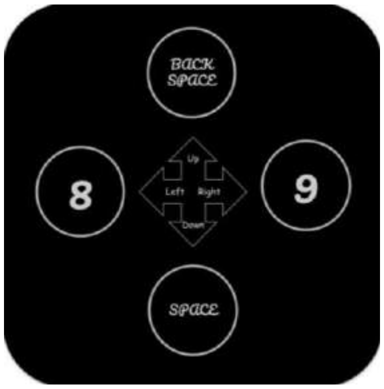


Fig. 24. Layout of number 8 to 9, space and backspace

Fig. 22., Fig. 23., and Fig. 24. shows that numbers 0 to 9, space and backspace could be accessed in two steps once the user enters the numeral mode.

Once the user moves in the down direction, there are several other options that available to select. Fig. 25. shows the layout of the options that are available.

The layout of the options available in the left direction of Fig. 25. could be checked in Fig. 26. Fig. 27. shows the options available in up direction of the Fig. 25. Fig. 28. shows the options available in the right direction of Fig. 25. And finally, Fig. 29. shows the layout of options available in down direction of Fig. 25.

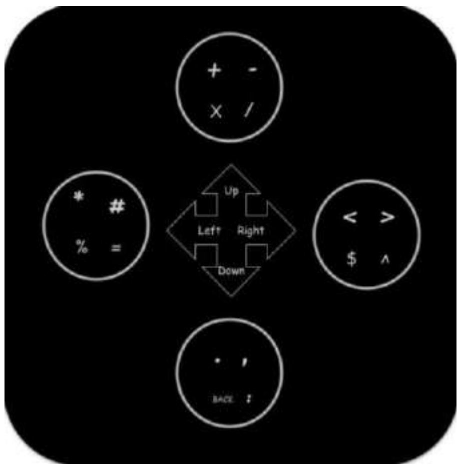


Fig. 25. Layout of options available at Down direction in numeral mode

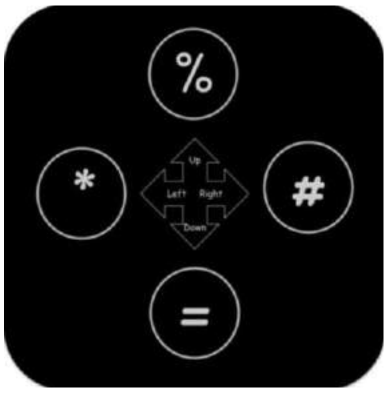


Fig. 26. Layout of options available in left direction of Fig. 25.

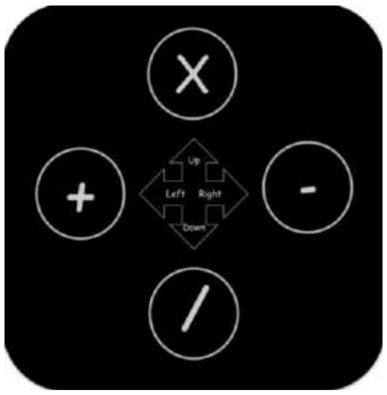


Fig. 27. Layout of options available in up direction of Fig. 25.

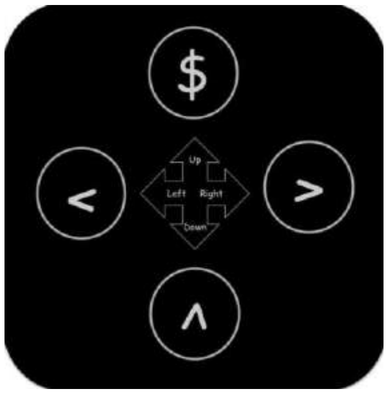


Fig. 28. Layout of options available in right direction of Fig. 25.

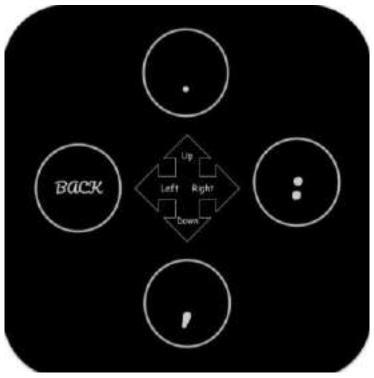


Fig. 29. Layout of options available in down direction of Fig. 25.

Moving the head in one of the direction in Fig. 26., Fig. 27., Fig. 28., and Fig. 29. allows selecting the available symbol in the corresponding direction.

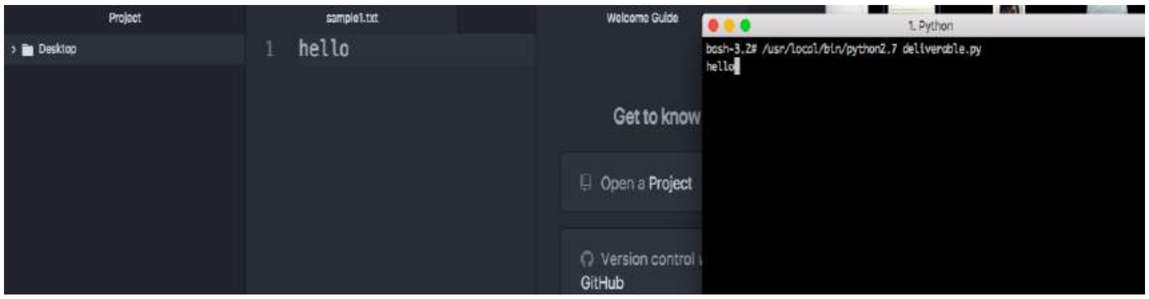
4. IMPLEMENTATION

IHFTES is dependent on several sub-components. Understanding the implementation and requirement of the sub-components is vital since each sub-component would support the IHFTES. A key-logger tool was implemented to listen to continuous text entries. Once the text entries are generated, an editor was needed to display the text, Atom editor was used for this purpose. An extension was developed for Atom editor which showed the updated text generated. Once the key-logger and extension for the Atom editor are built, word prediction functionality is implemented. Word prediction functionality will try to predict the word user is trying to type based on partial characters of the word typed. Another component helped in identifying the gestures based on the human face. These gestures are helpful for the purpose of non-rigid motions. The final sub-component was to detect a tool to detect head movements. Once all this component is in place, the whole tool was built to detect the text characters typed using the head movements by combining the sub-components and design or the system.

4.1. Key-Logger Tool

Key-logger tool is a command line tool developed in Python programming language. The objective of the tool is to identify the keys pressed on via keyboard or any other source. This tool was needed since when the user enters a key, it is required to identify that what key was pressed. This tool developed showed the updated output on the Atom [12] editor in the active text editor window. Active text editor window is the window that would be displayed when the editor would be launched. Since there is flexibility in Atom [12] editor to add new tabs to add or edit multiple files at the same time, active text editor window means the current window that is open to editing the file.

The tool uses subprocess library of Python 2.7 to run the terminal command to open the Atom editor. The file is stored on the desktop with the name sample1.txt. This file would be displayed on the Atom editor. When the program is run as an administrator, the output written on the terminal would be reflected on the Atom editor. Pynput library of python is used to continuously monitor the keyboard entry. Fig. 30. shows the output depicting the Atom editor in the dark theme mode reflecting the text entry entered on the right-hand side of the terminal window.



```
hello
```

```
bash-3.2# /usr/local/bin/python2.7 deliverable.py
hello
```

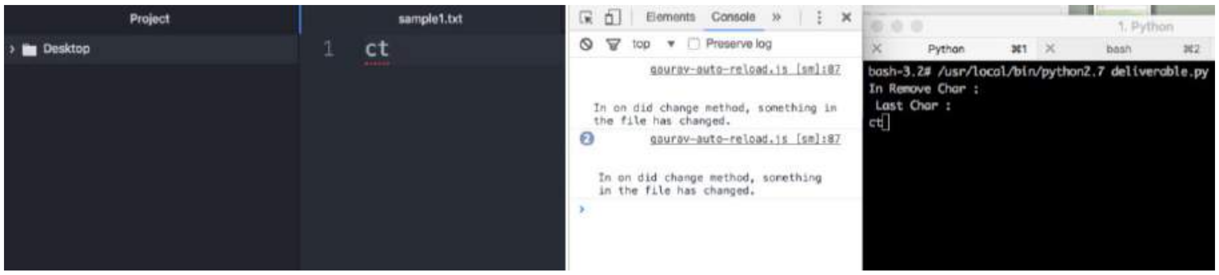
Fig. 30. Output of Key-logger tool

If while running the program a user encounters the error that atom command not found even when the Atom editor is downloaded and installed properly, the user is required to create a symbolic link to the Atom text editor in their respective operating system. Further, if the alphanumeric keys entered by the user are not shown Atom editor text window, then the user is required to run the program as an administrator.

4.2. <u>Atom Editor Extension -Auto Reload</u>

Auto reload is developed in Javascript language and is used to continuously monitor the files that is opened in the text editor window. If there are any change made to any of the files that currently opened in the Atom editor via some other medium other than Atom editor, then the Atom editor would reflect those changes. The auto-reload feature is available by default in the Atom editor and a user is not required to make any changes in order to enable it. For the curiosity of learning purposes and to use the extension as a part of the project, I have implemented an extension which provides the same feature. To enable the extension, a user is required to select the option available to auto reload in the sub-menu in the toolbar under packages. Alternatively, a user can simply press the Alt, Shift and 'X' key to enable the extension when Atom editor is launched. To extension developed should be kept in the appropriate directory so that it could be shown in the sub-menu options in the toolbar.

Fig. 31. shows the output generated using the tool. Whenever a change is made in the file from some other medium, the console shows that there is some change that is been made in the file. An example of using the extension is shown in Fig. 31., the file sample1.txt is changed from terminal console (displayed on right side of the Fig. 31), whenever a change is detected in the file, Atom editor displays the change and also shows a comment in the console window (shown in white background in Fig. 31.) of the Atom editor that some change in the file is detected with the help of the auto-reload tool.



```
gourav-auto-reload.js [sm]:87
In on did change method, something in
the file has changed.
2 gourav-auto-reload.js [sm]:87
In on did change method, something
in the file has changed.
```

```
bash-3.2# /usr/local/bin/python2.7 deliverable.py
In Remove Char :
 Last Char :
ct
```

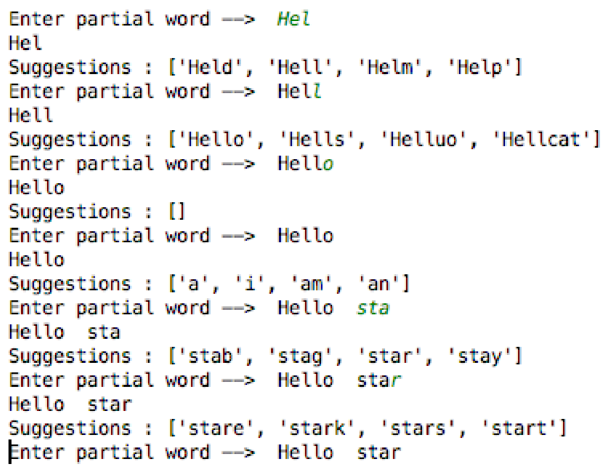
Fig. 31. Showing the output of Auto Reload Atom extension

4.3. Word Guessing Tool

Word guessing tool is developed in Python2.7. This tool helps in guessing and providing the top four results based on the partial word that is written.

Word guessing tool uses Trie data structure to store the words from the dictionary. The tool uses the dictionary corpus available at [13]. The dictionary corpus contains 10,000 words. Words in dictionary corpus also contain abbreviations, acronyms, and messaging format words. The words are initially filtered from the dictionary corpus so that the dictionary would contain valid English dictionary words. Based on the valid English dictionary words suggestion of top four words is provided to the user. When a user types a partial word, it triggers into evaluating the words that would be closest to finish in alphabetical order depending on the partial word typed. Once the tool finds the top four words that are shortest in length as compared to the other words, the result is provided to the user. The reason to limit the suggestion to four words is that since the rigid motion of head movement detection in our system will be detecting motion in four directions, it would easier for the user to select the word from the suggestion list using one single head movement.

For example, if a user types “sta” then the word guessing tool will provide the following suggestions [“stab”, “stag”, “star”, “stay”], the word suggested are alphabetically sorted. If space is given as a partial word then, it would provide the following words suggestions ['a', 'i', 'am', 'an'], since these words are the closest valid English words. If there are no words could be formed based on the partial words list, then the suggestion list will be empty. The output showing the example can be checked in Fig. 32.



```
Enter partial word --> Hel
Hel
Suggestions : ['Held', 'Hell', 'Helm', 'Help']
Enter partial word --> Hell
Hell
Suggestions : ['Hello', 'Hells', 'Helluo', 'Hellcat']
Enter partial word --> Hello
Hello
Suggestions : []
Enter partial word --> Hello
Hello
Suggestions : ['a', 'i', 'am', 'an']
Enter partial word --> Hello  sta
Hello  sta
Suggestions : ['stab', 'stag', 'star', 'stay']
Enter partial word --> Hello  star
Hello  star
Suggestions : ['stare', 'stark', 'stars', 'start']
Enter partial word --> Hello  star
```

Fig. 32. Output from Word Guessing Tool

4.4. Facial Gesture Detection

Face gesture detection is developed using Python 2.7 and Tensorflow. Face gesture detection is a neural network based model which helps in identifying the gesture made by face. It is a necessary component for the IHFTES since it provides the shortcuts available in the system using the non-rigid motions.

Several convolutional neural-network are built to train the images to identify the facial gesture. The neural network model is currently identifying faces with four emotions which are, neutral face expression having no emotions, face with sad emotion, face with happy emotion, and face with angry emotions. The best accuracy achieved is over 87% while training to detect all gestures.

Images with emotion confidence of above 85% are used for training and testing. The neural network is built using three hidden layers, one fully connected layer, and a final output layer. Initially, the image is resized to 100*100, since the images are of different sizes. Layers are filtered in such a manner so that it would capture the emotions in square pixels, vertical rectangle, and horizontal rectangle. From [5], it could be learned that forming such a neural network will help in capturing the essence of expression and faces with better accuracy.

Dataset containing the facial expression is obtained from [14]. Owner of the dataset had used to the dataset to learn the social relation traits from face images [15]. Dataset has seven different emotions available, which are neutral, sad, angry, happy, disgust, surprise, and fear.

Capturing four different emotions for the IHFTES will be enough to provide all the required shortcuts. Dataset provides good enough number of images for expressions neutral, sad, happy and surprise with emotion confidence of 85%. Fig. 33., Fig. 34., Fig. 35., and Fig. 36. shows the gesture detection with gestures neutral, happy, surprise, and sad.

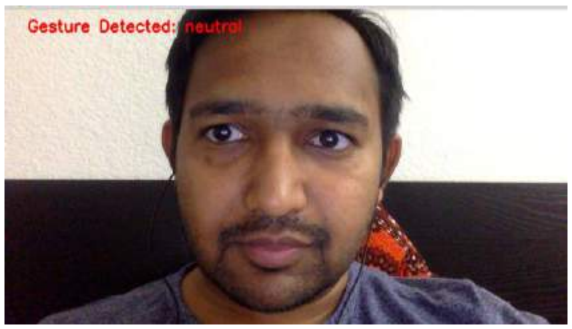


Fig. 33. Gesture detection of neutral expression

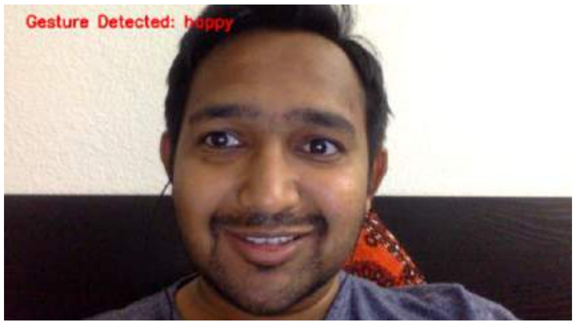


Fig. 34. Gesture detection of happy expression

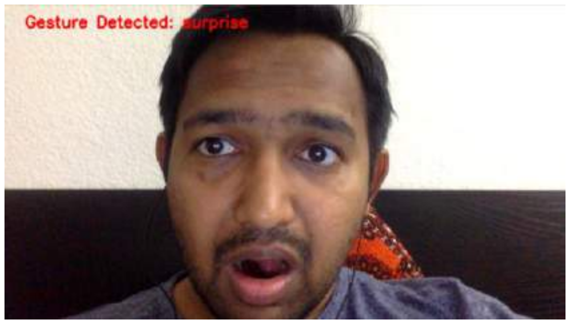


Fig. 35. Gesture detection of surprise expression

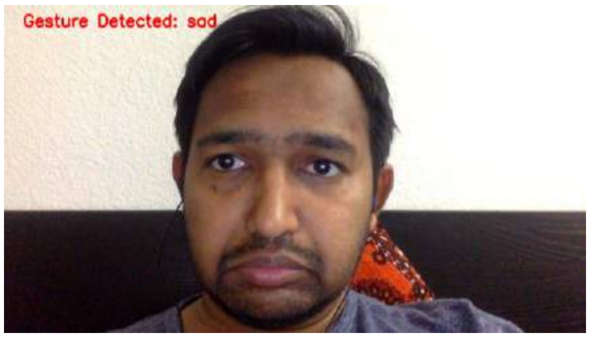


Fig. 36. Gesture detection of sad expression

4.5. Head Movement Detection

Head movement detection tries to identify the direction in which head is moved. To detect the head movement, it is necessary to identify the facial region. To detect the facial region, a Tensorflow based neural network has been used. This model identifies the facial region at different angles.

To detect head movements for IHFTES, it is necessary to identify the head movement of one single person, since the user would be the only person interacting with the system. A convenient way is required to detect head movements for a single person provided there are many persons present in the video frame. To detect a single person, the author finds the face with the maximum area of the image occupied. The reason is that the person closest to the screen will have the maximum area.

Every person has a different way to move their head in one particular direction. For example, if a user is required to move their head in the left direction, some user might tilt head in the left direction or shift head in the left direction or look in the left direction. Similarly, the difference in the shifting of head applies to other direction. To resolve this, for the first couple of frames a user is required to move their heads in the four directions so that the head movement detection system could calibrate and identify the head movements according to the user's pattern of movement. Calibration of head movement requires the user to look straight, then move the head towards left, then move the head towards up, then towards right and finally towards down side. This process is repeated every time the user restarts the system. The calibration process is shown in figures Fig. 37., Fig. 38., Fig. 39., Fig. 40., and Fig. 41.

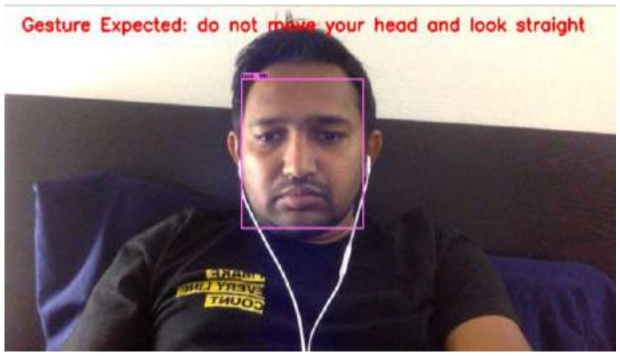


Fig. 37. Calibration process in Head movement detection to look straight

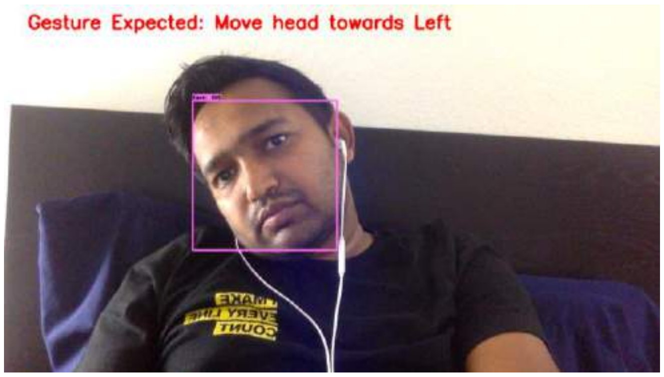


Fig. 38. Calibration process in head movement detection to shift head towards left

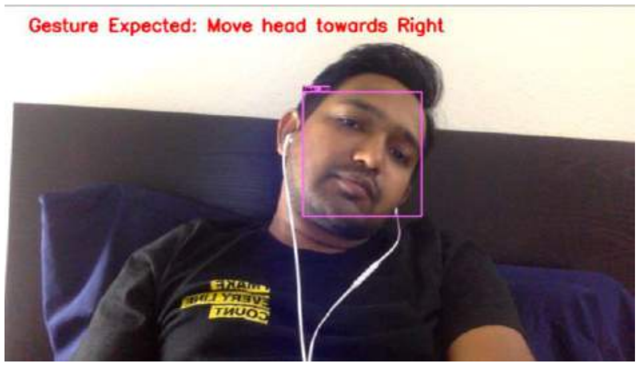


Fig. 39. Calibration process of head movement detection to shift head towards right

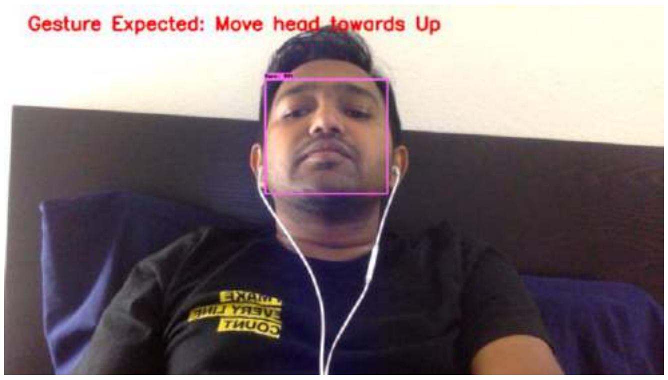


Fig. 40. Calibration process of head movement detection to shift head towards up

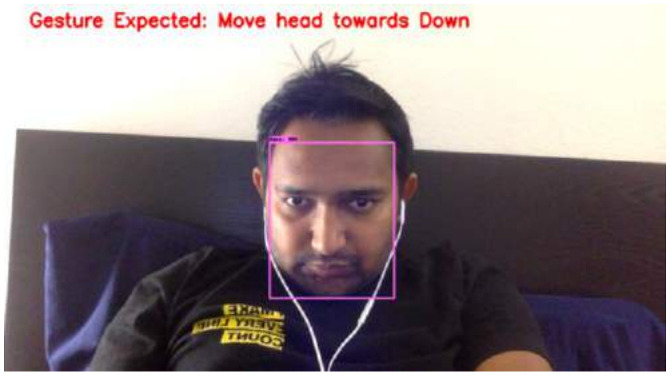


Fig. 41. Calibration process of head movement detection to shift head towards down

Once the calibration process is finished, the user head movement will be continuously tracked. When a user moves the head in any direction, the message will display on top-left hand corner displaying the position of that current is at. Fig. 42., Fig. 43., Fig. 44., Fig. 45., and Fig. 46. shows the detection phase of the head movement detection phase.

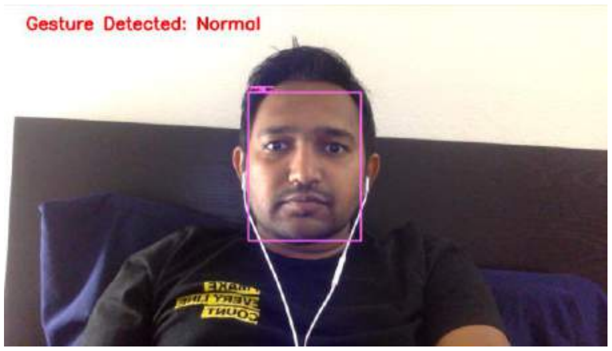


Fig. 42. Head movement detection phase detecting Normal head gestures

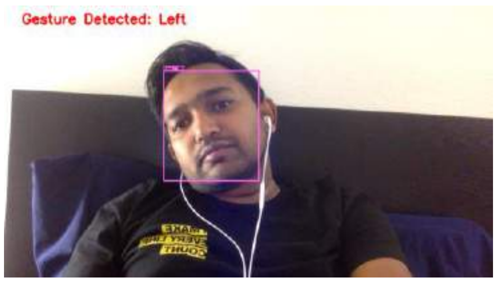


Fig. 43. Head movement detection phase detecting shifting left head gestures

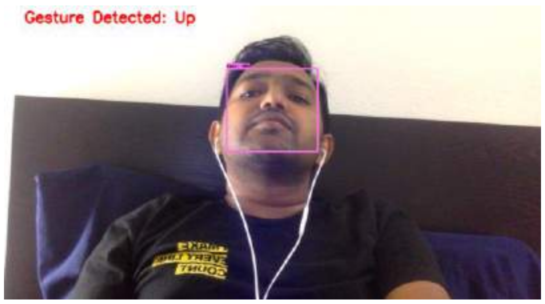


Fig. 44. Head movement detection phase detecting shifting up head gestures

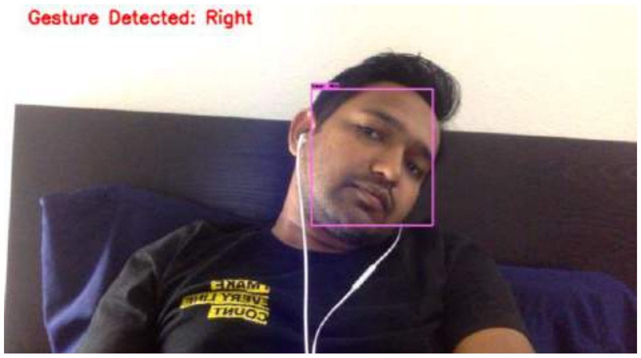


Fig. 45. Head movement detection phase detecting shifting right head gestures

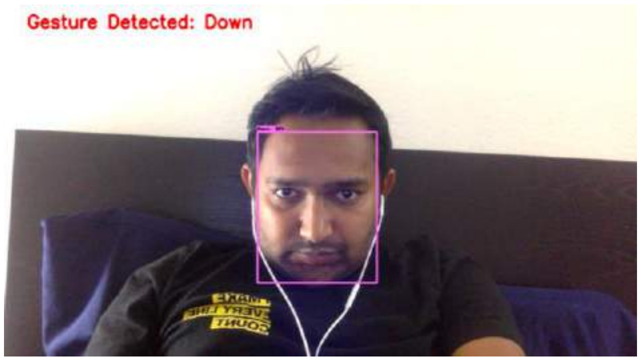


Fig. 46. Head movement detection phase detecting shifting down head gestures

5. EXPERIMENTS

Our IHFTES starts with the calibration of the head movements for each direction. Once the head movement is calibrated for each direction, the face detection window disappears from the display and the layout design of the window displaying the alphabet symbol is shown to the user. From this point onwards, whenever a user makes a motion in any of the direction, the possible options available in that direction gets selected. Once the user makes a selection of an option in any of the direction user is required to go back to the Normal position, looking straight in the window.

The output of the text generated using the motions of the head is written to a file named "SHFTE" (short for Simple Hands-Free Text Entry) on the desktop of the corresponding system where the IHFTES is running. The output is shown in the Atom text editor and console by default but a user could open the text editor of their own choice to check the updated text getting written to the SHFTE file. If the user chooses to select some other editor to check the output, the user should make sure that the editor provides the capability to update the content of the window when the text written in the file changes from some other source.

Sub-components of our implementation were unit tested separately. Observation based on the tested components is explained in the testing and observation sub-section. Dependencies for the components integrated to build the IHFTES are shown in the dependencies sub-section.

5.1. <u>Testing and Observations</u>

5.1.1. Testing and observation for detecting facial region

Detecting faces is difficult in the presence of a bright source of light behind the user. For example, a bright source of light like a lamp present in close proximity to the user could interfere with the detection. It was also difficult to detect the faces in the absence of light. For example, when a user is sitting in a pitch-dark room with no source of light. To prevent these scenarios, a user should choose an ideal place which does not have a light source in close proximity or in the background of the user. Also, a user should not test the system in the room with no light source or with a little light.

Another issue concerns multiple users in the image frame. To detect the primary person in image frame with multiple persons present, an assumption is made that the person sitting closest to the camera will have the maximum area of the facial region. An issue arises in situations where behind the person there is a poster or flyer. In such a scenario, the area of facial region the poster could be larger than the primary person. In this situation, the user would not be detected as the primary person. To prevent from this situation, the background of the person should be chosen carefully.

5.1.2. Testing and observation on head movement detection

Certain issue concerns the calibration and detection phase of head movement detection. During the calibration phase, the user can shift, tilt or move their head in the direction illustrated top left corner of the image frame. Extreme movement in one of the directions could prevent user to look at the screen. In this situation, user would skip the guidelines to follow.

Another issue concerning the calibration phase is if a user makes rapid head movements between normal and one of the four direction. These behaviors will result in calculation error for average head positons.

Another issue observed for head movement detection phase is for the users on laptop. Adjusting the position of the laptop after running the system will result in calculation error for user's head positions. For better result, camera should be still and mounted at fixed distance. Also, motions which result in the change of position of the facial region for the user should be avoided.

Another issue concerns the distance between user and camera. If a user sits too close to the camera the system will become sensitive to head movements.

5.1.3. Atom editor symbolic link

If the Atom editor cannot be opened using the Command Prompt, or Terminal, a user is required to symbolically link the Atom editor to its application. Failing to link the editor will not show the output even though the Atom editor is successfully installed system. Alternatively, a user could choose to open the file in other editor to check the output generated from IHFTES. In such situations, the output will still be visible on the Command Prompt or Terminal window.

5.2. Analysis

5.2.1. Analysis based on configuration of the system

Configuration of the computing system decides the frames that would be displayed to the user based on the configuration of the system. The system is tested on the configurations shown in TABLE II.

TABLE II. Analysis based on the computer configuration

Sr. No.	**Device Platform**	**RAM**	**GPU**	**Avg. Frames/Sec**
1.	Windows 10	16 GB	6 GB	16
2.	Mac OS	8 GB	N.A.	1.5

5.2.2. Analysis on writing context using IHFTES

The system developed was tested on couple of users. Users were requested to access three particular word as shown in TABLE III. It was observed that, while writing the first word, user took time, but as soon as the user got familiar with the design interface, typing of the word improved. Analysis is shown in TABLE III. User are tested on Mac OS platform.

TABLE III. Analysis for typing words using IHFTES

Dictionary Word	**User 1**	**User 2**	**User 3**
SIN	20 sec	34 sec	65 sec
COP	15 sec	20 Sec	45 sec
MUST	25 sec	33 Sec	45 sec

6. CONCLUSION AND FUTURE WORK

Our IHFTES is a novel system for entering text on a computer. The system developed is targeted especially for peoples whose only means of interaction with the computers would be with the head movements. The system captures rigid motions and non-rigid motions. A rigid motion of the head includes the motion of the head in one of the direction. Non-rigid motion captures the expression made by facial region. The system provides the word suggestion based on the partial word already written by the user. The number of words suggested to the user is limited to four; since the movement of the head could take place in one of the four directions, it would provide a faster selection of word suggested.

The design of the system is divided into two modes. The two modes are alphabetical mode and numerical mode. Placement of the alphabet symbol in the alphabetical mode is designed such that top seven highest frequency characters can be accessed in two steps while rest of the characters can be accessed in three steps. Once a user enters into the alphabetical mode, the user remains in the alphabetical mode until and unless user changes to the numerical mode. Similar, concept applies to the numerical mode; once a user enters into the numerical mode, the user remains in the numerical mode until and unless the user changes the mode using the shortcut or menu item. The reason for designing the architecture as a modal system is because if a user writes alphabet symbol or number, it is more likely that following characters would be of the same mode. For example, writing a mobile number using improved text free entry system in the absence of modal system, a user would need to select number mode ten times; this would drastically increase the number of steps to reach a particular character.

To conclude, IHFTES provides the flexibility to enter text using the head movements. In future, additional features will be provided to access file related operations. Features to send emails using the system will also be supported. Features with new shortcuts using additional non-rigid motions will be accommodated in future.

7. References

[1] A. Nowosielski, "Two-Letters-Key Keyboard for Predictive Touchless Typing with Head Movements," in Proceedings of International Conference on Computer Analysis of Images and Patterns, Ystad, Sweden, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-64689-3_6

[2] A. Nowosielski, "3-Steps Keyboard: Reduced Interaction Interface for Touchless Typing with Head Movements," in Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017, Polanica Zdroj, Poland, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-59162-9_24

[3] C. Lv, T. Zhang and C. Lin, "Face detection based on skin color and AdaBoost algorithm," in Control And Decision Conference (CCDC), 2017 29th Chinese, Chongqing, China, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7978729/

[4] S. Ma and L. Bai, "A face detection algorithm based on Adaboost and new Haar-Like feature," in 7th IEEE International Conference on Software Engineering and Service Science (ICSESS) , Beijing, China, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7883152/

[5] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 1, pp. 23 - 38, 1998. [Online]. Available: https://ieeexplore.ieee.org/document/655647/

[6] Wikipedia, "TensorFlow," [Online]. Available: https://en.wikipedia.org/wiki/TensorFlow.

[7] Google, "Tensorflow," [Online]. Available: https://www.tensorflow.org/.

[8] Google Brain Team, "Tensorflow Object Detection API," [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection.

[9] "Tensorflow Detection Model Zoo," [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[10] "Tensorflow tutorial for Object Detection API," [Online]. Available: https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/.

[11] Wikipedia, "Atom (text editor)," [Online]. Available: https://en.wikipedia.org/wiki/Atom_(text_editor).

[12] "Atom editor home page," [Online]. Available: https://atom.io/.

[13] J. Kaufman, "google-10000-english," [Online]. Available: https://github.com/first20hours/google-10000-english.

[14] Z. Zhang, P. Luo, C. C. Loy and X. Tang, "Learning Social Relation Traits from Face Images," [Online]. Available: http://mmlab.ie.cuhk.edu.hk/projects/socialrelation/index.html.

[15] Z. Zhang, P. Luo, C. C. Loy and X. Tang, "Learning Social Relation Traits from Face Images," in International Conference on Computer Vision (ICCV), Santiago, Chile, 2015. [Online]. Available: http://mmlab.ie.cuhk.edu.hk/projects/socialrelation/index.html.