International Journal of Pure and Applied Mathematics
Volume 117 No. 20 2017, 713-725
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

Perceptive Interaction for Amputated Fingers (PIAF)

[1]B. Rajesh Kanna and [2]R. Rajalakshmi

[1]School of Computing Science and Engineering,

Vellore Institute of Technology,

Chennai, India.

rajeshkanna.b@vit.ac.in,

[2]School of Computing Science and Engineering,

Vellore Institute of Technology,

Chennai, India.

rajalakshmi.r@vit.ac.in

Abstract

In today's generation, smart mobile phones have come to be a part of the human body element almost. These phones and touch devices are of special interaction medium and it has essentially become an element and parcel of our day-to-day life; thus making the consumer to interact with the touch or smart phone in a better way. There is much importance of touch based devices in everyone's life especially in the life of the people with amputated fingers'. The target of our project is to use smart telephones without using touch characteristic and use facial features to choose applications, as a result making existence extra comfortable to the end customers particularly the people with amputated or no fingers. Here, we proposed an application which facilitates the interaction to people with amputated fingers. Also, provided the conceptual design and its implement procedures to help amputated person which will work with the help of eye tracking and glabella of their face.

Key Words: OpenCv, multi-window, glabella, android, eye wink.

1. Introduction

In today's technological world, without the use of mobile phones, gadgets, etc. people are like fish out of water. These mobile phones have been featured with with various functions and the evolution of these was termed as smart mobile phones. It could be run on various operating systems like Android OS, Windows OS etc.

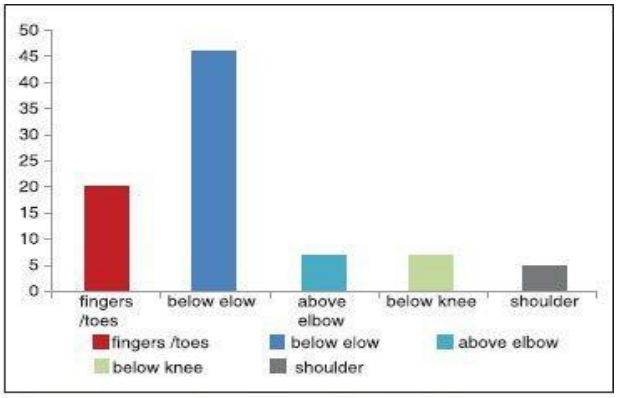

Figure 1: Amputation Risks [7]

India has become the world's fastest growing market in the era of mobile-based commercialism since its launch in the year 2010. The turnover of mobile transactions has grown up by a hundred and thirty percentages over the past year according to National stock exchange of India. The use of smart phones in India grew 54% throughout in 2014 and reached 140 million in range and then 651 million in number [9]. Again, the amount of tablets grew 1.7-fold in 2014, and reached 2 million then 18.7 million. Therefore, user-friendly phones or touch-based devices have become a part and parcel for all types of social communication. But the screens of these phones require finger usage as a medium of interaction and are important for triggering the applications in the phones. This upcoming modern technology cannot be utilized by the persons having no fingers or multiple amputated fingers since all interactions on these phones are mostly with the fingers. Thus, the necessity of using fingers as a medium for interaction rules out the access for the people with amputated and no fingers. The removal of a person's limbs or hands due to medical illness or some surgery is known as amputation. Figure 1, shows the amputation risks, and main causes for amputation is due to severe injury like burns, cancerous tumour in the muscle or bone of the limb, accidents, serious infection that doesn't heal with antibiotics or medicines and leads to thickening of nerve tissue. There has to be a solution for these people to use smart phones and be in the same pace as normal people and till date, no such effective solution has come up. Therefore, a system is proposed here after investigation to extend the usability of phones for the amputated persons. The system is proposed so that it will be simple enough to be used by these people without any difficulty so that they adopt it with open minds without a

second thought as they say-"*Necessity is the mother of invention".* In our project, we will be dealing with Android OS. It is a popular operating system which is used in most of the smart phones and because of its user- friendly interface to develop and use numerous applications, we have opted to use Android based smart phones for our project. The statistics shown in figure 1 shows the popularity of android based mobile phones in the year 2013 and 2014 respectively. The percentage of people using android is almost 6 times of the people using the famous Apple IO

2. Aim and Objective

The aim focuses on developing a system which will be well equipped with all the features of a usual smart phone as well as it will have ways to interact without using fingers too, specially designed for the people affected with amputated or no fingers. We will be using facial features for this purpose. Thus, we will make these people independent.

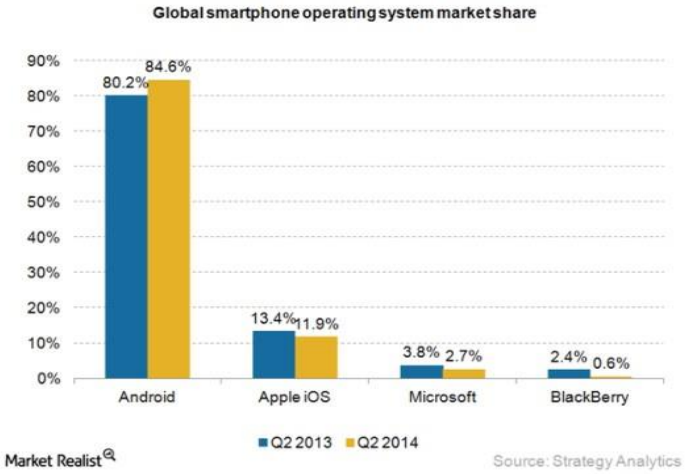

Figure 2: Popularity of Android Operating System

One of the common features of smart phones nowadays is facial recognition and it is gaining popularity. The face recognition is helpful for security purposes and has advantages over fingerprints and biometric because it is a non-contact process. The distance is not a constraint in face recognition as it can be captured with the camera from a distance. Our main goal is to look for an ideal mode of interaction with the phones which does not involve touch interactions and which is efficient at the same time.

3. Literature Survey

Literature survey includes a lot of research works which has face recognition techniques and algorithms to achieve the methods to detect facial features efficiently and robustly. Krlak et al. [6] used a vision-based human computer interface, it detects voluntary eye-winks and interprets them as control

commands and described methods like template matching techniques using Haar- features used in image processing. The detection of eye- wink was accomplished with a four step algorithm which consisted of detection of the face, eye-region extraction, eye-blink detection and eye-blink classification. There was a small constraint in this algorithm like the detection of voluntary and involuntary eye blinks and this was also resolved using the concept of threshold. If the duration of detected eye-blink is more than 250ms and less than 2s, then it is regarded as a voluntary eye blink and the rate of accuracy rate came out to be 99%. The system was a perfect until the time it was tried and tested with the persons with disability. The users required strenuous training sessions which reduced the popularity of the interface. Anand, B et al. [1] said that to use facial expressions for interactions especially with smart phones, tablets, etc. what is needed is efficiency. They presented a use-case scenario of eBook reader where the facial expressions were used to control the device i.e., is the eBook reader while accessing all its functions. A facial expression recognition system was introduced comprising of a detection module and an Action unit (AU) for mapping emotions of the facial features. The accuracy was quite good and the constraints were if the persons had facial hair, glasses, etc. then the detection had problems. Nose was also one of the facial features which were an efficient method of interaction because of its pointed structure. Dmitry O. et al. [3] used nose as facial feature and the face-tracking was carried out based on tracking a convex shaped nose feature. The nose became efficient because it was proved as the feature which could be tracked robustly. For the tracking of the nose, template matching technique was used which involves scanning a window of interest with the peephole mask and comparing each thus feature vector with the template vector. The feature template was made using this technique and the accuracy was 90% on this approach. Sometimes, image rescaling technique was used for nose detection. Esaki, S et al. [2] used eye blinks for quick menu selection on a communicator which was developed for the disabled persons. The input for the communicator was selected menu was executed using the pupil area which were obtained from the custom made image processor for eye-gaze detection. The voluntary and involuntary blinks were distinguished by the difference of their duration by giving a proper threshold and the accuracy results was 100%. Kanna, B et al. [10] suggested a device which would run with the help of face detection, eyes tracking, nose as a pointer for navigation and eye wink as a means of selecting the mobile application. But, since we are working on computational aspects which require efficiency of the system as well, there were little problems while tracking so many features all together and mapping it required more time. The tracking of nose was a problem as there was disparity in tracking the nose and the eyes simultaneously. Since the device has a shorter area as it is a phone and not like the usual laptops or desktops, we will minimize the number of tracking features and reduce it to tracking of only eyes and glabella as a method of using that device. Glabella is the part in between the eyes which can be tracked easily while tracking the eyes.

Thus, we will minimize the number of tracking features which will reduce the computational complexity of the device. Use of glabella is a new modification introduced in our system.

4. Conceptual Design

The overall conceptual design of the proposed work is divided into four phases and are mentioned as follows:

- Design of multiple windows in touch/smart phone.
- View location mapping of front-end camera.
- View to and Screen display view.
- Eyes Tracking and locating Glabella from front- end camera view.
- Eye wink tracking for event indication.

We're focusing more on looking for an alternate solution that does not involve finger interactions. So, face attention will also be an ultimate replacement considering the fact that it does now not require touch as an interaction medium. Furthermore, the facial snap shots will also be captured from a distance without touching the person being identified. This does not require interacting with that character. The next points provide a short view of what we want in the gadget without making use of finger interaction:

- Software system development involving the detection of feature points on face.
- Provide response according to feature response changes.
- Minimize interaction time.
- Work on any type of lighting condition.

Multi-window is a feature in smart phones which allows running two applications simultaneously on the screen. This feature is limited to supported mobile applications but does a great job of bringing true multitasking to the Smart-phone. The multiple windows decide which windows are visible and also perform transitions and animations of the window when opening or closing an app or rotating the screen.

From figure 3, we can see that the multi window will be used for maintaining one window for monitoring the face of the user and it is named as VIEW 1: Front end camera window. The other will be the actual workspace where all the Apps are available and it named as VIEW 2: Screen display window.

Coordinating the movement of the facial feature along with pointer on the menu to opt the prefer application is referred here as Mapping. Then the touch menu view is offered which is navigated with the aid of glabella by tracking the glabella coordinates from the face. The glabella motion is coordinated with the movement cursor keys like up, down, right and left. The path keys will give the position of the application with which we want to work with. We already know that the eye is the upper region of the face. One eye is detected

first and then, by using face symmetry, the other is detected. In this way the eye pair is successfully detected.

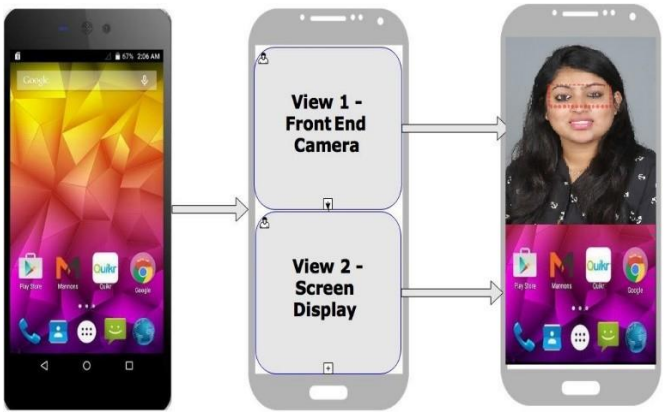

Figure 3: Provisioning of Two Views on the Touch Screen

Eye tracking is very essential in order to detect eye blinks accurately. Template matching is done here. From figure 4, we can see that eye wink will be used for selecting the mobile applications. Successful eye tracking leads towards accurate eye wink detection. If tracked eye is lost or eye is not working properly then the false detection rate and missed blink rate increases rapidly.

Figure 4: Emulation of Touch Tapping from Eye Wink

5. Experimentation

The face detection, glabella detection and eye wink tracking were accomplished with the help of Viola-Jones algorithm and all these implementations were carried out in Android Studio framework embedded with Open Computer Vision libraries. To execute this, a minimum of 10 GB disk space is recommended for Android Studio & OpenCv. Further, the Java Development toolkit with the version 8/7 is preferable. Also for the size of the Main Memory which makes the implementation faster, we preferred 8

GB in our implementation. In our implementation, we have used some of the Android Studio classes and few are listed below:

- android:camera
- android.hardware.camera2
- android:surfaceView
- android:supportsPictureInPicture
- android:gravity, android:minimalSize
- android.view.DropPermissions
- android:resizeableActivity

These android functionalities are available as open source for developers to develop applications which can be further used for marketing in the android market. Some of its features include direct manipulation on the mobile device using touch inputs, full optimization for mobile devices, multitasking of applications, native support for multi-touch etc.

In OpenCV, we have handled the following classes:

- Class Imgproc
- Class CameraBridgeViewBase
- Class CascadeClassifier
- Class JavaCameraView
- Class FeatureDetector
- Class OpenCVLoader
- Class BaseLoaderCallback
- Class VideoCapture
- Class Mat, Class Rect

It is an open source computer vision and machine learning software library which was mainly built to provide a common infrastructure for computer vision applications. Some of its features include motion tracking, facial Recognition systems, gesture recognitions etc.

: Integration of Android SDK with OpenCV Library

To experience the functions of OpenCV in android, we have integrated Android and OpenCV. The architecture of Android has 4 layers. Linux kernel is the very bottom layer which directly interacts with the hardware components. It also includes few components such as display driver, camera driver etc. On top of that we have libraries that include Android runtime such as Dalvik Virtual Machine, Core libraries. On top of this, we have application framework which felicitated the activity manager, window manager etc. Finally, the application layers are available wherein the user can execute their applications. The integration of OpenCV functions is happening on the android runtime which could be considered as the second layer of the android architecture.

The Dalvik Java VM is the key component in the Android Runtime

Environment. It enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. In the same way, it allows the OpenCV classes and functions to create instances of their own in the Dalvik Java VM and run them. The OpenCv and the Android integration require 2 steps and are mentioned below and the figure 4 illustrates this integration.

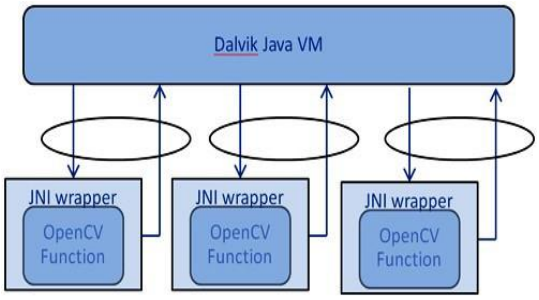

Figure 5: OpenCV and Android Integration

Multi window feature in our system is implemented through various APIs namely:

android: resizeableActivity which sets attributes in our manifest's <activity> or <application> node to enable or disable multi-window display. It also enables the user for split screen and freeform modes.

android: supportsPictureInPicture sets attributes in our manifest's <activity> node to indicate whether the activity supports picture-in-picture display. This attribute is ignored if android:resizeableActivity is false.

android: defaultWidth gives default height of the activity when launched in freeform.

mode.android:defaultHeight gives default height of the activity when launched in freeform.

mode.android:gravity gives initial placement of the activity when launched in freeform. modeandroid:minimalSize gives minimum height and minimum width for the activity in both split-screen and freeform modes.

requests.android.view.DropPermissions gives token object responsible for specifying the permissions granted to the app that receives a drop.

View.startDragAndDrop() New alias for View.startDrag(). To enable cross-activity drag and drop, pass the flagView.DRAG_FLAG_GLOBAL. If we need to give URI permissions to the recipient activity, we pass the new flagsView.DRAG_FLAG_GLOBAL_URI_READ or

View.DRAG_FLAG_GLOBAL_URI_WRITE, as appropriate.

View.cancelDragAndDrop() Cancels a drag operation currently in progress. Can only be called by the app that originated the drag operation.

View.updateDragShadow() Replaces the drag shadow for a drag operation currently in progress. Can only be called by the app that originated the drag operation.

For Setting up a general-purpose Android development environment, first the lastest version of Android Studio can be downloaded from the developer site. Before launching Android Studio, the computer system is required to be installed with Java JDK. After launching Android Studio, JDK path or later version in android studio installer is to be mentioned. Then, OpenCV library is imported to Android Studio by choosing the java folder. Updating of build. Gradle is required. Module dependency must be added. Libs folder must then be copied under the SDK to Android Studio.

: Customization and modeling of workspace window

As per the conceptual design, we are supposed to provision two windows with equal size namely Front end camera (view 1) and Screen display (view 2). However, the resolution of the android screen depends on the developed mobile application. So, we need to normalize both the views within the range of minimum and maximum. Thus, the normalization felicitates any image independent of the size and resolution. Here the normalization is done between [0, 1] with a decimal point up to 3 or 4 decimals.

: Modeling of the screen display (view 2)

Modeling of the screen display is mandatory to logically dividing the screen display to accommodate maximum number of app icons as well as the swift way to search and execute the selected app. Hence, an appropriate data structure could be employed for storing and executing the mobile applications. Here, we opted quad-tree because it is the best data structure to geometrically locate the icons in the workspace. A quad-tree is a tree data structure in which each node consists of exactly four children internally. It is mainly used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. These regions may be square or rectangular, or may have arbitrary shapes. The icons in the screen display are triggered in a fraction of seconds. In order to select a particular icon, the event on it is invoked explicitly. The Android standard "View" class supports touch events. In our implementation, we are focusing on triggering the icons using "eye wink" which is equivalent to touch. The corresponding coordinates from the eye wink could be then mapped to a touch event in android. Reacting to touch based events is made possible by running a code that overrides the touch to wink.

: Customization *of quad-tree for storing and retrieving mobile application*

Figure 5 shows a simple representation of hierarchical partition of quad-tree. Here, the bigger rectangle represents the size of the screen display (view 2) and this quad-tree could be partitioned until the size of the partitioned region is equivalent to the dimension of the mobile app icon. Advantages of using a quad-tree includes easy and fast access to a particular region of the window, all the icons put without any difficulty, easy interpretation since behaves like a tree.

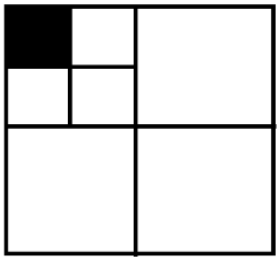

Figure 6: Quad-Tree Data Structure

We used three labels for representing the quad-tree. Region with background colour represents the full availability of space to accommodate a mobile application. Region with black represents no space to accommodate the mobile application. Regions shaded with gray represents the partial availability of free space.

Division of the quadrant: During our search for the icon, we may need to divide our uniform square (workspace) into sub-squares. Given a node in the quad tree, it is divided into quadrants. Tree Traversal algorithm is as follows:
Step 1: Store the size of the root node and last node.

Step 2: Check if any icon is found in any of the four quadrants.

Step 3: If icon is found, check inside the quadrant up to the least size of the node i.e. the size of the icon. The algorithm is then recursively called on the parent of the given node.

Tree Traversal: Given a node, the tree Traversal generates the leaves of the node and then repeats the process on each of the leaf nodes. Quad-tree is shareable by both the windows. The number of leaf nodes will give us the number of icons in the screen display.

: Eyes detection and eye wink tracking

For eye detection and eye wink tracking, we have referred the OpenCV libraries wherein the Viola-Jones algorithm for face detection is utilized which are the first ever real-time face detection system. There are three operations working in this algorithm: the integral image for feature computation,

Adaboost for feature selection and an additional cascade for efficient computational resource allocation. The cascade object detector uses the Viola-Jones algorithm to detect people's faces, noses, eyes etc. Here, we have used eye detection. This eye detection returns the rectangular window bounded on the eyes. This window will keep track of eyes' as well as wink. From this rectangular window, it is easy to identify the glabella space. Figure 6 shows a real time tracking

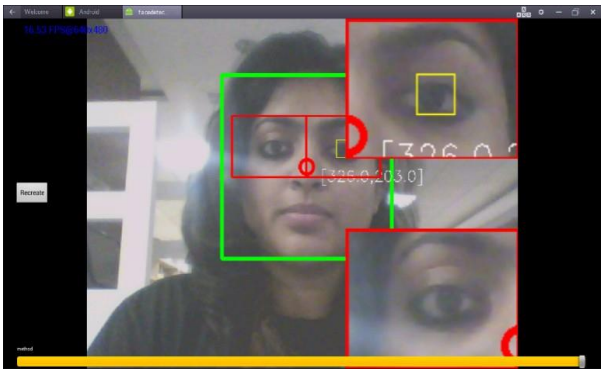

Figure 7: Face Detection with Coordinates

The rectangle's top left corner is located implicitly in the glabella region. So, by tracking eye we can easily track the galabella too. It reduced the tracking of nose in the earlier methodology [10]. Since the provision views are normalized, the location of the glabella can be easily mapped to another view. As soon as wink is encountered, the mapped coordinate of the glabella location on the screen display will trigger a touch event on the selected mobile application.

6. Conclusion

In this work, the screen space management was done with the help of quad tree method which does the spitting of region into equal spaces. However, for some cases, it may lead to the left-out of some of the spaces around the boundary. So, instead of quad-tree, yConvex hyper graph Decomposition method can be replaced to efficiently utilize all the spaces [11]. To conduct this experimentation, we have visited a non-profit organization ANBAGAM under the Manomani Trust. Through them we were able to meet a person with amputated hands. The information provided by that person has helped us in developing our project one step closer to our goal.

References

[1] Katz James E., Mark A., Perpetual contact: Mobile communication, private talk, public performance. Cambridge University Press, (2002).

[2] www.biomed.brown.edu/Courses/BI108/BI108_2003_Groups/Hand_Prosthetics/stats.html

[3] Indian Journal of Burns | Published by Wolters Kluwer.

[4] Amputation: //en.wikipedia.org/wiki/Amputation.

[5] Anand B., Navathe B.B., Velusamy S., Kannan H., Sharma A., Gopalakrishnan V., Beyond touch: Natural interactions using facial expressions, IEEE Consumer Communications and Networking Conference (CCNC) (2012), 255-259.

[6] Esaki S., Ebisawa Y., Sugioka A., Konishi M., Quick menu selection using eye blink for eye-slaved nonverbal communicator with video-based eye-gaze detection, 19th Annual International Conference of the Engineering in Medicine and Biology Society 5 (1997), 2322-2325.

[7] Gorodnichy D.O., Roth G., Nouse 'use your nose as a mouse' perceptual vision technology for hands-free games and interfaces, Image and Vision Computing 22 (12) (2004), 931-942.

[8] Królak A., Strumiłło P., Eye-blink detection system for human computer interaction, Universal Access in the Information Society 11 (4) (2012), 409-419.

[9] Kanna B.R., Kar D., Felice S., Joseph N., Intuitive Touch Interaction to Amputated Fingers, In Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16'), Springer International Publishing (2016), 387-397.

[10] Kanna B.R., Aravindan C., Kannan K., Image-based area estimation of any connected region using y-convex region decomposition, AEU-International Journal of Electronics and communications 66 (2) (2012), 172-183.