DETECTION OF FINGERS WITH A DEPTH BASED HAND-DETECTOR IN STATIC FRAMES

by

SANJAY VASUDEVA IYER

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements for the Degree of

MASTER of SCIENCE in COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2013

Acknowledgements

I would like to convey my heartfelt gratitude to my supervising professor Dr. Vassilis Athitsos for giving me the opportunity to work on a research project under him. I also thank him for his guidance, support and encouragement from the beginning of my thesis till the end. I wish to thank my committee members Dr. Gian-Luca Mariottini and Mr. David Levine for their interest in my research and for taking time to serve in my thesis committee.

I would also like to thank students of the VLM lab especially Christopher Conly, Zhong Zhang, Paul Doliotis and Alexios Kotsifakos, who have helped me in collection of data and also in my work whenever I needed them.

I would also like to thank the Computer Science and Engineering department of University of Texas at Arlington for providing all the facilities and infrastructure necessary to carry out my Master's studies.

I would like to thank my beloved parents, who have always inspired me to work hard. I would like to thank all my friends for supporting me through my graduate studies.

Finally, I would like to thank everyone whose direct and indirect support has helped me in completing my thesis on time.

August 12, 2013

Abstract

DETECTION OF FINGERS WITH A DEPTH BASED HAND-DETECTOR IN STATIC FRAMES

Sanjay Vasudeva Iyer

Masters of Science in Computer Science

The University of Texas at Arlington, 2013

Supervising Professor: Dr. Vassilis Athitsos

This thesis presents a method for a finger detection system. It is assumed that the user taps their fingers on a table and the camera is placed on the same table in front of their fingers. This setup is motivated by an application of analyzing the movement of fingers in patients engaging in physical therapy. Fingers are detected in static images, which is a more challenging task than detecting and tracking fingers in videos which are based on motion.

The Microsoft Kinect sensor has been used as the source to collect data, and it provides color and depth images at each video frame. OpenNI framework version 1.5 was used to capture data as it allows alignment between color and the depth frames. Detection of fingers is performed using two different methods: Template Matching and Principal Component Analysis (PCA). Additional information present in the image, such as skin color and depth data, is used to improve accuracy and efficiency. The depth frames are used to separate the foreground from the background, and also to provide information for detecting hands. A face detector is also utilized and the position of face is used as a reference to determine where the hands are located.

An additional contribution of the thesis is a graphical interface developed in MATLAB for annotating finger positions. This tool provides abilities for users to load various sequences of images and manually annotate the position of fingers in those images. Using this tool, we have annotated a large number of video frames, and these annotations have been used for training and testing the proposed method. In addition, these annotations remain as a valuable resource for future research on finger detection and tracking. For testing purposes, the MATLAB system also allows running the proposed method and measuring the accuracy of the results, based on the manual annotations. The thesis includes a comprehensive study on the effect of possible design decisions, as well as accuracy of user-dependent and user-independent settings.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

One of the most successful applications of image analysis and understanding has been detection. Detection involves identifying occurrences of a particular object, person or a feature. A related concept is that of Recognition, which involves detection and then matching them to personal or objects information stored in a database. Face detection has been a popular technique now for several years in commercial and law enforcement applications. Technologies available after decades of research have also made it possible for recognition systems of smaller objects with equal precision e.g. body parts such as the gesturing arm in an American Sign Language (ASL) Recognition System. The invention of motion sensing devices like Kinect by Microsoft and other 3D sensors like Primesense have bought depth frames into popularity amongst research in detection and recognition. Each pixel of the depth frame contains the Cartesian distance, in millimeters, from the camera plane to the nearest object at that particular (x, y) coordinate. The (x, y) coordinates of a depth frame do not represent physical units in the room; instead, they represent the location of a pixel in the depth frame. Open source libraries like OpenCV and OpenNI SDK also aid in solving Computer Vision problems.

Here, as part of research for study on patients undergoing physical therapy, a finger detection system was developed. Microsoft's Kinect for Windows has been used as a source for data, i.e., it provides RGB images and the depth frames. Data has been captured using the OpenNI 1.5 framework which enables aligning depth and the RGB frames. This alignment of depth and RGB frames is called Registration and is an inbuilt feature in OpenNI framework. When registration mode is turned on, the depth image is transformed to have the same apparent vantage point as the RGB image. This shift causes the depth frames to have border values of zero.

This thesis involves 3 parts – a finger detection system, a hand detector and the GUI annotation tool. The detection system provides 2 popular methods for detection of fingers of both hands – Template Matching and Principal Component Analysis (PCA).

In Template Matching, templates for all types of fingers were created (thumb, index, middle, ring and little). A template is an average over the training instances. The training samples for each of the fingers were created by extracting a piece of that particular finger from training images manually. They are then rotated as required to align them in a particular way and resized to a standard size. Average over the total number of such training examples results in a template and this was done for all the finger types. When the detection algorithm runs, one template is searched at a time for top 5 matches in an image. For this we use an algorithm called Normalized Correlation with multi - scale, which is explained in detail later. [Section 6.2.1]

PCA is a dimensionality reduction method. The results from template matching method along with a predefined number of eigenvectors and the average of fingers are used to detect fingers.

Skin Detection is used to filter the results of detection algorithms. The skin detection method used in this work is a histogram based skin detection, which is described in detail in another section [Section 6.1]. A threshold is used to decide the minimum probability of skin presence. The system allows user to select this particular parameter.

The user can analyze the detection and check for true positives and false positives. For this purpose the system provides annotation of the fingers.

The annotation tool is developed as a GUI in MATLAB. It provides abilities for users to load various sequences of RGB and depth frames. Once a sequence is loaded, an individual frame in the sequence can be selected by providing its frame number. The user can now annotate various fingers in this image. The tool uses different colors to annotate different fingers. The same can be used to annotate up to 5 objects in an image while collecting data for research purposes. The tool also provides options to mark base and tip of different annotations. The GUI tool also has provisions to save the annotated objects which can be used later.

Depth frames are used to avoid background. The system includes a face detector which was used to obtain the position of face and its depth. All pixels having greater depth than the depth of face are considered background. Depth frames also form an important role in detecting the hands. Hands are detected by considering all depth segments having lesser depth than that of the face. This is based on the assumption that hands are placed between camera and the face. The work here improves on a method devised earlier for detecting hands for American Sign Language Recognition [19]. While that method relied on detecting hands using motion, here the hands are detected on static frames.

Chapter 2

Related Work

There have been research papers in the past which discuss finger detection. Many of these works were either done for gesture recognition systems using motion or security systems. Here are some of the works which are closely related to work done in this paper.

One such work involved detection of finger tips based on depth information [2]. This method could robustly detect single fingertip regardless of the position and direction of the hand. With depth information of front view, depth map of top view and side view were generated. Thickness histograms are then used to segment the finger from the fist. This system could detect the fingertips with robustness and accuracy using a depth sensor. In this paper we use depth as an option to improve accuracy and the system does not rely purely on having a depth sensor. Another similar work was done to detect fake fingers based on skin elasticity analysis [6].

A paper from Yeo, Lee and Yim [14] presents a robust marker-less hand/finger tracking and gesture recognition system using low-cost hardware like webcam. This method would allow hand tracking despite complex background and motion blur. The method is able to translate the detected hands or gestures into different functional inputs which was used to interface with other applications. A system is implemented which begins by detecting face and eliminates the background. Next, skin color contours are extracted. For the skin contour, the convex hull and convexity defects points are then found. The maximum inscribed circle and the minimum enclosing circle are decided as the location of the palms. There are assumptions to decide shapes to be considered as fingertip. The largest convexity defect which fulfills the fingertip assumptions is detected as the thumb. Based on the vector between thumb and the nearest finger, left or the right hand is determined. Some implementations of this paper are designed to work only when palms are facing the camera and detect only the fingertips using contours. In the paper presented here, locations of the palms are not required while obtaining position of fingers and aims to detect fingers directly.

Software for hand detection was developed at MIT to create a graphical interface inspired by the movie "Minority Report". User stands in front of a screen with Kinect placed under it (camera facing the user). The user raises his hands and his palms get detected. Fingertips are obtained and tracked when the user moves the hand. User interacts with a graphical interface and can select an image by pointing at it. Image can be zoomed in or out by moving the palms closer or apart. The selected image can be discarded with the flick of the hand. It uses the Kinect sensor and the libfreenect driver to interface the Kinect on Linux. The graphical interface and the hand detection software were written to interface with the open source robotics package 'ROS'. This hand detection software displays the ability of the Point Cloud Library (PCL). This hand detection software is able to distinguish hands and fingers in a cloud of more than 60,000 points at 30 frames per second, allowing natural and real time interaction. Though this software detects fingertips, majority of the operations are done by the palm. The setup for the system described in this paper is that of a patient placing his hands on the table with palms facing downwards. So hand detection software could not be used.

Ravikiran's paper [3] involved a fast and an efficient algorithm for identification of the number of fingers opened in a gesture representing an alphabet of the American Sign Language. Finger Detection is accomplished based on the concept of boundary tracing and finger tip detection. The system does not require the hand to be perfectly aligned to the camera or use any special markers like input gloves on the hand. This is closely related to this paper but involves motion and work here is focused on static frames.

Another paper [1] provided methods to stop attacks on fingerprint-based biometric systems by presenting fake fingers at the sensor. It introduces a new approach for discriminating fake fingers from real ones, based on the analysis of skin distortion. The user is required to move their fingers while pressing it against the scanner surface, thereby deliberately exaggerating the skin distortion. Though this involves finger detection there is no camera involved and designed for a fingerprint scanner.

Kang's work [4] presents a methodology for hand and finger detection. The detected hand and fingers can be used to implement a non-contact mouse. This technology can be used to control home devices such as curtain and television. Skin color is used to segment the hand region from background and counter is extracted from the segmented hand. This paper is based on gesture recognition for device-free communication.

Leap Motion, a gesture-control technological company has released a commercial device called Leap which provides finger tracking [5]. This device can be used to interface with computers as an input device. Leap motion controller enables multiple finger tip detection and tracking of these multiple fingers. Work from Chan [10] also talks about multi finger detection for images captured using an IR camera and a single diffuser.

Another work [7] uses depth sensors, i.e., the Kinect sensor for human-computer interaction (HCI). This work focuses on building a robust hand gesture recognition system using Kinect sensor. To handle the noisy hand shape obtained from the Kinect sensor, a novel distance metric for hand dissimilarity measure, called Finger-Earth Mover's Distance (FEMD) is proposed. This is a gesture recognition system based on depth images rather than finger detection on images.

Another paper [8] introduces a fast decision tree based finger detection method where feature classification for hand gesture recognition and pose estimation is proposed. Training of the decision trees is performed using synthetic data and classification is performed on images of real hands. The presence of each finger is individually classified and gesture classification is performed by parts. The attributes used for training and classification are simple ratios between the foreground and background pixels of the hand silhouette. This is a very useful concept in Augmented Reality applications and its mainly focused on detecting gestures.

Some other research papers mention use of hand contours to search hand postures [9]. There is also an interface proposed in a paper [11] for mixed reality, which consists of a stereo camera to track the user's hands and fingers robustly in the 3D space.

Most finger detection papers are done for 2 reasons –first for devices which use fingers as an input like fingerprint scanners and the second being for gesture recognition [12][13][15] ,where motion or video is part of the input and not static images.

Chapter 3

Contributions

As discussed in the last chapter, there are many finger detection systems with different approaches proposed and implemented. This thesis provides a new approach to build a finger detection system. It aims at detecting fingers directly without the help of gestures or motion.

The first contribution of our work is a finger detection system which implements two detection algorithms to find fingers. When one of the algorithms is run, the system allows use of additional data like skin and depth to filter results.

The second contribution of this work is to provide depth based hand detector in static frames. Locating hands when the hand motion is involved is an easier task. Here we provide a simple method which can be used to detect hands in a static frame. The system also includes a face detector which detects position of the face and its depth. After locating the hands, those sub windows can be used to run finger detection algorithms.

The final contribution of this work is to provide a GUI annotation tool. Annotation of dataset can be a time consuming task. This GUI gives is user friendly and simple to use Annotation Tool. Annotations can be done by drawing polygons around the objects we intend to annotate using a mouse or a track pad. The annotation tool also provides options to mark the top and bottom of the annotated object. These annotations can be stored and later be used for testing the system.

Chapter 4

The Dataset

4.1 Source for Data Collection

Data was collected using the device Kinect for Windows. OpenNI framework version 1.5 was used to capture the data. OpenNI can collect frames at a rate of about 1-30 frames per second (fps). The exact fps is not fixed as it depends on the time complexity of the video capture application. The user may also select a target frame rate and achieving this frame rate depends on system configuration. The OpenNI viewer shows the depth frame on the left and the RGB frame on the right.

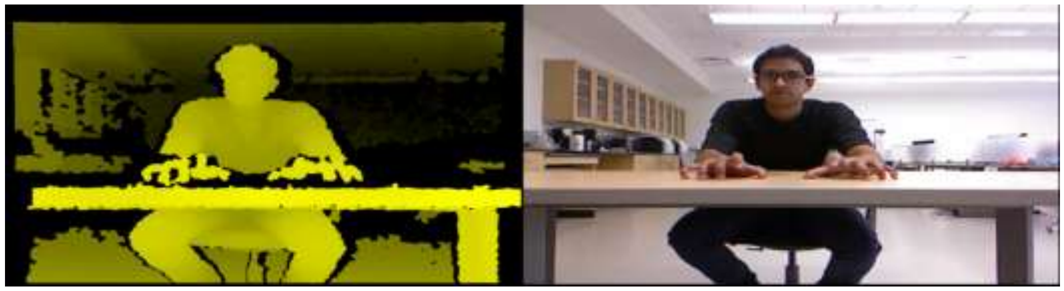

Figure 4-1 – OpenNI viewer

For data to be aligned, the registration mode should be on before capturing the data. Since the depth camera is aligned to RGB, border values appear as 0 for depth.

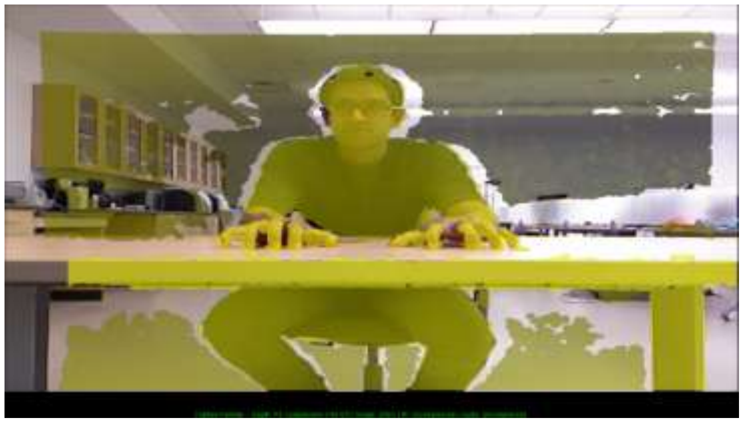

Figure 4-2 Registration mode on.

The output of the data capture is an ONI video file. Using the OpenNI framework, ONI files were converted to binary files for RGB and the depth streams. The program which converts the ONI files is made available with the code. These binary files were then converted to sequences of RGB and depth. The RGB stream collected were of the size 640*480 resolution.

4.2 Setup

The users sit on a chair with their hands placed on a table in with the Kinect. The Kinect sensor should not be placed too far or close from the person. The system treats objects behind face as background. An ideal distance for user to sit is somewhere between 1.5 - 2 feet from the Kinect with their hands placed on the table such that all the fingers are clearly visible.

The users were then made to move their fingers by tapping on the table to form a sequence. Each sequence had a certain number of frames which were determined by the amount of the time the capture was allowed to run and also the fps. The proximity of the palm to the table was varied in different datasets.

4.3 Converting the sequences into mat files

On collecting the images from Kinect, they had to be converted into mat files to be used in the MATLAB GUI tool. In MATLAB, a 4-D matrix of size 480*640*3*number-of-frames-in-sequence and 480*640*1* number-of-frames-in-sequence was created for the RGB frames and depth frames respectively. After collecting the filenames of individual RGB and Depth frames in the sequence, each of the images were then read and then inserted in the 4-D array at the same position as the frame in the sequence. All frames belonging to one particular sequence have to be put under the same 4-D array. This avoids errors in the GUI tool as it automatically picks up the depth frame at the position specified.

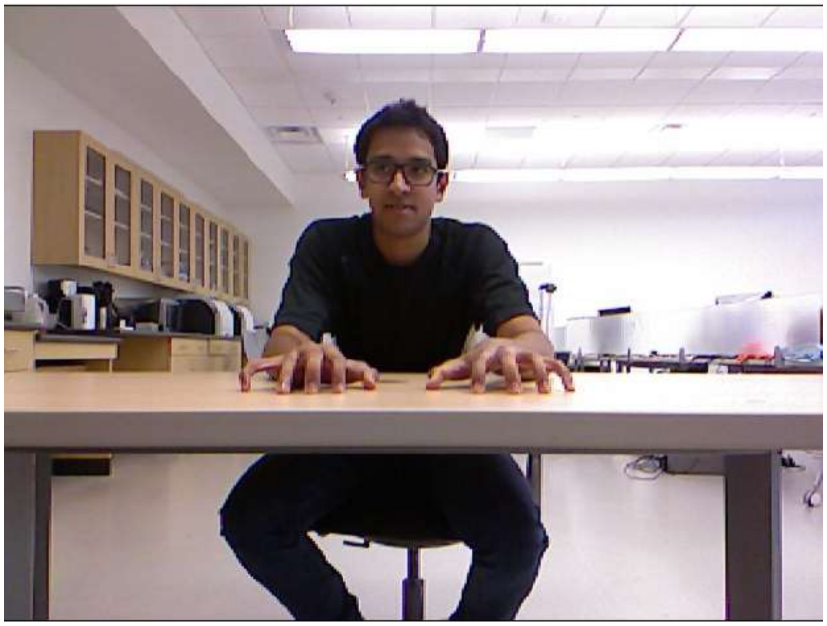

Figure 4-3 RGB data captured of a user.

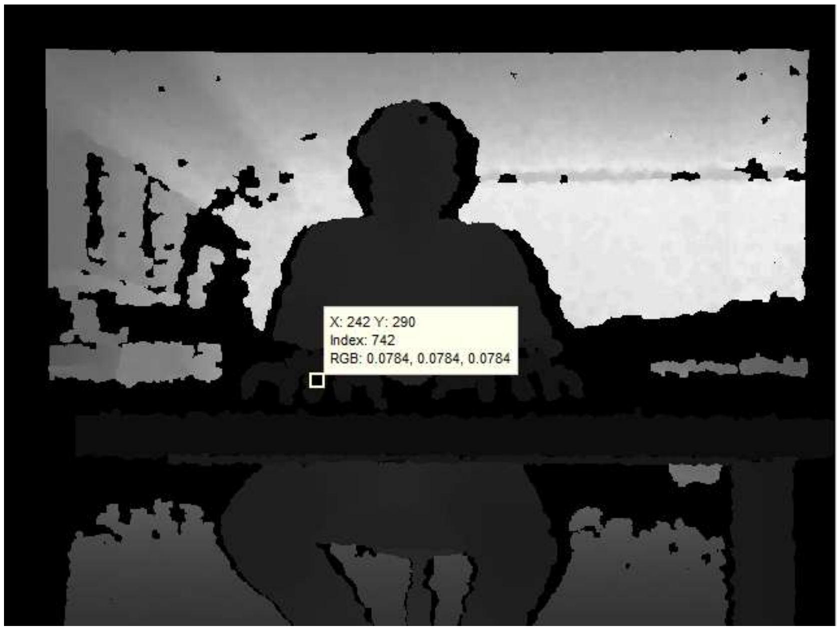

Figure 4-4 Depth Frame of a user

4.4 People in Datasets and No of Images in the sequence

There are 3 people in the datasets. Each person has contributed to 2 sequences of images. Below are the list of sequences (with their names) and the number of frames in each.

Table 4-1 Dataset (Person, Sequence and No of Files)

Person	Sequence1(No of Frame)	Sequence2(No of Frames)
Sachin	201	210
Shreyas	251	163
Sushruth	163	280

Chapter 5

GUI Annotation Tool

A MATLAB GUI was created to include all functionalities. This GUI makes all operations user-friendly and easy to use. Detection algorithms can be initiated by clicking on buttons and annotations can be made by dragging the cursor and creating shapes in the required manner. It also has functionalities like zoom and pan.

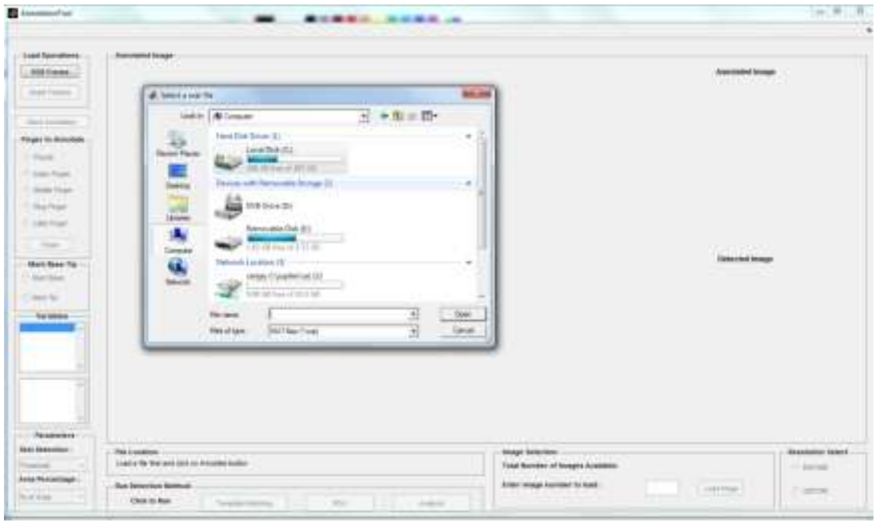

Figure 5-1 Initial Screen.

The initial screen allows only one operation – clicking of the ‘RGB Frames’ button. All the other fields and buttons are disabled. On clicking this button, a pop up window opens which can be used to select the file to be loaded. This file has to have a sequence of RGB frames of 640*480 or 320*240 resolution. The resolution can be chosen next by selecting the radio buttons of the ‘Select Resolution’ panel at the bottom right corner of the tool.

Once the file is loaded and the resolution is selected, the variables in the file loaded are shown in the top box under ‘Variables’ panel. On selecting one of these variables, the ‘Image Selection’ panel shows the number of images in the sequence selected next to ‘Total Number of Images Available’. If there are no images available it displays the message “*No Images Available”*. User may input the frame number to load under ‘Enter image number to load’ and click on ‘Load Image’ button. That brings up the actual image onto the main window (Figure 5-2). In case a wrong resolution is selected, the user can select the correct resolution radio button and then select the sequence under ‘Variables’ again to reload the sequence.

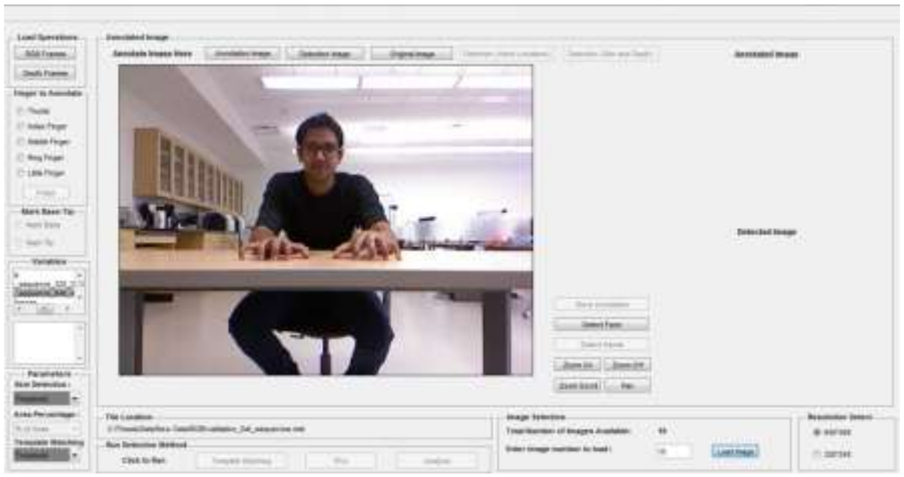

Figure 5-2 On Click of Load Image Button

The main window where in the user can view the image for annotation or after detection. This action of clicking ‘Load Image’ also brings up few other buttons. On top of the main window there are 5 buttons – Annotation Image, Detection Image, Original Image, Detection (Hand Locations) and Detection (Skin and Depth). If the user clicks on ‘Original Image’ button it brings up the original image onto the main window. The user can also choose to view -

1. ‘Annotation Image’ (Image on which annotation are done and have to be done)
2. ‘Detection Image’ (Image after detection using one of the methods along with Skin)

3. ‘Detection Image(Hand Locations)’ (Image after detection using the location of hands)
4. ‘Detection Image(Skin or Depth)’ (Image after detection using one of the methods, applying Skin Detection and removing background based on face depth information)

If no operation has been done all of them show the Original Image. To perform annotation the user must click on ‘Annotation Image’ button.

5.1 Annotation

The GUI tool allows annotation of fingers. The user can select the finger he wants to annotate in the panel ‘Finger to Annotate’ (Figure 5-2), a ‘+’ mark appears as a cursor on the main window, acts as a drawing tool to draw the polygon. The user can then draw a polygon around that particular finger by –

- Single click to mark a vertex.
- Dragging to make a line.
- Single click again to mark another side and complete the side.

To complete the entire polygon, user must connect the first vertex made. The tool cursor changes from a ‘+’ to a ‘O’ (circle) on clicking of which completes the polygon. The user can now drag the vertices to make the polygon bigger or smaller. Once the user feels the annotation is complete, clicking on ‘Esc’ on keyboard completes the annotation. Before user completes the annotation, the polygons appear blue in color and on completion (pressing Esc) are shown in different colors. User can move to annotating a different finger only when he completes annotating a finger completely. User can do this by selecting the radio button of the intended finger under the ‘Finger to Annotate’ panel.

On the left, user may modify the annotation by dragging the vertices to enlarge or compress the polygon. User draws polygon around the thumb and hits the Esc key to complete annotation and the polygon turns green. It cannot be modified now.

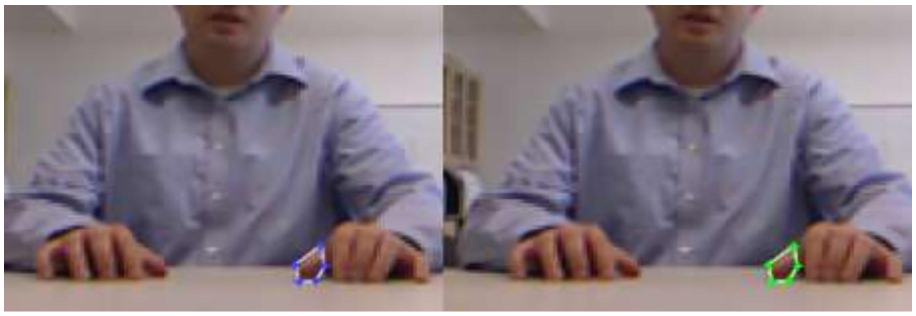

Figure 5-3 Drawing a polygon around the finger

On completing the annotation, different fingers are represented in different colors. Thumb annotation is shown in green color and index finger is shown in yellow color. Complete annotation of middle finger is shown in red, ring finger in blue and the little finger in cyan color.

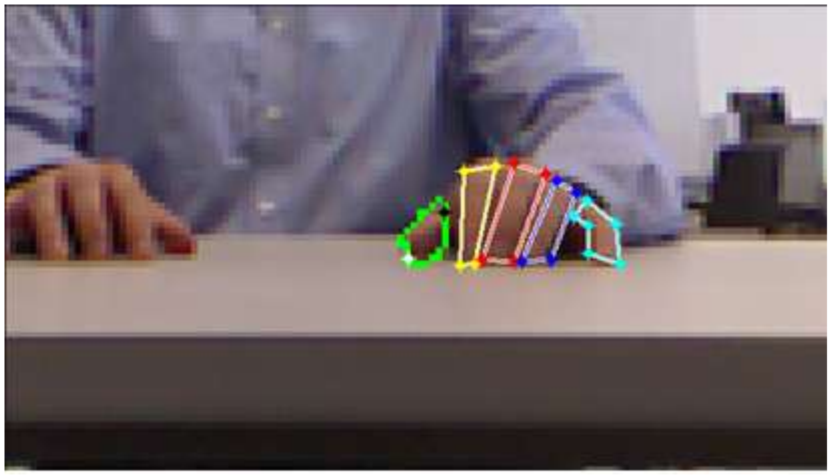

Figure 5-4 Fingers annotated in different colors

Later the user can also mark the base and the tip of the tool. To mark base and the tip, the user must click on the ‘Finish’ button at the bottom of the ‘Finger to Annotate’

Panel. It loads the annotated image on to the main panel, in which the completed annotations are shown. The user can select either of ‘Mark Base’ and ‘Mark Tip’ radio buttons to mark base and tip respectively for a particular finger.

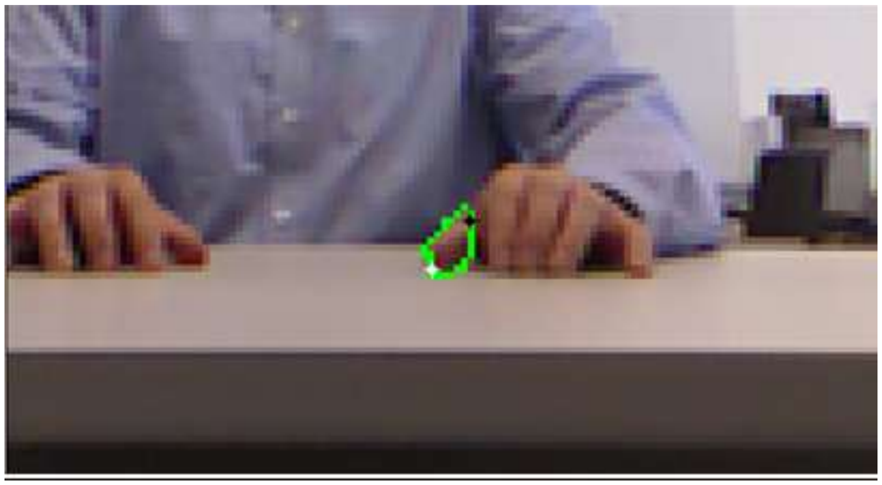

Figure 5-5 Marking Base-Tip

At all times the Annotated Image and the Detected Image appears on the right side of the tool. It is shown only when it is available, i.e., image loaded and at least one finger annotated and detection run at least once. Annotations can be saved by clicking on the ‘Store Annotation’ button and giving the file a name.

5.2 Other Operations

The user may also load the depth frames for an RGB sequence loaded. User clicks on the ‘Depth Frames’ button in the ‘Load Operations’ panel. A window pops up with the disk drives where user can select a mat file with the sequence. Care should be taken not to load wrong sequences with same number of frames as it will load the wrong depth frame. The variables in this file are shown in the bottom box of ‘Variables’ panel.User can select one of these sequences to load the depth file, which will be picked up by frame number specified for RGB image. In case, the user loads a wrong variable an error message “Depth sequence loaded is incorrect” is displayed.

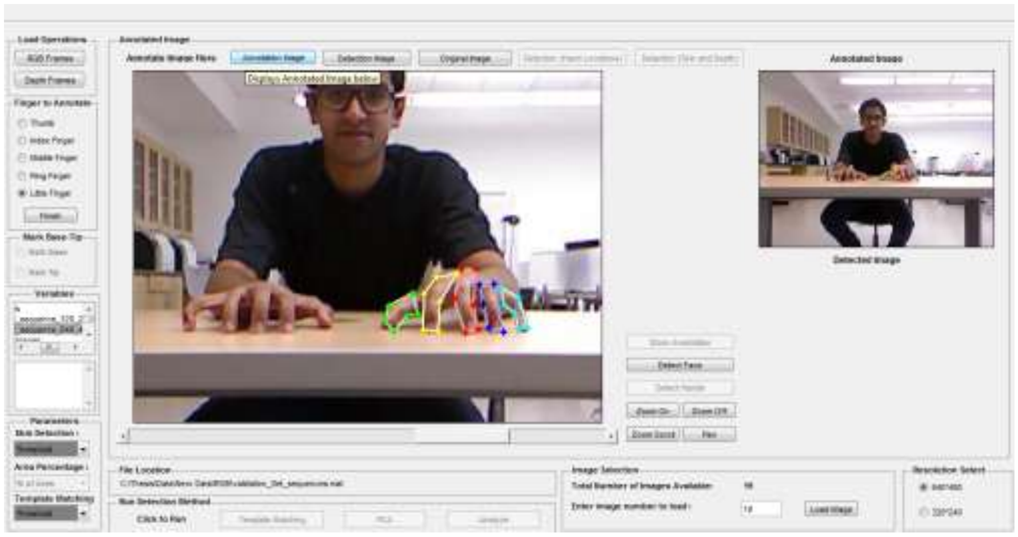

Figure 5-6 Annotation Image on the right

In the GUI tool, the user can zoom in on the main window. User can click on the 'Zoom Scroll' button next to the main window to make the scroll bar appear and use the scroll bar to perform zoom on main window. The user can zoom in on main window by either drag the scroll bar under the main window or click on the sides of the scroll bar. Pan automatically appears on the main window once user drag zoom scroll bar.

There are two more buttons 'Zoom On' and 'Zoom Off' which offer zooming on the main window. A magnifying glass appears on the main window on clicking 'Zoom On' button which can be used to zoom in. To turn off this zoom, user should click on 'Zoom Off' button. User may click on 'Pan' button anytime to bring in pan ability on to main window.

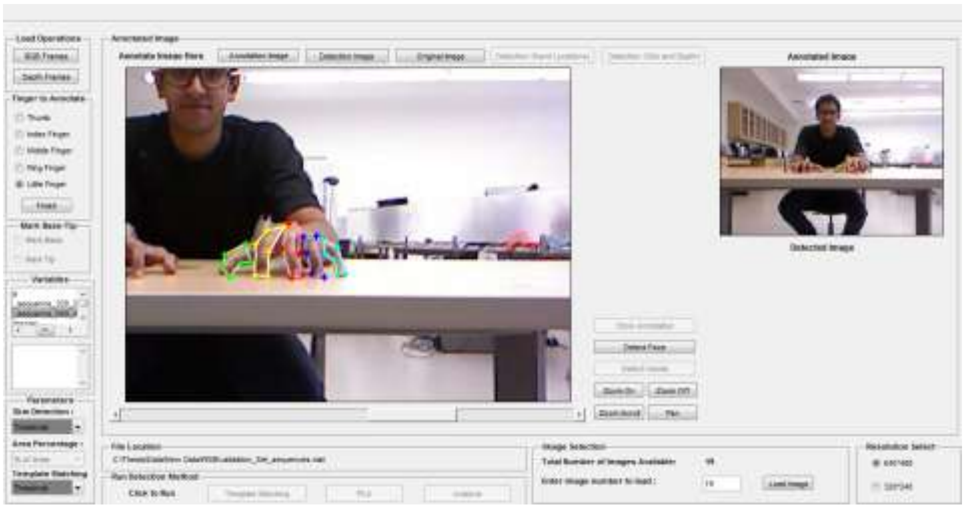

Figure 5-7 Zoom feature using scroll bar

The tool also has buttons to initiate the finger detection algorithms and analyze the detections. These are the ‘Template Matching’, ‘PCA’ and ‘Analyze’ buttons respectively. To enable these buttons, user has to select the parameters for these algorithms. The parameters used for these detection algorithms are present at the bottom left panel called ‘Parameters’. It includes 2 dropdowns

1. Skin Detection: This is the parameter of skin detection threshold. The dropdown provides values from 0.5 to 1. The selected value will be used as the threshold for all the detection operations
2. Area Percentage: This parameter is used for skin detection. Value denotes in percentage the amount of pixels with skin in a particular detection box. This values ranges from 50% to 100%. Also in annotation vs. detection, it denotes what percentage of pixels co-inside in a given annotation with detection.

To run the detection algorithms, there is a panel ‘Run Detection Method’. It includes 3 buttons – ‘Template Matching’ which initiates the template matching method on the given image with the templates obtained from training, ‘PCA’ which initiates the

PCA method on the results of template matching method and ‘Analyze’ which analyzes the detection made by an algorithm and returns the true positive and false positive count for the selected parameters. These results are displayed in the ‘Result Panel’. The ‘Analyze’ button is enabled only once one of the two detection algorithms is run.

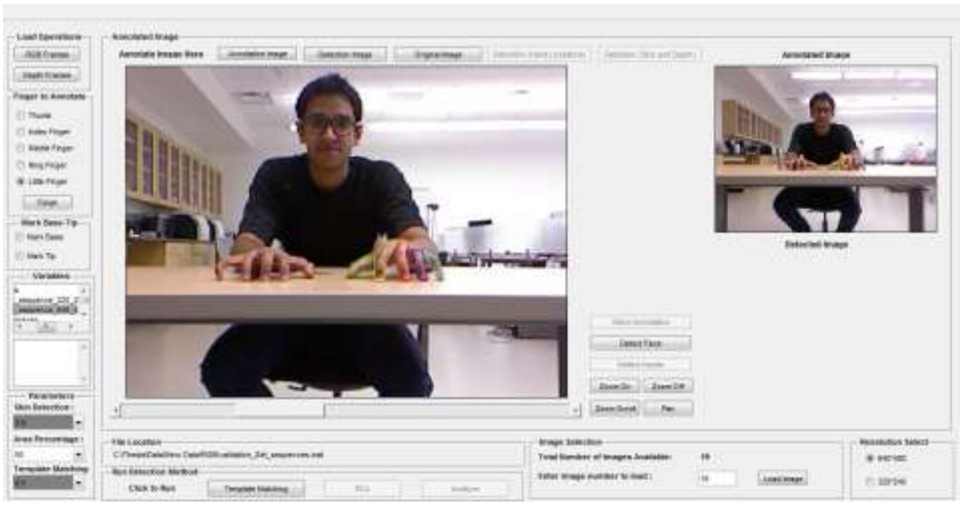

Figure 5-8 Running the Detection Algorithm

Chapter 6

Detection

In this chapter we describe the different detection algorithms we have used. The main focus of this work was to achieve finger detection. There are 2 algorithms used to detect fingers in this work – Template Matching and Principal Component Analysis (PCA). There are 2 other detection techniques used to support finger detection, namely skin detection and hand detection. We use a histogram based [18] skin detection to filter results from finger detection. Depth frames with skin data are used to detect hand locations and perform finger detection in the hand sub windows.

6.1 Skin Detection

Skin Detection refers to detecting the presence of skin pixels in an image. Training for skin detection involves extracting sub windows consisting only of skin pixels from many training images. Probability that we observe intensity values of RGB when we know a pixel is a skin pixel is calculated. The probability that we observe the same intensity values of RGB when a given pixel is non-skin is also calculated (from sub windows with no skin). This gives two values for all combinations of RGB values at the end of training

$$\mathrm{P(RGB \mid Skin)\ and\ P(RGB \mid Non-Skin)}$$

For every pixel in a test image, probability that a pixel is skin given its intensity values of RGB is estimated using Bayes Rule, i.e., P(Skin | RGB) probability that pixel is skin given its RGB values.

$$\mathrm{P(Skin|RGB)} = \frac{\mathrm{P(RGB|Skin)P(Skin)}}{\mathrm{P(RGB)}}$$

P(RGB) is calculated with P(RGB|Skin) and P(RGB|Non-Skin) values using

$$\mathrm{P(RGB)} = \mathrm{P(RGB|Skin)} * \mathrm{P(Skin)} + \mathrm{P(RGB|Non-Skin)} * \mathrm{P(Non-Skin)}$$

. There are 2 models to get P(RGB|Skin) and P(RGB|Non-Skin). They are Parametric Models and Non-Parametric Models.

6.1.1 Parametric and Non Parametric Models

In a Parametric Model, Gaussian method of estimation is used. It is assumed that the RGB colors are mutually independent. The 2 parameters on which the estimation is based are the mean and standard deviation of each of RGB streams. Given the mean and standard deviation for each color stream, P(RGB|Skin) can be calculated as

$$\mathrm{P(RGB|Skin) = P(R|Skin) * P(G|Skin) * P(B|Skin)}$$

P(R│Skin), P(G│Skin) and P(B|Skin) are the Gaussian probabilities of RGB streams given by the formula

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Here μ is the mean and σ is the standard deviation of the color band. A similar approach is taken to estimate P(RGB|Non-Skin) by calculating P(R|Non-Skin), P(G│Non-Skin) and P(B|Non-Skin) .

A non-parametric model is a model which does not depend on any parameters. It produces a more robust system than a parametric model. To overcome the Gaussian assumption of independence, we require large training data. Estimation of P(RGB|Skin) and P(RGB|Non-Skin) for each RGB is done explicitly. For this purpose we use the color histogram based model.

In Histogram based model we create histograms for skin and non skin. Training is done by counting how many pixels have color RGB for skin and non skin in the training samples. The count is stored in a histogram for those RGB values in histogram for skin and non-skin respectively. For example, if 100 pixels were skin and 200 pixels were non-skin for R=213, G=133 and B=144, the value in the skin histogram at (213,133,144) will be 100 and in the non-skin histogram value at (213,133,144) will be 200. The range of values for RGB is a decision which must be made before creating histograms. For example, if 8 bits are used to represent a color, RGB can have a range from 0 to 256. The histogram will be an array of 256*256*256 for both skin and non-skin. Creation of coarser histograms is preferred to creating histograms of large size like 256*256*256. Section 7.1.2 will describe how these histograms were created and used for the work here. For every pixel in a test image with value of RGB, the P(Skin|RGB) is given by

$$\mathrm{P(Skin|RGB)} = \frac{\mathrm{P(RGB|Skin)P(Skin)}}{\mathrm{P(RGB|Skin)} * \mathrm{P(Skin)} + \mathrm{P(RGB|Non - Skin)} * \mathrm{P(Non - Skin)}}$$

The values of P(RGB|Skin) and P(RGB|Non-Skin) are obtained from the skin and non-skin histograms respectively.

Since Histogram based skin detection is a non-parametric model and thereby more robust, we have chosen to use that as the method of skin detection over the Gaussian model. This method is combined with the use of threshold, a minimum probability of P(Skin|RGB) required in the image to be classified as skin. All instances of skin detection mentioned in this work use histograms for skin detection [16].

6.1.2 Creation of Histograms for Skin Detection

Histograms were created for skin and non-skin. The size of the histograms for both skin and non-skin was 32*32*32 arrays. Since the intensity values range from 0 to 256 and we have used 32*32*32 size for histograms, every intensity value for R, G or B will be divided by a factor of 8. The resultant values for the 3 color bands will be used as index to get the P(RGB|Skin) value in the histogram. We also need to decide on the P(Skin) and P(Non-Skin). We have used a value of 0.5 for P(Skin) and P(Non-Skin). Thereby the P(Skin|RGB) can be derived as

$$\mathrm{P(Skin|RGB)} = \frac{\mathrm{P(RGB|Skin)P(Skin)}}{\mathrm{P(RGB|Skin)} * \mathrm{P(Skin)} + \mathrm{P(RGB|Non - Skin)} * \mathrm{P(Non - Skin)}}$$

$$\mathrm{P(Skin|RGB)} = \frac{\mathrm{P(RGB|Skin)} * 0.5}{\mathrm{P(RGB|Skin)} * 0.5 + \mathrm{P(RGB|Non-Skin)} * 0.5}$$

$$\mathrm{P(Skin|RGB)} = \frac{\mathrm{P(RGB|Skin)}}{\mathrm{P(RGB|Skin)} + \mathrm{P(RGB|Non-Skin)}}$$

Therefore the value of P(Skin|RGB) is value from the skin histogram for RGB divided by the sum of values from skin and the non-skin histogram for RGB.

6.2 Finger Detection

6.2.1 Template Matching for Finger Detection

One of the detection methods used for finger detection in the work here is Template Matching. Template matching method requires templates, which are usually an average over the samples from the training examples. Then using Normalized Correlation, matches similar to the templates in an image are detected. The Normalized Correlation method implemented here uses the *normxcorr2* function inbuilt in MATLAB to do Template Matching. The use of Normalized Correlation or Cross-Correlation for template matching is motivated by the distance measure (squared Euclidean distance) [17]

$$d_{f,t}^2(u,v) = \sum_{x,y}[f(x,y) - t(x-u, y-v)]^2$$

(where f is the image and the sum over x,y under the window containing the feature t positioned at u,v). In the expansion of d^2

$$d_{f,t}^2(u,v) = \sum_{x,y}[f^2(x,y) - 2f(x,y)t(x-u,y-v) + t^2(x-u,y-v)]$$

The term $\sum t^2(x-u,y-v)$ is constant and if the term $\sum f^2(x,y)$ is approximately constant then the remaining cross-correlation term

$$c(u,v) = \sum_{x,y} f(x,y)t(x-u,y-v)$$

is a measure of the similarity between the image and the feature

The *correlation coefficient* normalizes the image and feature vectors to unit length, yielding a cosine-like correlation coefficient

$$\gamma(u,v)=\frac{\sum_{x,y}[f(x,y)-\overline{f}_{u,v}][t(x-u,y-v)-t]}{\{\sum_{x,y}[f(x,y)-\overline{f}_{u,v}]^2\sum_{x,y}[t(x-u,y-v)-t]^2\}^{0.5}}$$

where $\bar{t}$ is the mean of the feature and $\overline{f}_{u,v}$ is the mean of *f*(*x*,*y*) in the region under the feature. We refer to this equation as *normalized cross-correlation*.[20]

In order to create templates training examples had to be selected. To select the training examples, all possible hand positions were analyzed. The basic criteria for deciding the hand positions were based on the position of the palms and wrist with respect to the table. There are 3 possible cases here –

Case 1: Palm entirely on the table – This is a case where the person's palms lie entirely on the table. The fingers appear flat in these cases.

Case 2: Wrists placed on the table – This is a case where person's wrist is placed on the table and only the edge of the palms was in contact with the table. The fingers in this case appear completely.

Case 3: Palms not in contact with the table – This is a case where neither the person's palms nor the wrist is in contact with the table. The fingers in this case appear to be raised.

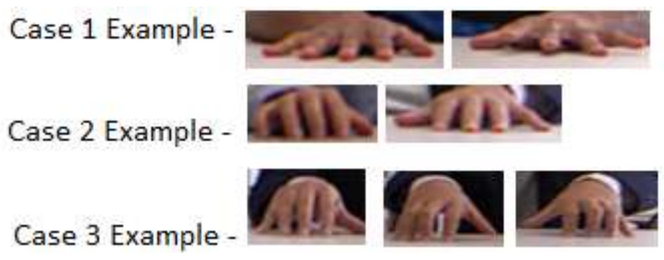

Figure 6-1Hand Positions

Based on this idea, 3 hand positions were decided. They are

1. Flat (Case 1)
2. Normal (Case 2)
3. Raised (Case 3)

The idea behind considering the 3 different hand positions is to assess different type of possible finger lengths, orientation and shapes. To achieve templates of all possible finger shapes – samples from the training examples in which fingers appear in 3 possible cases were collected. For example, consider the little finger. Templates of little finger from Case 1, Case 2 and Case 3 were collected. In order to consider the lengths multi-scale search is used. Orientation is achieved through rotation of samples and taking their average. Based on all these templates a finger detection system for the most hand positions and finger shapes was achieved.

In template matching method, one should also consider the number of classes of templates which needs to be created. One possible approach here was to make templates of each of the finger for different hand positions. That makes it 10 possible classes for 3 hand position cases and a total of 30 templates. Along with rotation, that creates a very large number of templates. At first, this approach was taken. During initial testing on the validation set, it was found that the ring, middle and index finger templates of one hand will work to detect the same finger of the other hand. E.g. – Ring finger template of right hand was useful in detecting ring finger of left hand and vice versa. A significant reduction in the number of templates was achieved if same templates were used for detection of these 3 fingers on both hands. Therefore it was decided to have only 7 classes of fingers –

1. Ring finger
2. Middle finger

3. Index Finger

4. Right Thumb

5. Left Thumb

6. Right little

7. Left little

For the first 3 classes, the template used for each of the finger class is same for both hands. The last 4 classes required individual templates for each hand. Also in all of them, rotation and resizing was required. The number of possible ways in which the last 4 classes of fingers appear was significantly larger than the first 3 classes. A detailed analysis on how templates for these 4 classes were collected appear later in this chapter.

To collect the samples, training images were required. On viewing the different datasets a particular sequence was selected which consisted of all possible finger positions. This sequence was from a dataset collected from one single user and contributed to all hand position types. From this sequence 20 training examples were selected. Each of the training example consisted fingers in different possible shapes and orientation. Training samples were collected from these 20 training images by extracting pieces of all the fingers .All training samples of a particular finger were aligned for one particular hand position and resized to 20*10. A template was then created for that particular hand position for that finger by taking their average. For example, if some of the samples for little finger were created in raised hand position, it was rotated in the required direction and by a required angle to make samples of flat position. All the samples of little finger were made to look like little fingers in flat position, resized and their average was taken. This rotation of training example ensured alignment in training samples for a particular hand positions from the 20 training samples for that particular finger. Snapshots below show how a sample for raised hand position of left little finger on the left was rotated to get sample flat hand position of little finger. All the samples of left little finger were rotated as required for a flat position, the extra pixels were removed on the sides to align them. It was then resized and averaged to create a template for left little finger at flat position.

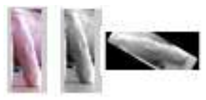

Figure 6-2 Rotation of training examples

Also, each of the samples collected was from the base of the finger to the tip of the finger. The first image above of the left little finger shows how the sample is taken from base to the tip of the finger. Every valid sample must consist of the finger from base of the finger (knuckles) to the fingertip. After rotating each sample, it was resized to 20*10 sizes, since not all training examples were of the same dimensions. Rotation of training examples was done to form an alignment between all training samples.

For template matching method, another parameter which had to be decided was the number of top matches for templates. The template matching method here tries to find top 5 matches for each of the template in the image. The number of 5 for the top matches was decided after tests on a validation set which consisted of 20 images. After initial training was done by creating templates, template matching method was run on these 20 validation images. At first only one top match for each template was selected. Since the images are of 640*480 resolution, templates which are of the size 20*10 did not find a particular finger from a template as a top match at all times. This method of finding the top match was accurate when hand position was applied but was not successful in detecting fingers when the entire image was considered. Since the user can run the detection method without using the hand detector, the number of matches must be increased. The number of matches was then increased to 5. Top 5 matches in most validation images had detected the particular finger intended for detection. Therefore detection of top 5 matches of the template in the image for Template Matching method is done. The top 5 matches were filtered again using skin, depth and hand information.

There are also depth frame associated with each of the images. After finding all the results using Template Matching, there is an option of using the depth frame. At first the average depth of the finger detection box was compared to the average depth of the face. If the average depth was lesser than of the face it was considered as a true positive. But on observing the depth frames, the sides along the fingers had zero depth. Please refer figure 6-3 below where the sides along a finger have zero depth.

In case pixels like this appear in the detection box the average depth was affected. Therefore to avoid these outliers, median of the detection boxes was considered rather than the average. All the detection boxes have a median depth associated . If the median happens to be zero then it is considered background since zero depth means the detection box is present too far away from the camera. This is explained in detail in the experiments section

In order to achieve more accuracy a hand detector was included. This hand detector (discussed later in this chapter 6.3) when done before running the finger detection Template Matching method, employs the position of hands and applies template detection only at the location of hands. The hand detector requires the depth frames to be loaded. After loading the depth frame, face detector should also should be run since the position of the hands are based on the median depth of the face. If the median depth of the finger detection box is not zero but greater than that of median depth of face detection box, then it is also considered background and is ignored.

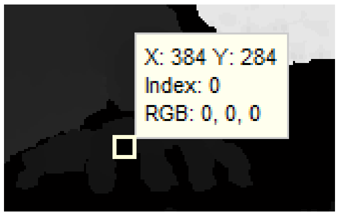

Figure 6-3 Sides of finger having depth zero.

6.2.2 Training and class wise template collection

To collect templates first training examples had to be selected. Using the sequence mentioned above, 20 training images were collected. In each of the image, fingers were in different positions. For example, in one image if the left little finger was placed on the table, in another it was raised. In yet another image, it was raised only mid way. Below are some images which explain the same example. All images are from the same user.

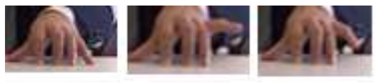

Figure 6-4 Different positions of the same finger.

Similar cases were considered for all fingers. Importance was given to dissimilar positioning of thumb and the little finger, since they had the most variation in positioning. It was also made sure that the middle, index and ring fingers were as distinct as possible. From the training images, samples of different fingers were collected by extracting them individually. This was achieved in MATLAB by - first reading the image and then extracting the position using the co-ordinates where the particular finger was in that image. An ideal sample should contain the sample subject right in the middle with no extra pixels read along the sides. But for some samples, especially thumb and little finger examples, there were extra pixels on the side. This happens for 2 reasons - as MATLAB considers everything as matrix if the sample subject lies along the diagonals some extra pixels were read and also because these samples were usually small. To avoid such cases the sample was rotated in the direction requires and then the sub window of the image without the extra pixels was considered as samples. The samples were then resized so that they fit the actual dimensions of template of 20*10 pixels. In experiments, it was noticed that sometimes because of the template having these extra pixels, finding a top match for a particular finger was very difficult. But this accuracy was improved when hand position was used.

6.2.2.1 Template collection for Ring, Middle and Index Finger.

Samples of each of these fingers based on the 3 cases of hand positions were made. From the training examples, these samples were extracted manually. Each sample collected contained the finger from base to the tip of the finger. For these 3 template classes, after samples of the fingers from the image were extracted, it was rotated to hand position desired and resized to 20*10. The average on each finger on both hands was taken separately. For example, average over the left index finger samples and right index finger samples were taken separately. The right hand and left hand average was then averaged to get the final template for that particular finger. Therefore for the earlier index finger example, the final index finger template was an average over the combination of the left index finger average and right index finger average. Similarly, the other templates were created for middle and ring fingers.

Rotation of training examples was not required for these 3 fingers since they cannot be bent sideways while the user taps these fingers on the table. The flat hand position these 3 fingers results in a finger tip; therefore samples of just normal and raised hand positions were considered. For index and ring fingers, rotations were considered for cases where the hands were positioned sideways (not facing front-on to the camera) or fingers appear slanted at an angle. An example of hand in such a case is shown below in the left, notice the ring finger is at an angle instead of facing front on like the example shown on the right side. To cover such cases rotation of ring and the index fingers were considered. Those rotations were achieved by rotation of final templates at an angle of 30 degrees both clockwise and anti-clockwise.

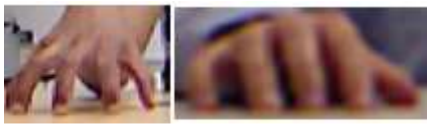

Figure 6-5 Need for rotated templates

Here are some examples of the samples and averages collected as templates for Template Matching for these 3 classes – Index Finger, Middle Finger and Ring Finger

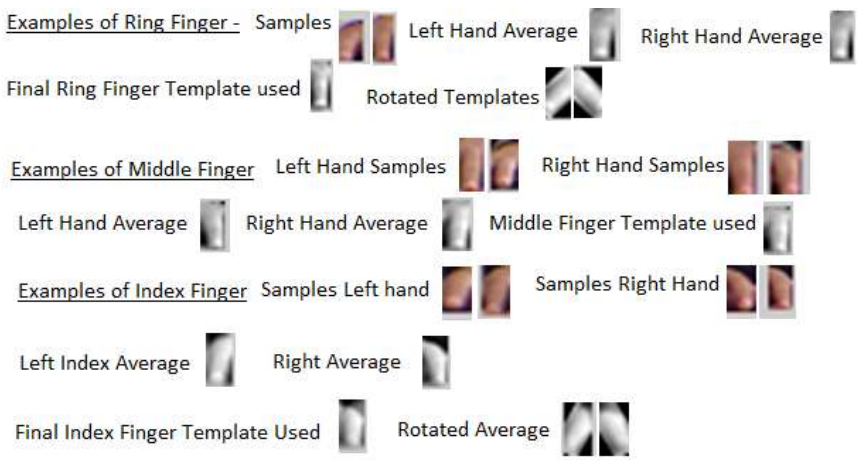

Figure 6-6 Samples and Templates of Ring, Middle and Index Finger

6.2.2.2 Rotation, Aligning and Resizing of training examples

As explained above, not all training examples looked similar and were of the same dimensions. For example, the little finger samples varied from sizes of 13*11 to 9*15, based on the class of the hand position it belonged to. In order to create unified samples, resizing had to be done. Also a decision had to be made on what size the resizing should be done. Most of the samples extracted for middle, ring or the index finger were of the size 20*10. That size was consistently able to find good matches for template detection and also formed good samples covering base to tip of a given finger. Therefore for simplicity and consistency, 20*10 was decided to be the size in which all templates will be made. Resizing was lesser of a problem for all the templates created for ring, middle and index fingers. Although, all of the samples in these 3 classes were not of the same size, the sizes of samples were somewhere close with either of the dimensions being off by a maximum of 5 pixels.

For the little finger and thumb the range and the type of possible occurrences varied a lot. For example, here are some cases of how a left little finger can appear (these are not samples). The first image is where the little finger is lifted; the second one is that of a normal hand position and the third one being another possibility for normal position. The last one is an example where it is raised.

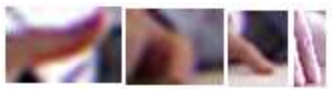

Figure 6-7 Little Finger Appearance

In order to bring such samples of different sizes to 20*10, *imresize* command of MATLAB was used. Here is an example of how a training example of left hand little finger was resized. The resulting resized sample was of the size 20*10. The first parameter of imresize is the grayscale image of the sample and the second parameter is that of the target resolution.

resized_raised_left_little_sample_1=imresize (raised_left_little_sample_1, [20 10]);

Before resizing some training examples required rotation. For example, in the case of left little finger explained above, the first sample wherein the finger is raised cannot be aligned with a sample in which the little finger is in raised hand position. The sample has to be rotated in order to align it with the other samples of raised little finger.

For this purpose, *imrotate* command of MATLAB is used. Here is how the example explained above can be aligned with the other samples.

raised_left_little_sample_1 = imrotate (gray_left_little_sample_1,-90,'bilinear','crop');

This function imrotate has 4 input parameters- the resized sample, the degree by which the sample should be rotated anti-clockwise, the method used for interpolation and size of the image box returned.

The work here uses resizing of the images first and then rotates the training examples. Resizing is done first since rotation of very small samples truncates some of the edges. In order to retain most of the sample image, resizing to 20*10 was done first. To retain the size after the rotation, the last parameter here is used as ‘crop’ and the output image after rotation is of the same size 20*10. The method used for rotation here is Bilinear Interpolation. Bilinear Interpolation uses a weighted average of the four nearest pixels and the range of output image is always within the same range of values as the input image. In the example explained above, the lifted little finger sample has to be rotated by 90 degrees clockwise to be aligned with the raised left little finger training sample. So a value of -90 degrees under the degree parameter of imrotate was given.

Likewise all training samples of thumb and little fingers were rotated to form sample of raised, normal and flat positions. In case of thumb and little fingers, samples of the flat position were a case when the finger was lifted (as shown in the first image in the left little example samples shown earlier in the same section).Every sample required a different angle to align with the required hand position. Each sample was rotated in the required direction (clockwise or anticlockwise) and in the required angle (in degrees) to align the training samples. By rotating and making all the samples of the same size, alignment of all the training samples was achieved. Extra pixels were removed on the sides so that the sample was as pure as possible. It was then resized and average over all the resized sample of one type formed the final template for each hand position in each class. Therefore after rotating, aligning and resizing these training examples samples for raised, normal and flat positions were obtained for all the 7 classes of fingers.

6.2.2.3 Template collection for Thumb and Little Finger.

The cases of detection and template creation of thumb and little fingers are different to that of ring, middle and index fingers. In this section a study on why the left and the right hand was made into separate classes, how the templates are achieved for this purpose and how accuracy can be improved there.

An earlier section (section 6.2.2.1) discussed how templates are created for ring, middle and index fingers. For these 2 fingers – thumb and little finger, the amount of variation of possible finger shapes are large. These 2 fingers on each hand were in different direction, so templates for these were separated out into 4 different classes. A decision was taken to make it into separate classes for 2 hands. Except for the 3 hand positions, the earlier 3 classes didn't have any sideways shape. While thumb and little fingers posses sideway shapes. The pictures below show the sideway shapes of thumb and little fingers.

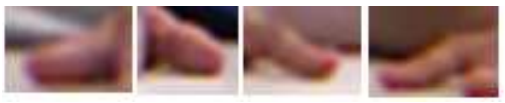

Figure 6-8 Sideway shapes of Little Finger and Thumb

Thumb is placed sideways, flat on its side during both the normal and the flat hand position. For raised position the thumb is at an angle with the fingertip on the table. First, samples of thumb from both the hands separately from all the 20 training images were collected and resized to 20*10. The samples where in the thumb was in raised position was rotated as described in the previous section and made to look like a sample of other 2 classes. After rotation, sub windows with just the fingers were extracted and resized. Average of these resized samples was collected to form flat and normal template. The sample of normal and flat hand positions were then rotated to form samples of raised hand position. Average of these samples were now formed the template for raised hand position. This process was done for thumbs of both the hands to create templates for right and the left thumb. So it consists of 2 templates each – one for flat and raised hand positions together and another one for raised hand position. Therefore there are 4 different templates for thumb.

For the little finger though the case is different. For the flat hand position, the little fingers are placed sideways (3rd and the 4th picture above) and for the other 2 hand positions the samples look different. But during testing on validation set, it was found that for the raised position templates from middle, index or ring finger can detect the raised little finger. So template for raised hand position for little finger was not created. Therefore a total of 4 templates were created for little fingers like the case of thumb with 2 each for both the hands. The process of rotation, resizing and aligning was similar to that explained for thumb.

This system of template matching improved the accuracy in detection of the thumb and little fingers. Further using the hand detector increases performance of the system even more. Using the hand location reduces the search space for template to a very small section of the image and therefore is more accurate. Further increase in performance can be achieved by creating more number of templates in which all the variations in shape and orientation is captured.

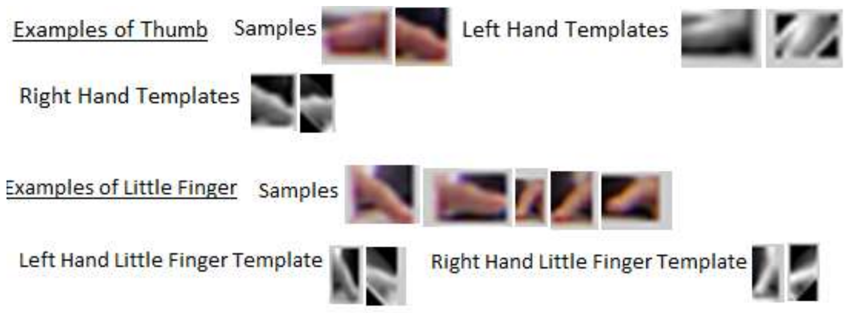
Examples of Thumb
Samples
Left Hand Templates
Right Hand Templates
Examples of Little Finger
Samples
Left Hand Little Finger Template
Right Hand Little Finger Template

Figure 6-9 Samples and Templates for Little and Thumb

6.2.2 PCA for Finger Detection

Principal Component Analysis (PCA) is an unsupervised learning algorithm. Dimensionality reduction is an unsupervised learning problem which allows data compression and increases speed. Inputs to a learning algorithm are called features. In some problems, the number of features is large and sometimes redundant. Reduction in dimensions reduces the total data involved in the process and thereby increases the speed. Consider a problem of n features where n is a large number. To simplify this problem a reduction in the number of features is required. PCA provides a method to project n-features onto k-features (where k is a very small number compared to n) with least projection error. For a set of data points x_1, x_2, x_3 ……. x_n

$$Error(P) = \sum_{i=1, j=1}^{n,n} Error(X_i, X_j) \text{ where i != j}$$

In PCA the aim is to preserve the distance between data points unlike linear regression where the motive is to minimize the squared error from data point to projection. If the distances between data points are preserved the error is zero.

PCA requires preprocessing of training data. It involves mean normalization, which involves computing the mean of each feature and then replace each training example for a feature with the difference with the feature mean.

For a training set : x^1,x^2,…,x^m of m examples and n features.

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^i$$ replace each feature x_j^i with $x_j^i - \mu_j$

If features are on different scales, scale each features to have comparable range of values. This is called as Feature Scaling. This should be done only when features take on different range of values.

$$x_j^i = \frac{(x_j^i - \mu_j)}{s_j}$$ where s_j is the standard deviation for the feature

To reduce from n-features to k- features, we first compute the Covariance Matrix denoted by Sigma (∑)

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$$ where $x^{(i)}$ are n –feature vectors(n * 1 matrix)

∑ is therefore a n*n matrix

We compute Eigenvectors (U) and Eigenvalues (S) using Singular Value Decomposition (SVD)

[U, S, V]=svd(∑) or [U, S, V]=eig(∑)

Eigenvectors (U) is an n*n matrix. We use the first k columns of the Eigenvectors matrix to reduce from n-features to k-features. Therefore an n*k matrix is considered (U_r). Eigenvalues is an n*n matrix with values only at the diagonal. Eigenvalues at a column depicts the value for that particular Eigenvectors. Higher value in a column of Eigenvalues indicates that particular Eigenvector (feature) has higher importance. Projection using the reduced Eigenvectors matrix U_r is given by

$$z = U_r^T x$$ x is feature vector

The dimensions of U_r is n*k and U_r^T is k*n. The dimensions of x feature vector is n*1. The projection z, is therefore of the size k*1, a representation of n-features using k-features.

The projection is then back projected to convert it back into the dimensions we started with we use

$$x = U_r z$$ (n*k and k*1 yielding n*1)

6.2.2.1 Training for Finger Detection

The samples of fingers collected for template matching are used as training examples. The 7 class approach described for Template Matching is the same for PCA. Mean normalization was carried out by taking average of the training examples of one type of class and deducting the average of those training examples and dividing it by the standard deviation. Eigenvectors and Eigenvalues were computed using the eig function. The number of eigenvectors to use is fixed as 10. The results of template matching were used as a starting point for PCA method. All detections from Template Matching were projected using the top 10 eigenvectors. The projection is then back projected using the projection and the top 10 eigenvectors, to convert it back into the dimensions we started with of n*1. The projection error is calculated as the difference between the detection we started with and the back projection.

6.3 Depth Based Hand Detector

Each RGB frame has an associated depth frame. These frames are aligned; therefore the value at every pixel in the depth frame is the distance from the camera. This system works on the assumption that hands are placed in front of the face and in between camera and the face. Hands combined with the arm are the only region where skin is present between face and the camera. We have used a face detector to locate the position of the face and its center. Depth frame gives us the depth at the center of the face. We use this as the threshold for the depth frame. All the pixels in the depth frame with the value of depth greater than the face are removed. We also use histogram based skin detection and fixed threshold of 0.5 the result to find areas with skin. The 2 threshold results are multiplied (element-by-element) to get the areas with skin having lesser depth. The 2 largest connected components from the resultant image are identified as hand locations.

The user can load sequences of depth frames and on selection of a particular sequence tries to match the corresponding depth sequence. If an incorrect sequence is loaded, the system provides an error saying “*Depth Sequence loaded is incorrect*” in a message box. On loading the correct sequence, the system automatically picks up the frame from that sequence. Note that, the loading of the frame depends on the number of frames in the sequence. So loading another sequence with the same number of frames as the current sequence may cause loading the incorrect frame.

The most use of the depth frames comes in increasing the accuracy of the system. The baseline system of detection algorithm using Template Matching or PCA along with skin detection does prove to be effective. But sometimes the system might find some matches which might be in the background. The depth frames are also very useful in detection of hands. The system includes a face detector. The assumption made here for detection of hands is that – Both the hands are placed before the face. It means that the depth of the face is more when compare to that of the hands or in turn the hands are much closer to the Kinect camera than the face.

There is an additional button added on top of the main window Detection(Skin and Depth). This uses Depth and Skin to find the presence of fingers in the given image. The input for this method is the result of one of the algorithms for detection combined with Skin Detection. Another thing to note is that, this button can only be triggered when depth frames of the sequence are loaded.

Chapter 7

Experiment and Results

7.1 Experiments with Skin

In experiments with skin we run the detection algorithm and apply skin detection with threshold to the results. The 2 parameters on which the results depend are the Template Matching threshold and Area Percentage. Here as the first experiment we vary the skin detection threshold from 0 to 1, set area percentage to 0.5 and set the template matching threshold to 0.5. The cumulative results for True Positives and False Positives for Template Matching and PCA is given by

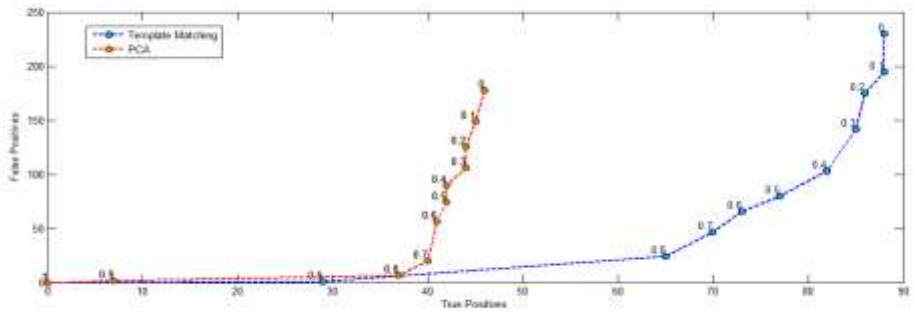

Figure 7-1 Experiments with Skin – Varying Skin Threshold

In a second experiment with skin, we set the Template Matching threshold to 0.5 and Skin detection Threshold to 0.5. We vary the area percentage from 0% to 100%.

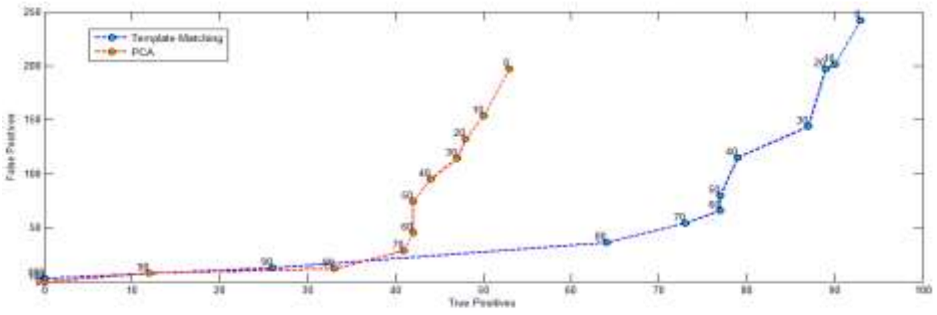

Figure 7-2 Experiments with Skin – Varying Area Percentage

7.2 Experiments with Hand Detector

In experiments with hand detector, we run the detection algorithm only on the hand location sub windows and apply skin detection with threshold to the results. Here similar to the first experiment with skin we vary the skin detection threshold from 0 to 1, set area percentage to 0.5 and set the template matching threshold to 0.5. The cumulative results for True Positives and False Positives for Template Matching and PCA is given by

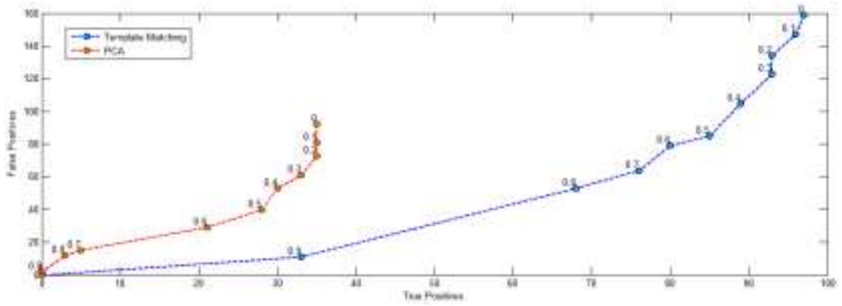

Figure 7-3 Experiments with Hand Detector – Varying Skin Threshold

In a second experiment with hand detector, we set the Template Matching threshold to 0.5 and skin detection threshold to 0.5. By varying the area percentage from 0% to 100%, we get cumulative results for Template Matching and PCA displayed by this plot.

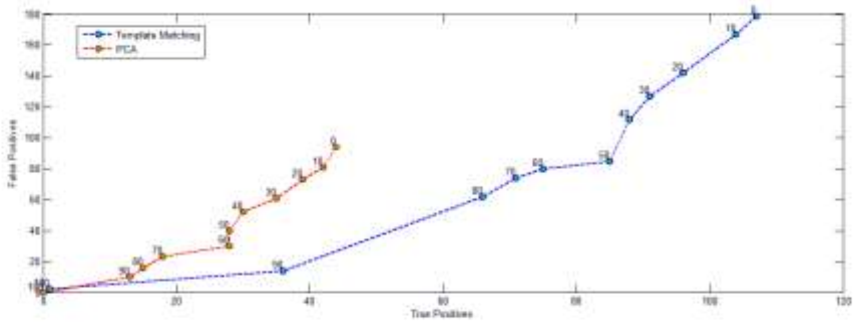

Figure 7-4 Experiments with Hand Detector – Varying Area Percentage

7.3 Experiments with Depth

In experiments with depth, we run the detection algorithm and apply skin detection with threshold to the results similar to experiments in 7.1. To filter the results further more we use the depth information. The median value of each of the detection boxes are calculated and checked if the median value is less than the depth of the face. The results show lesser false positives.

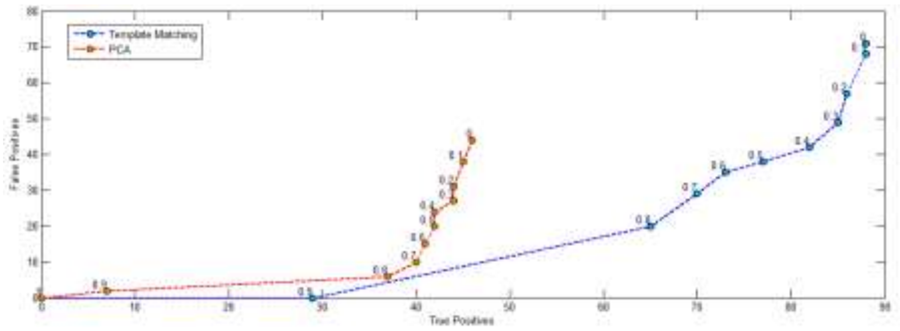

Figure 7-5 Experiments with Depth – Varying Skin Threshold

In a second experiment with hand detector, we set the Template Matching threshold to 0.5 and skin detection threshold to 0.5. By varying the area percentage from 0% to 100%, we get cumulative results for Template Matching and PCA displayed by this plot.

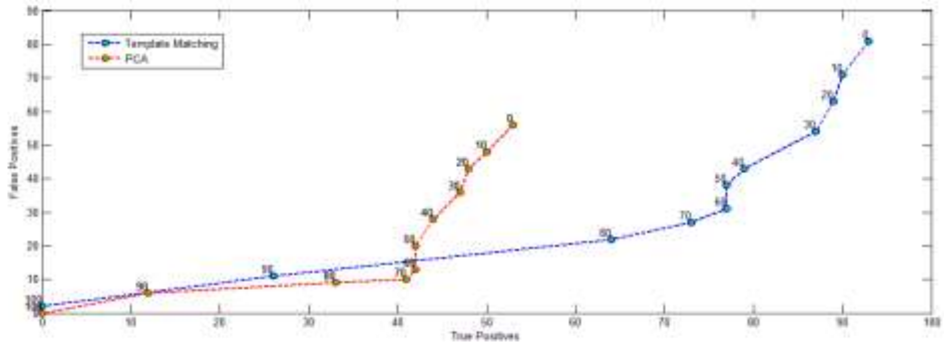

Figure 7-6 Experiments with Depth – Varying Area Percentage

Table 7-1 Template Matching Cumulative Results of Experiments

TP-True Positives	TEMPLATE MATCHING					
FP- False Positives	Skin Detection		Hand Detection		Depth	
Skin Threshold	TP	FP	TP	FP	TP	FP
0	88	230	97	159	88	71
0.1	88	195	96	147	88	68
0.2	86	175	93	134	86	57
0.3	85	142	93	123	85	49
0.4	82	103	89	105	82	42
0.5	77	80	85	85	77	38
0.6	73	66	80	79	73	35
0.7	70	47	76	64	70	29
0.8	65	24	68	53	65	20
0.9	29	1	33	11	29	0
1	0	0	0	0	0	0
Area Percentage	TP	FP	TP	FP	TP	FP
0	93	242	107	179	93	81
10	90	201	104	167	90	71
20	89	197	96	142	89	63
30	87	144	91	127	87	54
40	79	115	88	112	79	43
50	77	80	85	85	77	38
60	77	66	75	80	77	31
70	73	54	71	74	73	27
80	64	36	66	62	64	22
90	26	13	36	14	26	11
100	0	3	1	2	0	2

Table 7-2 PCA Cumulative Results of Experiments

TP-True Positives FP-False Positives	PCA					
	Skin Detection		Hand Detection		Depth	
Skin Threshold	TP	FP	TP	FP	TP	FP
0	46	178	35	92	46	44
0.1	45	149	35	81	45	38
0.2	44	126	35	73	44	31
0.3	44	106	33	61	44	27
0.4	42	90	30	53	42	24
0.5	42	74	28	40	42	20
0.6	41	57	21	29	41	15
0.7	40	20	5	15	40	10
0.8	37	7	3	12	37	6
0.9	7	2	0	2	7	2
1	0	0	0	0	0	0
Area Percentage	TP	FP	TP	FP	TP	FP
0	53	197	44	94	53	56
10	50	154	42	81	50	48
20	48	132	39	73	48	43
30	47	114	35	61	47	36
40	44	95	30	52	44	28
50	42	74	28	40	42	20
60	42	45	28	30	42	13
70	41	29	18	23	41	10
80	33	12	15	16	33	9
90	12	8	13	10	12	6
100	0	0	0	0	0	0

References

[1] A Antonelli, Forli Biometrika s.r.l, R Cappelli , D Maio and DMaltoni , “Fake Finger detection by Skin Distortion," *IEEE Transactions in Information Forensics and Security ,* 2006. Page 360 - 373

[2] Weixin Yang, Zhengyang Zhong, Xin Zhang, Lianwen Jin, Chenlin Xiong, Pengwei Wang, “Depth Camera Based Real-Time Fingertip Detection using Multi-View Projections*”. 15th International Conference, HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part V*.Pages 254-261

[3] Ravikiran J, Kavi Mahesh, Suhas Mahishi, Dheeraj R, Sudheender S, Nitin V Pujari., “Finger Detection for Sign Language Recognition," *in Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol I IMECS 2009, March 18 - 20, 2009, Hong Kong*.

[4] Sung Kwan Kang ; Dept. of Comput. Sci. & Eng., Inha Univ., Incheon ; Mi Young Nam ; Phill Kyu Rhee, “Color Based Hand and Finger Detection Technology for User Interaction" *in International Conference on Convergence and Hybrid Information Technology, 2008. ICHIT '08 Pages 229 - 236*

[5] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak and Denis Fisseler, “Analysis of the Accuracy and Robustness of the Leap Motion Controller”. *Sensors 13*, Accepted 6 May 2013 no. 5: 6380-6393

[6] Jia Jia, Lianhong Cai, Kaifu Zhang, Dawei Chen, “A New Approach to Fake Finger Detection Based on Skin Elasticity Analysis,” *International Conference, ICB 2007, Seoul, Korea, August 27-29, 2007. Proceedings*

[7] Zhou Ren, Junsong Yuan and Zhengyou Zhang , “Robust hand gesture recognition based on finger-earth mover’s distance with a commodity depth camera” *MM '11 Proceedings of the 19th ACM international conference on Multimedia Pages 1093-1096*.

[8] J. Mackie and B. McCane. “Finger Detection with Decision Trees,” *University of Otago, Dept. of Computer Science*

[9] A. Corradini, “Finger Detection based on hand contour and color information” *in 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI) [97 - 100]*

[10] Li-wei Chan, Yi-fan Chuang, Yi-wei Chia, Yi-ping Hung, Jane Hsu, “A new method for multi-finger Detection using a regular diffuser,” *in 12th International Conference, HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part III*

[11] Peng Song, Hang Yu and Stefan Winkler, “Vision-based 3D finger interaction for mixed reality games with physical simulation” *VRCAI '08 Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. Article No. 7

[12] Oka, K.; Sato, Y.; Koike, H., “Real time fingertip tracking and gesture recognition,” Computer Graphics and Applications, IEEE.2002 [64 - 71]

[13] Ravikiran J, Kavi Mahesh, Suhas Mahishi, Dheeraj R, Sudheender S, Nitin V Pujari, “Finger Detection for Sign Language Recognition,” *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol I IMECS 2009, March 18 - 20, 2009, Hong Kong*.

[14] Hui-Shyong Yeo, Byung-Gook Lee, Hyotaek Lim, “Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware,” *in Multimedia Tools and Applications Journal(2013)*

[15] Wei-Yun Yau, Hoang-Thanh Tran, Eam-Khwang Teoh, Jian-Gang Wang, “Fake Finger Detection by Finger Color Change Analysis,” in International Conference, ICB 2007, Seoul, Korea, August 27-29, 2007. Proceedings. pp 888-896

[16] Michael J. Jones and James M. Rehg, “Statistical Color Models with Application to Skin Detection,” *in Cambridge Research Laboratory*. International Journal of Computer Vision.January 2002, Volume 46, Issue 1, pp 81-96

[17] J. P. Lewis, “Normalized Cross-Correlation,” *Industrial Light & Magic,* Pages[120-123]. This is an expanded version of apaper from Vision Interface, 1995(reference [10])

[18] M. Jones and J. Rehg,n, “Statistical color models with application to skin detection,” *Intenational Journal of Computer Vision*, vol. 46, no. 1, pp. 81–96, January 2002.

[19] Paul Doliotis, Vassilis Athitsos, Dimitrios Kosmopoulos, and Stavros Perantonis. “Hand Shape and 3D Pose Estimation using Depth Data from a Single Cluttered Frame”. International Symposium on Visual Computing (ISVC), July 2012.

[20] Kai Briechle ; Uwe D. Hanebeck “Template matching using fast normalized cross correlation” *Optical Pattern Recognition XII, 95* (March 20, 2001)

Biographical Information

Sanjay Vasudeva Iyer was born in Bangalore, India in 1986. He completed his B.E in Computer Science and Engineering from Visvesvaraya Technological University, India in 2008. Following his bachelor’s degree, Sanjay was working a Software Engineer and late as Senior Software Engineer at Infosys Technologies Limited, Mangalore over a span of 3 years. In 2011, he joined University of Texas at Arlington for his masters in Computer Science. His current research interests are in the area of Computer Vision, Data mining, Data Analysis & Modeling and Machine Learning.