Improved Hands-Free Text Entry System

CS 297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By Gaurav Gupta

Dec 2017

TABLE OF CONTENTS

INTRODUCTION 3
I. DELIVERABLE 1: SIMPLE PYTHON PLUGIN FOR GEDIT EDITOR.......................................4
II. DELIVERABLE 2: PYTHON TOOL TO CAPTURE IMAGES USING CAMERA ON MAC 6
III. DELIVERABLE 3: HEAD MOVEMENT DETECTION...8
IV. DELIVERABLE 4: MAP TEXT TO HEAD MOVEMENT 10
V. CONCLUSION...12
REFERENCES 13

Introduction

An input device is a hardware device which is used to send input data to the computer, which is further utilized to control and interact with the computer system. Providing an input has always played a vital role in interacting with a computer system. Contemporary input mechanisms can be categorized in several ways depending on the medium the input is provided, for example audio based input, video based input, input in the form of images. Few examples of contemporary input devices include: Keyboard, Mouse, Siri or Alexa (voice based input devices for Apple devices and Amazon Echo devices), Touchscreens (included on mobile interfaces and many others), Video based input devices like used in self driving cars where continuous frames of images are provided as input. 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 would help people with disability to interact with the computing devices easily.

Head movements would be continuously tracked with the help of images taken from camera mounted on the computing device. Nowosielski [1] [2] suggest methods to generate text from head movements. In [1], it is proposed that every alphabet could be reached with the help of at most three head movements. Author has mentioned 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, which includes alphabets, digits, punctuation, symbols, and backspace capability. Further in [2], author has mentioned that the time to reach any particular alphabet can be reduced to two head movements. In order for the system to work, he had paired two alphabets into one and provided support for English dictionary. The words formed in dictionary are suggested from the pairs of alphabets selected. User is required to select a word from this suggestion.

To understand how to implement a technique that could generate text entries with the help of head movements, it is vital to understand different smaller fragments. Implementing this smaller fragments would helped in achieving the final goal. The organization of the paper is divided into four deliverables. First deliverable is to develop any extension for the text editor, so that it could provide a better understanding for the text editor. Second deliverable will help in interacting with camera of the computing device. Third deliverable provides a mechanism to identify head movements and finally the fourth deliverable will map text from head movements.

I. Deliverable 1: Simple Python Plugin for Gedit Editor

The objective of this deliverable was to develop a plugin for the Gedit editor using the Python programming language. Originally, we were going to use Gedit as the target text editor on which our input system produces, so this deliverable allowed me to gain experience in making plugins for Gedit.

The tool developed for the Gedit editor was “Toggle Content”. It was developed using the python programming language. The plugin developed lets the user show or hide the content of a window using the keyboard shortcut or with the help of a submenu item available in the tool menu items. If there was some text written over the screen, and the option hide/show was selected than the content of the window would disappear, when the same option is selected again the whole text would be rewritten on the screen.

Tool developed is useful in scenarios where an unwanted person was trying to peep at the screen of the user. In order to prevent the person seeing on the screen a simple keyboard shortcut or a submenu item access in toolbar provided a convenient way to hide the information available on the window.

To enable the extension on Gedit editor, the tool needs to be put in the folder /home/username/.local/share/gedit/plugins on Mac. If the folder did not exist, then it needs to be created. Once the repository holding the entire code was placed in the correct folder, the plugin is enabled from preferences setting available in the Gedit editor. Fig. 1 shows how to select the menu item preferences within the Gedit editor.

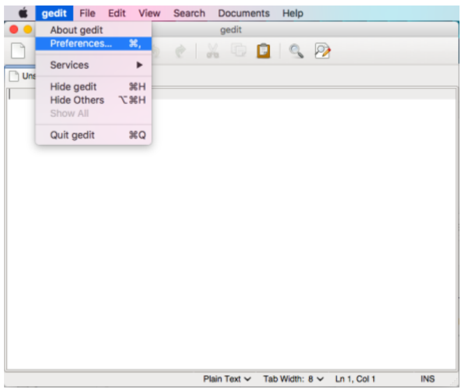
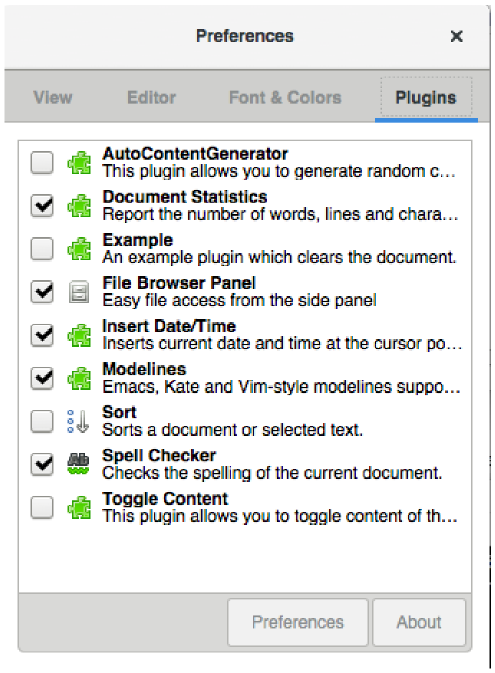


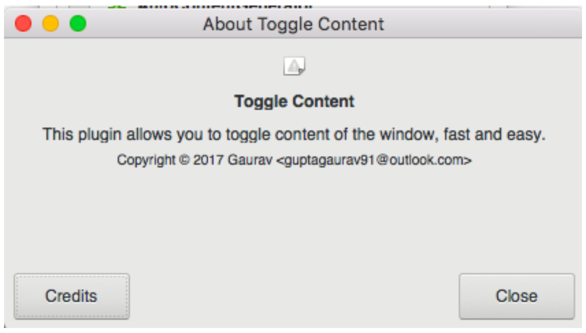
Fig. 1. Select preferences in Gedit editor

Further, once the preferences option was selected, a user was required to enable the plugin within the plugin section as shown in Fig. 2. User can also check the plugins “About Me”, which is shown in Fig. 3.



- [ ] **AutoContentGenerator** This plugin allows you to generate random c…
- [x] **Document Statistics** Report the number of words, lines and chara…
- [ ] **Example** An example plugin which clears the document.
- [x] **File Browser Panel** Easy file access from the side panel
- [x] **Insert Date/Time** Inserts current date and time at the cursor po…
- [x] **Modelines** Emacs, Kate and Vim-style modelines suppo…
- [ ] **Sort** Sorts a document or selected text.
- [x] **Spell Checker** Checks the spelling of the current document.
- [ ] **Toggle Content** This plugin allows you to toggle content of th…

Fig. 2. Enabling the plugin.



Toggle Content

This plugin allows you to toggle content of the window, fast and easy.

Copyright © 2017 Gaurav <guptagaurav91@outlook.com>

Fig. 3. About me for plugin

After the tool was enabled, it could be viewed under the tools menu section with the submenu name as “Toggle Content.” To further use the functionality of the tool, the keyboard shortcut – Primary Key (Command on Mac and Ctrl on Windows) +Alt+ 2 should be pressed or the option show/hide content from the toolbar should be used.

II. Deliverable 2: Python tool to capture images using camera on Mac

A Camera Capture Tool was developed for Deliverable 2. The objective of Deliverable 2 was to understand the API between Python programming language and the camera mounted on the computing device. One requirement for this deliverable was to capture images using the camera and to save it to some location so that it could be utilized at later point of time.

The Camera Capture Tool developed for deliverable 2 provided several additional features apart from capturing images. It allowed drawing rectangles on the image or changing the mode of camera. Building the tool helped gained a better understanding of the integration and usage of libraries like OpenCV 3.0, numpy developed for Python. The tool was developed in Python 2.7.

The additional features provided by Camera Capture Tool are as follows:

1. **Change camera modes:** The camera modes in the tool could be changed by pressing the keyboard shortcut key 'c' or 'C'. Change of mode means that the image could be changed from black/white to RGB mode. The tool had integrated several different modes. Fig. 4. displays all the modes that were available in the tool.

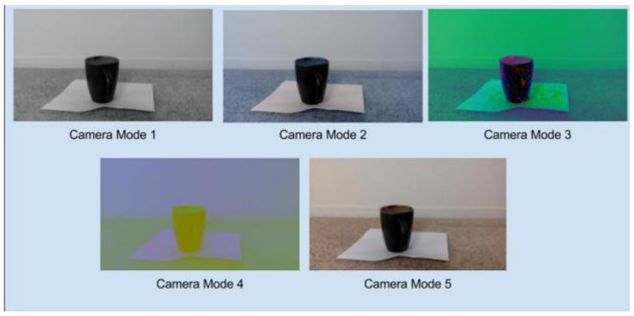


Fig. 4. Modes of camera

2. **Draw rectangle shapes on the image:** This feature provided the flexibility to draw rectangles on the frames being displayed. In order to draw a rectangle on the image, click on the mouse button and should not be released. Now, from the point at which the mouse click was pressed was considered to be the start point of one end of the diagonal of the rectangle. Releasing the mouse click would take that point as the other end of the diagonal. The rectangle would be

drawn such that one end of the diagonal was the point at which the mouse button was clicked and the other end being where the mouse click was released. Fig. 5 shows the with rectangles drawn on the black and white image.

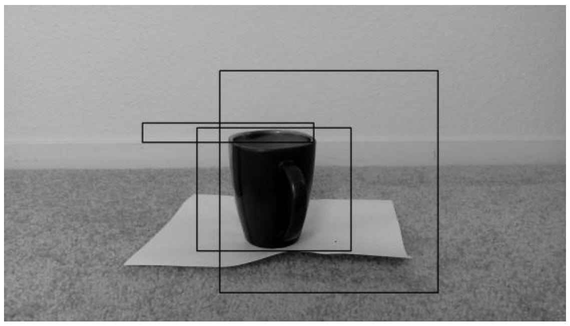


Fig. 5. Rectangles drawn on black and white image

3. **Save image**: It is possible to save the images currently being shown in the video buffer by pressing the keyboard shortcut key ‘s’ or ‘S’. All the images in different modes for this deliverable were saved using the keyboard shortcut key. By default, the images were stored on the desktop location.
4. **Remove rectangle:** Tool provided the ability to remove rectangles drawn in the images in two ways.

   4.1. Removing single rectangle: This feature provided the ability to remove the last drawn rectangle on the image. In order to remove the last drawn rectangle, user was required to press ‘l’ or ‘L’.

   4.2. Removing all rectangles: This feature provided the ability to remove all the rectangles at once by pressing the keyboard shortcut key ‘r’ or ‘R’.
5. **Exit the tool:** In order to exit the tool user could either press ‘ESC’ or ‘q’ or ‘Q’. Pressing the mentioned keyboard shortcut will quit the execution of the tool.

III. Deliverable 3: Head Movement Detection

To detect head movement, it was necessary to understand how to detect the facial region within the image. There are several algorithms which helps in finding the facial region in the image. I studied the research papers that considered the technique to detect the facial region using the algorithm based on neural network[3] and on combination of Ada Boost Algorithm and Haar-Based feature [4] [5]. The research papers following the technique based on Ada Boost algorithm combined with Haar-Based feature detection showed improved accuracy. In the Ada Boost algorithm, the combination of many weak classifier was combined to form a strong classifier. A weak classifier could be classified as one recognizing only a small part of the image. For example, classifying whether the image had face or not based on a fragment of image. In Haar-Based feature classification, there were particularly 4 Haar-Like feature also known as T-Feature as shown in Fig. 6. The face was divided into several T-shaped regions as shown in Fig. 7. The T-shaped feature helped in recognizing the features of a face like eyes, nose, cheeks, mouth, eyebrows, forehead. Haar-Like feature basically divided the face into several rectangle boxes and calculated the gray area which provide the Haar-Based feature value. The gray area selected for a face is shown in several sub images of Fig. 7.

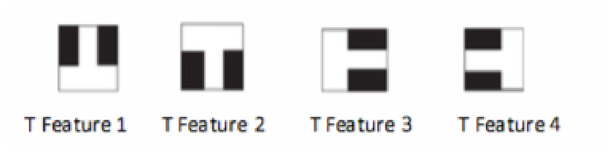


Fig. 6. New Haar-Like feature [4]

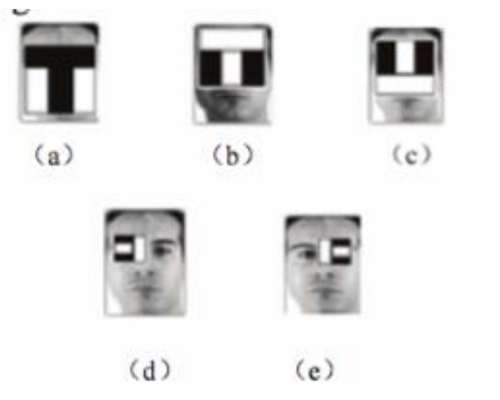


Fig. 7. The matching relationship between all Haar-Like feature and facial geometric features [4]

After the facial region was identified in an image, then consecutive frame of the images was monitored in order to track the special feature points on the face. If the location of the feature points was greater than some threshold value, then there was some movement detected in the current frames, otherwise there was no movement detected in the current frames [6] [7]. Both the papers on Lucas-Kanade(LK) algorithm specified that there were several facial features that are available to track in the image. But all the features could not be used in tracking the consecutive video frames and defining if there was a head movement or not. The reason for that, with the change in expression the feature point was likely to change. For example, if feature point was center point of the eyes, then if the eyes were shifted to look in some other direction while keeping the head still. It would suggest that there was some head movement. Another example could be tracking of feature points on mouth or lips. If the person laughed or changed expression while keeping the head still would suggest that some head movement occurred. Since, most of the feature points that were being tracked could suggest that there was some head movement even when there was no movement at all. Choosing these feature points to identify the head movement would result in bad classifier of the head movement detection. But not all feature point was not bad, LK algorithm suggested that tracking the nose was the best feature in order to identify if there was any head movement or not. The reason is that if the center of nose was being tracked then, it did not change with change in expression. Based on this approach, the Head detection algorithm was built.

In the current head detection algorithm, the center of the facial region, which was in proximity to center of nose was being tracked and if the movement of the feature point exceeded certain threshold then it classified as movement detected based on whether the movement was in top-down direction or left-right direction. The message was displayed in the left corner of the window specifying the type of gesture detected. Here the type was basically left-right or top-down. The reason for including a threshold value was that even a small negligible movement of face would trigger a change of pixels in the image and which would be identified as a movement of head which would not be detected by the human naked eyes. This would lead to an impression that the algorithm was classifying incorrectly. To prevent this behavior, threshold value was included, so that if the movement of head in consecutive frames exceeded certain threshold only then movement of head would be identified.

IV. Deliverable 4: Map Text to Head Movement

With the help of Deliverable 3, head movement could be identified. But it was further required to classify the head movements, i.e., whether the head movement was either left or right when head movement gesture was detected as left-right. Also, it was required to classify whether the head movement was up or down, when the gesture detected was top-down. To classify the head movement further, the same trick of LK algorithm was used. The position for the nose feature point of the facial region depicting the directions left, right, up, down and normal (looking straight) was calculated over couple of frames. Average position of each direction was taken individually. Once the average value was calculated properly, the head movement was identified properly based on the 2-norm distance of feature point. If the feature point for the current frame was close to feature point of left direction than any other direction, then the position of the head was identified as turned towards left. Here the user's average feature point for head movement towards left was already available based on the calculation of the average of frames when the user was looking towards left. Similar technique was followed to identify the movement of head for all the other directions namely, right, up, down and looking straight. Once the head movement was made, it was shown in the top left corner of the window. After the head is shifted in any direction rather than looking straight, the system waited for the user to shift the head toward straight or normal direction. The same message was also displayed on the top left corner of the window, so that user could be notified what the system was currently processing.

Once the user moved the head towards any one of the directions, the option of characters available in that direction were available to select, but before the user would select another head movement, the user was required to move the head towards the normal position also known as the looking straight position. Initially when no choice was made or after the character was chosen, all the choices are grayed out shown in Fig. 8.

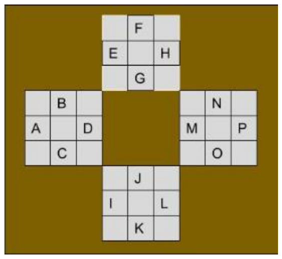


Fig. 8. Choice of characters available after or user has made a selection of character or starting fresh.

After the head was moved in any direction, the characters available in that direction were highlighted as shown in Fig. 9. Suppose the head was moved towards left then the characters available in the left direction were highlighted. Now, the user was required to return back to the normal position. After the user returned to the normal position, user could further move the head in any one of the directions and the character in that direction would be printed on the console. Similarly, if the user moved the head towards right, the choice of characters on the right-hand side were available to select. Head movement in the top or bottom direction enables the user to select the characters available in that direction. Note, here the user was allowed to select the characters ranging from A to P rather than A to Z. Additional capabilities of selection all characters, symbols, digits and other important punctuations would be added in future developments.

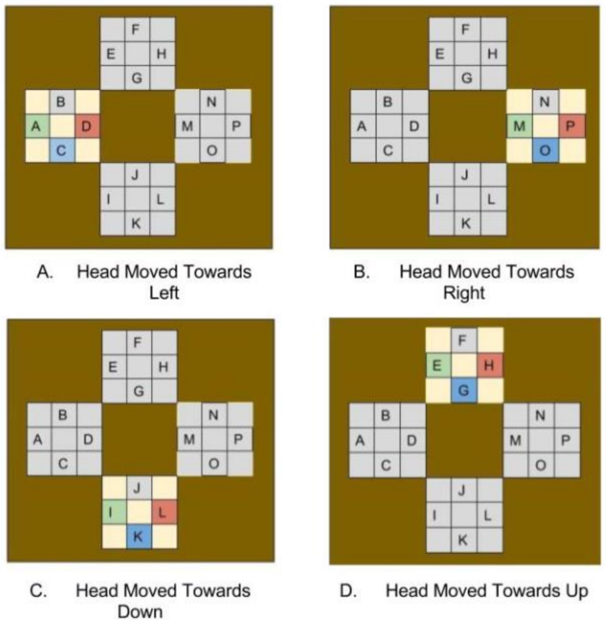

A. Head Moved Towards Left
B. Head Moved Towards Right
C. Head Moved Towards Down
D. Head Moved Towards Up

Fig. 9. Character choice based on direction of head movement

V. Conclusion

I have completed the preliminary steps required towards my CS 298. Completing these steps helped me to get a better understanding of the different fields needed for the completion of CS 298. I will further utilize the progress made in current semester as a part of the course CS 297 to develop a system which would provide a convenient way to enter inputs using the head movements.

In detail, I developed a plugin for Gedit editor. This is needed as a part of my project since the text generated using the head movements should be written in some text editor. Developing the plugin for Gedit editor provided a better understanding of how the commands could be interpreted to perform some action on the editor. For CS 298, I will use the knowledge gained while developing the plugin for Gedit editor. The knowledge gained would help in transforming the text generated using head movements and mapping the text to one of the open source text editor.

Further, I learned to interact with the camera of the Mac using the Python programming language. This will help in recognizing head movements by monitoring continuous consecutive frame of images. Later those head movements were utilized to map them to text characters as shown in Deliverable 4.

For CS 298, I will merge the potentially useful features developed during CS 297 and further increase the accuracy and robustness to map the head movements to text characters. I will also include faster accessibility to the key features like ability to go back, provide word suggestion, accessibility to punctuation and symbols.

References

[1] 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.

[2] 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.

[3] H. Rowley, S. Baluja and T. Kanade, "Neural network-based face detection," in *Computer Vision and Pattern Recognition, 1996.* In *Proceedings of 1996 IEEE Computer Society Conference on CVPR,* San Francisco, USA, 1996.

[4] S. Ma and L. Bai, "A face detection algorithm based on Adaboost and new Haar-Like feature," in *Proceedings of 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 2016.

[5] . C. Lv, T. Zhang and C. Lin, "Face detection based on skin color and AdaBoost algorithm," in *Proceedings Of 29th Chinese Control And Decision Conference (CCDC)*, Chongqing, China, 2017.

[6] Z. Zhao, Y. Wang and S. Fu, "Head Movement Recognition Based on Lucas-Kanade Algorithm," in *Proceedings of International Conference on Computer Science & Service System (CSSS)*, Nanjing, China, 2012.

[7] L. Jian-zheng and Z. Zheng, "Head Movement Recognition Based on LK Algorithm and Gentleboost," in *Proceedings of 7th International Conference on Networked Computing and Advanced Information Management (NCM)*, Gyeongju, South Korea, 2011.