CHAPTER -5

ARCHITECTURE

This system is perform adequate actions. A camera is the major hardware component of system. Camera is used to captures the hand movements. These images are sending to Computer for image processing. Image is processed in Computer i.e. hand is detected and gesture is recognized from the image. With respect to hand gesture information is retrieved from the internet. These information displays on the Computer. The proposed system consists of both face and hand detection and recognition, daily information retrieval from Internet, shown on Figure 5.1 below.

5.1 Proposed Architecture:

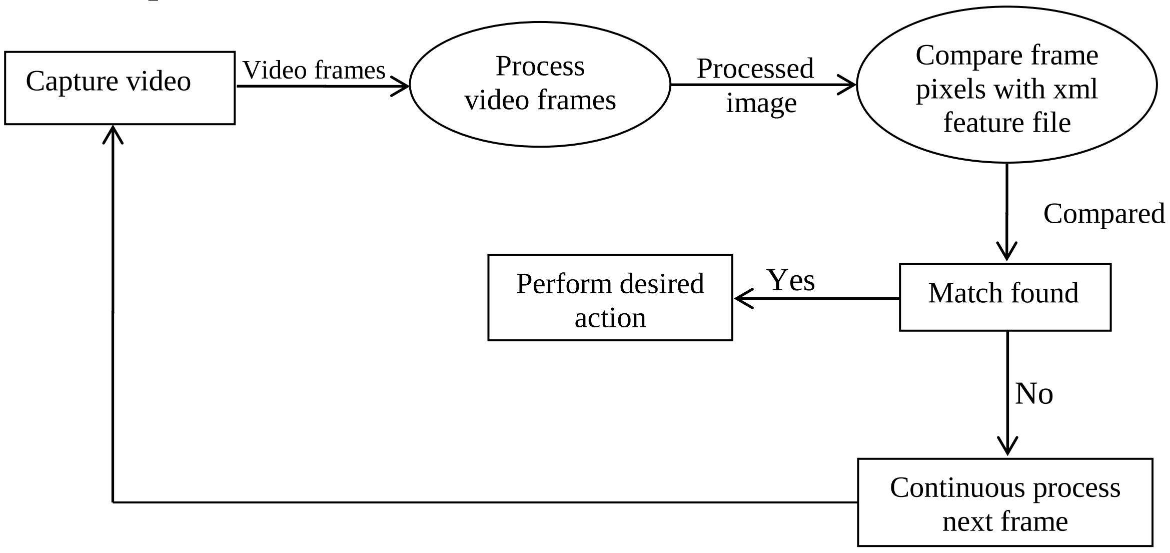

Figure 5.1: Proposed System Architecture

Video capturing is the first step in any vision system, only after this process you can go forward with the image processing. In this application it is done by using IPWebCam of laptop. The application uses the camera present in the laptop for continuous image capturing and a simultaneous display on the screen.

The second step is the processing of video frames. Here each frames from the video is processed separately. There are many ways as shown in Figure 5.2 below.

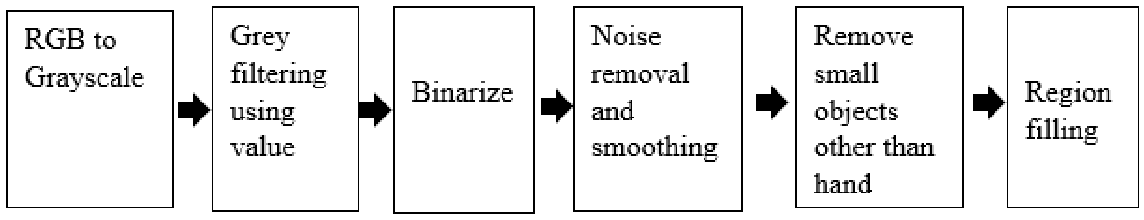
RGB to Grayscale
Grey filtering using value
Binarize
Noise removal and smoothing
Remove small objects other than hand
Region filling

Figure 5.2: Image Processing Steps

(1) Skin Segmentation: The first step in implementing our particular gesture recognition system is being able to effectively segment skin pixels from non-skin pixels. By using only a simple RGB-based webcam we are limited in methods for locating and distinguishing static hand gestures. For this reason, we have chosen to focus on segmentation using various color spaces. Skin segmentation methods are generally computationally inexpensive, and moreover, they can function robustly across many different models of simple webcams - an important feature given the notable difference in quality, color, etc. that can exist between webcams. Specifically, we use a basic thresholding technique, choosing min and max threshold values which contain well-known and thoroughly tested skin pixel value regions. In total we use three different color spaces: 1) RGB (red-blue-green), 2) HSV (Hue-Saturation-Value), and 3) YCbCr (luminance, blue-difference and red-difference chroma). Using three color spaces is a computationally efficient procedure, especially given the added robustness it provides to distinguishing skin and non-skin values. HSV and YCbCr color spaces provide additional information about the separation between luminance and chrominance that RGB color space doesn't provide. In all we use nine different pairings of min- and max- threshold values, each representing a given color space, then, applied certain boundaries rules introduced by Thakur et al. A particular pixel is part of skin region if and only if that pixel value passes boundaries rules.

(2) Canny Edge detection: Edge detection is one of the fundamental operations when we perform image processing. It helps us reduce the amount of data (pixels) to process and maintains the structural aspect of the image. We're going to look into two commonly used edge detection schemes - the gradient (Sobel - first order derivatives) based edge detector and the Laplacian (2nd order derivative, so it is extremely sensitive to noise) based edge detector. Both of them work with convolutions and achieve the same end goal - Edge Detection.

Canny Edge detection was invented by John Canny in 1983 at MIT. It treats edge detection as a signal processing problem. The key idea is that if you observe the change in intensity on each pixel in an image, it’s very high on the edges. In this simple image below, the intensity change only happens on the boundaries. So, you can very easily identify edges just by observing the change in intensity. The effect of Canny edge detection on input image is shown in Figure 5.3. The Canny Edge Detector identifies edges in 4 steps:

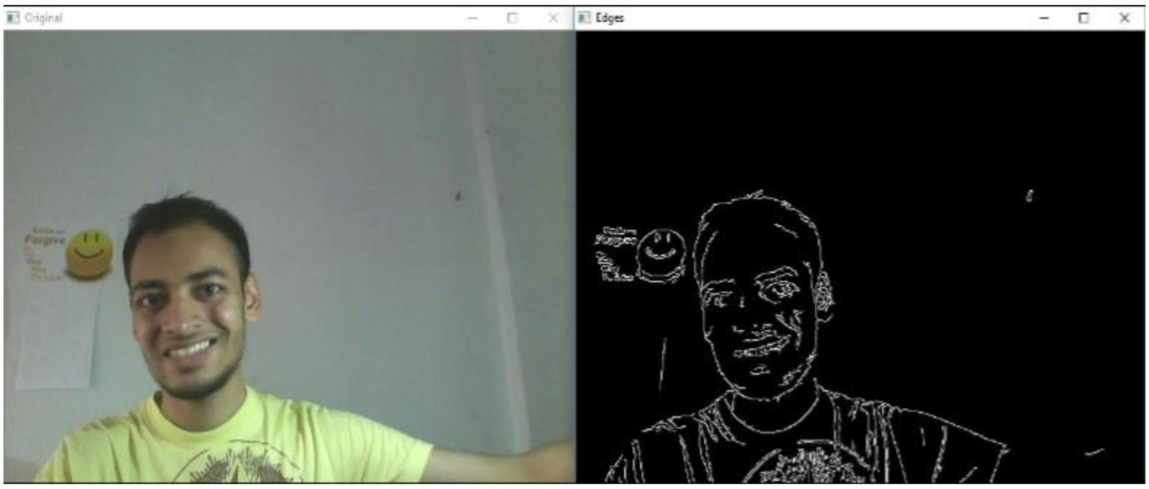

Figure 5.3: Thresholding Effect On Input Image

2.1. Noise removal: Since this method depends on sudden changes in intensity and if the image has a lot of random noise, then it would detect that as an edge. So, it’s a very good idea to smoothen your image using a Gaussian filter of 5×5.

2.2.Gradient Calculation: In the next step, we calculate the gradient of intensity(rate of change in intensity) on each pixel in the image. We also calculate the direction of the gradient. Gradient direction is perpendicular to the edges. It’s mapped to one of the four directions(horizontal, vertical, and two diagonal directions).

2.3. Non-Maximal Suppression: Now, we want to remove the pixels(set their values to 0) which are not edges. You would say that we can simply pick the pixels with the highest gradient values and those are our edges. However, in real-world images, gradient doesn’t simply peak at one pixel, rather it’s very high on the pixels near the edge as well. So, we pick the local maxima in a neighborhood of 3×3 in the direction of gradients.

2.4. Hysteresis Thresholding: In the next step, we need to decide on a threshold value of the gradient below which all the pixels would be suppressed(set to zero). However, Canny edge detector using Hysteresis thresholding. Hysteresis thresholding is one of the very simple yet powerful ideas. It says that in place of using just one threshold we would use two thresholds:

High threshold= A very high value is chosen in such a way that any pixel having gradient value higher than this value is definitely an edge.

Low threshold= A low value is chosen in such a way that any pixel having gradient value below this value is definitely not an edge.

(3) Background substraction: Background subtraction is a major preprocessing steps in many vision based applications. For example, consider the cases like visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

If you have an image of background alone, like image of the room without visitors, image of the road without vehicles etc, it is an easy job. Just subtract the new image from the background. You get the foreground objects alone. But in most of the cases, you may not have such an image, so we need to extract the background from whatever images we have. It become more complicated when there is shadow of the vehicles. Since shadow is also moving, simple subtraction will mark that also as foreground. The effect of background subtraction on input image is shown in Figure 5.4.

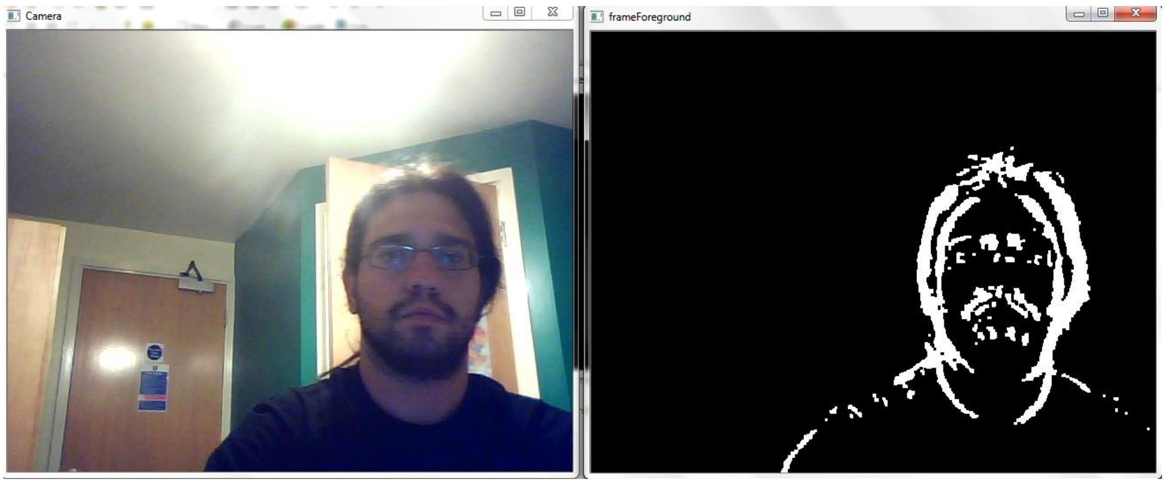

Figure 5.4: Effect Of Background Substraction

(4) Thinning: Thinning is the transformation of a digital image into a simplified, but topologically equivalent image. It is a type of topological skeleton, but computed using

mathematical morphology operators. Thinning is a morphological operation that is used to remove selected foreground pixels from binary images, somewhat like erosion or opening. It can be used for several applications, but is particularly useful for skeletonization. In this mode it is commonly used to tidy up the output of edge detectors by reducing all lines to single pixel thickness. Thinning is normally only applied to binary images, and produces another binary image as output.

The thinning operation is related to the hit-and-miss transform, and so it is helpful to have an understanding of that operator before reading on. Thinning operation is calculated by translating the origin of the structuring element to each possible pixel position in the image, and at each such position comparing it with the underlying image pixels. The effect of thinning on input image is shown in figure 5.5.

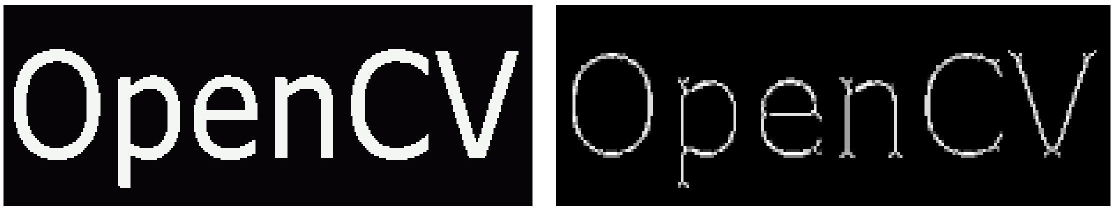

Figure 5.5: Thinning Effect On Input Image

(5) Image thresholding: Thresholding is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images. The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity is less than some fixed constant or a white pixel if the image intensity is greater than that constant. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is cv2.threshold. It is then used to detect objects in other images. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.

OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

• cv2.THRESH_BINARY

• cv2.THRESH_BINARY_INV

• cv2.THRESH_TRUNC

• cv2.THRESH_TOZERO

• cv2.THRESH_TOZERO_INV

In next step large dataset of both positive and negative images is captured. We have generated 600 positive images and 800 negative images and are stored in folders. The steps of creating XML file are:

1. Install OpenCV & get OpenCV source

```
brew tap homebrew/science
brew install --with-tbb opencv
wgethttp://downloads.sourceforge.net/project/opencvlibrary/opencv-
/unix/2.4.9/opencv-4.0.1.zip
unzip opencv-2.4.9.zip
```

2. Put your positive images in the ./positive_images folder and create a list of them:

```
find ./positive_images -iname "*.jpg" > positives.txt
```

3. Put the negative images in the ./negative_images folder and create a list of them:

```
find ./negative_images -iname "*.jpg" > negatives.txt
```

4. Create positive samples with the bin/createsamples.pl script and save them to the ./samples folder:

```
perl bin/createsamples.pl positives.txt negatives.txt samples 1500\
"opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\
 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"
```

5. Use tools/mergevec.py to merge the samples in ./samples into one file:

```
python ./tools/mergevec.py -v samples/ -o samples.vec
```

6. Start training the classifier with opencv_traincascade, which comes with OpenCV, and save the results to ./classifier:

```
opencv_traincascade -data classifier -vec samples.vec -bg negatives.txt\
-numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 1000\
-numNeg 600 -w 80 -h 40 -mode ALL -precalcValBufSize 1024\
-precalcIdxBufSize 1024
```

7. Wait until the process is finished. It may take long time, depending on the computer you have and how big your images are.

8.Use your finished classifier!

```
cd ~/opencv-2.4.9/samples/c
chmod +x build_all.sh
```

```
./build_all.sh
./facedetect --cascade="~/finished_classifier.xml"
```

Later, in next step we use HAAR classifier which matches each frames features with the XML file. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle as shown below in Figure 5.6.

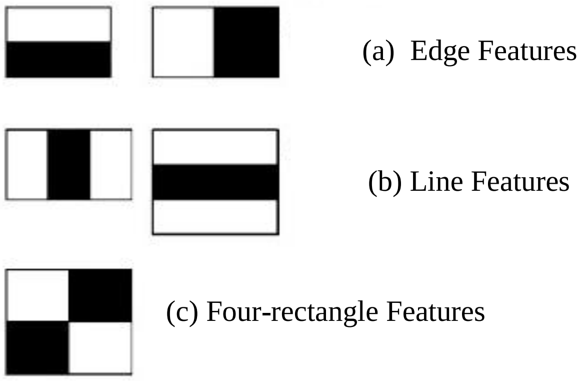

Figure 5.6: Haar Cascade Classifier

Initially, the algorithm needs a lot of positive images and negative images to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. For this they introduced the concept of Cascade of Classifiers. Instead of applying all the features on a window, group the features into different stages of classifiers and apply one-by-one. If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. OpenCV comes with a trainer as well as detector. If we want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one.