



VIRTUAL KEYBOARD USING HAND DETECTION

Saurabh P. Jain, Abhishek Deshmukh, Abhii Shah, Mahendra Pawar
Department of Computer Engineering
V.I.I.T., Pune, India

Abstract: Traditional QWERTY keyboard provides all functionality necessary to use a computer but many people cannot make a versatile use of it i.e. they are not aware of all the shortcuts as well as several handy key-combinations used for specific tasks (e.g. ALT+ENTER displays properties of selected items, ALT+SPACEBAR displays system menu of an active window etc.). Sometimes user need only a specific set of keys for their use (e.g. For gaming etc.), in such cases rest of the keys available are of no use. Traditional keyboards are not re-configurable as per the need of user. Also use of traditional keyboard in multi-lingual environment is tricky job. The virtual keyboard we proposed uses only 2D webcam and no other hardware and a sheet of paper with keyboard printed on it. This paper addresses problems with traditional keyboard implementations and describes some unique and efficient techniques to solve these problems.

I. INTRODUCTION

We are developing a virtual keyboard using 2D webcam instead of any other specialised hardware. This keyboard will overcome some problems of traditional keyboard and serve as an add-on for it to make computing very easy and efficient for general users of computer. Traditional keyboards were not re-configurable. We cannot change the specifications of keyboard according to need of individual. We cannot use a key for some abstract or user defined purpose, those keys of traditional keyboard behaved in a specific pre-defined or say pre-programmed manner. In virtual keyboard every key will be configurable according to the need of user. There will be several profiles created by user and depending on profiles nature of every key will differ eg. If user is creating a profile called Internet then pressing the customised key will open a web browser, others can be customised accordingly to open an email service site or to enter user ID and password. User can create desired number of profiles according to ones need. In this way same key will behave in different manner depending on the profile selected. On further development the same keyboard can be used in multi-lingual environment and with various portable devices.

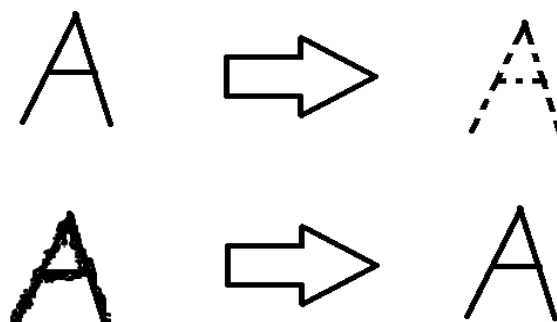
II. RELATED WORKS

A Virtual Keyboard can be implemented in several different techniques. In [1], a virtual key board (vkb) based on a true-3D optical ranging is presented. It is accurate and robust; however it requires a 3D optical imaging system which will increase the cost of the system.

III. WORKING

1. *Frame Extraction:* Using 2D webcam we are capturing the live stream of keyboard. From this live stream we will check for the hand detection. To check if the finger in all frames is same then it will ensure the finger is detected over a particular key. That frame will be extracted for further processing.
2. *Blur/Sharpening:* This process is manual and will be done by user of keyboard. Depending on the lighting condition user will choose between sharp and blur operation to be performed on frame. Why to blur/sharp image (frame)?

Fig. 1: Example of a Blur and Shar image.



As seen in figure above, if the image is too sharp then after conversion we will not get all pixels correctly that's why we need to blur the sharp image. And if image is too blurred then it will be difficult to recognise a letter on the key.

3. **RGB to HSV:** Extracted frame will be in RGB format. There are drawbacks of RGB color model like color loss. To overcome this, there is a need of conversion from RGB to HSV.

RGB to HSV conversion formula:

The R,G,B values are divided by 255 to change the range from 0..255 to 0..1.

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

Hue calculation:

$$H = \begin{cases} 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & C_{max} = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & C_{max} = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & C_{max} = B' \end{cases}$$

Saturation Calculation:

$$S = \begin{cases} 0 & , \Delta = 0 \\ \frac{\Delta}{C_{max}} & , \Delta > 0 \end{cases}$$

Value Calculation: $V = C_{max}$.

Thresholding: In thresholding, threshold values will be considered to separate the foreground and background colors. After thresholding we will get only the characters on the keyboard and rest of the color pixels will not be considered.

4. **Segmentation:** We are logically dividing the keyboard in 15 different section, each block belonging to single key. This we help us to detect the presence of finger over that particular key. While scanning the frame from top left to bottom right if finger is present in any block that block can be recognised. Logically divided keyboard is shown in fig below. Green color boxes are used to divide the keyboard.

Fig. 2: Example of a Segmented image



5. **Reference Comparison:** During the execution, when the live feed is clearly visible to user, user will set the reference using single button from our application as shown in fig. 3 below.

SET REFERENCE					SET REFERENCE	CURRENT READINGS				
2170	2110	1959	2115	1600		2312	2219	1906	2103	1756
2006	2175	1636	1745	1817		2226	2238	1735	1837	1870
1965	1818	1736	1934	1165		2078	1867	1815	2018	1286

Fig. 3: Example of a Reference setting

Then current values of each key will be set as a reference. When we get the frame with finger on it i.e. key press, that particular segment values will be compared with the reference value. This comparison will be based on pixels count in that segment. If the result of the comparison is less than 50% then it will be considered as a key press and respective key code will be generated. Theoretically result of comparison should be zero that means the key is completely covered with the finger. But practically it does not happen. This is the reason we need to keep the result values less than 50%. This will increase the efficiency for key press detection.

6. **ROBOT API (Java):** As we are in need of keyboard emulation, we need to use Robot API. This API is used to generate real time keyboard events as if given to OS directly from actual keyboard.
7. **Output:** After keystroke is detected and code is generated, we will get the output depending on the profile selected. We will be able to see the detected character in application.

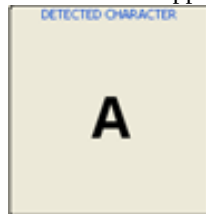


Fig. 4: Example of a detected character in application.

REFERENCES

- [1] H. Du, T. Oggier, F. Lustenburger and E. Charbon, "A virtual keyboard based on true-3D optical ranging", Proc. British Machine Vision Conference (BMVC), Oxford, pp.220-229, Sept. 2005.
- [2] Andrew D. Wilson, "Play anywhere: A compact interactive tabletop projection-vision system".
- [3] Jani Mantyjarvi¹, Jussi Koivumaki², Petri Vuori³, "KEYSTROKE RECOGNITION FOR VIRTUAL KEYBOARD", 2002.
- [4] Jakub Segen, Senthil Kumar, "3D hand pose estimation using a single camera".
- [5] Mathias Kolsch, Matthew Turk, "Keyboard without keyboards: A survey of virtual keyboards".