HAND GESTURE RECOGNITION

REVIEW-2

Group ID-6

COURSE CODE : CSE4019

COURSE TITLE : IMAGE PROCESSING

SLOT : G2

Under the Guidance of

SANTHI V

GROUP MEMBERS

Eashan Kaushik 17BCE2031

Tanay Nileshkumar Nagarsheth 17BCE2230

Samartg Chauhan 12BCE0283

Prakhar 17BCE2251

I. ABSTRACT

An increasingly impactful part of everyday life is how we interact with our technological devices - most notably our computers. Much of the ingenuity stemming from human-computer interaction research focuses only on improving current mainstream devices out there today. Only a few modes of Human-Computer interaction exist today: namely through keyboards, mouses, touch screens, and other handheld helper devices. Each of these devices has been confronted with their own limitations when adapting to more powerful and versatile hardware in computers.

A commonality between the issues of these devices are this: providing an intuitive mode of human-computer interaction cheaply without the additive of extra devices. With that said, we felt there were ways we could build an intuitive gesture-control mode of human-control interaction without the need to necessarily build another device. When tackling this problem of creating a new mode of human-computer interaction we knew we could utilize a combination of built-in functions that is typically provided in most computer designs. Specifically we have exploited the built-in webcam that has become a standard feature in most computers. This feature provides us a way to track and respond user hand-free movements and gestures.

Using a combination object detection and recognition, the following project successfully builds a computationally inexpensive static hand gesture recognition system using a simple RGB webcam - creating a truly more natural form of human-computer interaction.

II. INTRODUCTION

Generally, Hand Gesture Recognition technology is implemented using “Data Gloves” or “Color pins” which in turn leads to additional cost and lack of availability among majority of masses. Also, using additional devices, involves more amount of maintenance. Webcam is an easily available device and today every laptop has an integrated webcam along with it. In our project, we will implement a hand gesture recognizer which is capable of detecting a moving hand with its gesture in webcam frames. In future, it may be considered that willing to be more natural and more comforted, human being, who has been communicated with computers through mouse, keyboards, several user interfaces and some virtual environments, may use their bare hands to interact with machines without any mediator. As the set of materials above, recognition of hand gestures and postures is a satisfactory way to first steps of solutions instead of using keyboards, mouse or joysticks. A very common disease known as Parkinso’s disease is very relevant now-a-days among the common masses. This disease is caused due to excessive use of keyboard and mouse. Use of Gesture recognition technology will prevent this in future.

The Various steps involved in hand gesture recognition are:

1. 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. [2]. A particular pixel is part of skin region if and only if that pixel value passes boundaries rules.

2. Hand Detection

The next step in implementing this static gesture recognition system is locating the hand in the video frame. At this stage we are provided a segmented skin mask of the original image. While the segmented skin mask provides us the regions in which the hand could be, the hand is only part of all that is segmented from the mask. Since the mask captures skin regions - most notably the arm and face skin - we must do more processing to find the hand. Specifically, we want to find an indistinguishable characteristic to the hand, namely a consistent feature we can always find. With that mindset, we felt the palm of the hand provided that consistency. Much of the difficulty in finding the location of the palm is the variability in size. We didn't have any information regarding the position of the user's hand as well as size (i.e. the closer the user's hand to the screen the larger hand appears). In addition, we were more interested in fingers which played a crucial role in static hand gesture recognition algorithm (discussed in the next section).

To solve this problem, we took advantage of the hand geometry. Any human hand's largest cross section area lies somewhere near the middle of the palm and independent of size. With this property, we use the centroid equation to find the arithmetic mean position of the hand which corresponds to the center of the palm. We can easily differentiate between the top and bottom of the hand after finding center of the hand. By using this method, we provided a robust algorithm to detect the center of any human hand.

3. Static Hand Gesture Recognition

After the location of the hand is found, the challenge becomes identifying different gestures of the given hand. The main difficulty in this is what type of features to use in distinguishing various static gestures. We concluded that most static gestures relied on the hand's fingers. Though a simple conclusion, it is this very deduction that our algorithm was built around. While more complex algorithms for contour detection and gesture recognition exist, we chose a simpler algorithm based on and adapted from Malima [3].

In our adapted model, we draw a rectangle around the center of the palm found in the preceding procedure. This rectangle's size is proportional to the furthest skin pixel value from the center of the palm. Ideally the rectangle drawn does not entirely encapsulate the hand. Instead, the lines of the rectangle only contain the palm, cutting through the fingers and wrist. In this process we make a key assumption, in that the hand is upright (i.e. the line of the rectangle that cuts through the fingers is known.) Then we simply calculate the number of fingers that cross the line of the rectangle (calculations explained in greater detail in the section below.) This algorithm provides us a very simple, computationally efficient method for counting the number of fingers any given hand has up.

III. LITERATURE SURVEY

Hasan applied multivariate Gaussian distribution to recognize hand gestures using nongeometric features. The input hand image is segmented using two different methods ; skin color based segmentation by applying HSV color model and clustering based thresholding techniques . Some operations are performed to capture the shape of the hand to extract hand feature; the modified Direction Analysis Algorithm are adopted to find a relationship between statistical parameters (variance and covariance) from the data, and used to compute object (hand) slope and trend by finding the direction of the hand gesture , As shown in Figure 5.

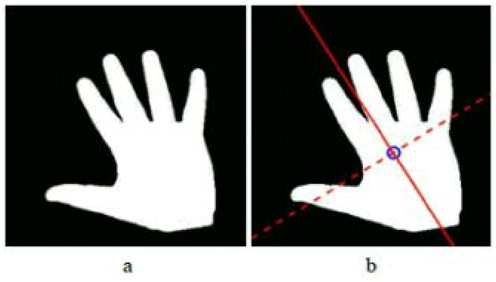


Figure5. computing hand direction .

Then Gaussian distinction is applied on the segmented image, and it takes the direction of the hand as shown in figure 6.

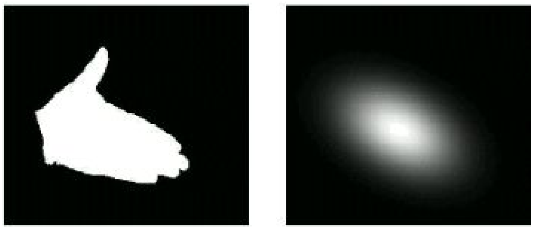


Figure 6. Gaussian distribution applied on the segmented image .

Form the resultant Gaussian function the image has been divided into circular regions in other words that regions are formed in a terrace shape so that to eliminate the rotation affect . The shape is divided into 11 terraces with a 0.1 width for each terrace . 9 terraces are resultant from the 0.1 width division which are; (1-0.9, 0.9-0.8, 0.8-0.7, 0.7-0.6, 0.6, 0.5, 0.5-0.4, 0.4-0.3, 0.3-0.2, 0.2-0.1), and one terrace for the terrace that has value smaller than 0.1 and the last one for the external area that extended out of the outer terrace . An explanation of this division is demonstrated in Figure 7.

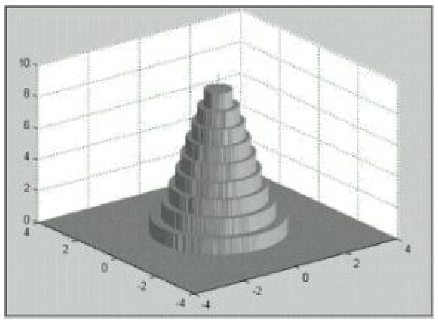


Figure 7. Terraces division with 0.1 likelihood .

Each terrace is divided into 8 sectors which named as the feature areas, empirically discovered that number 8 is suitable for features divisions , To attain best capturing of the Gaussian to fit the segmented hand, re-estimation are performed on the shape to fit capturing the hand object , then the Gaussian shape are matched on the segmented hand to prepare the final hand shape for extracting the features, Figure 8 shown this process .

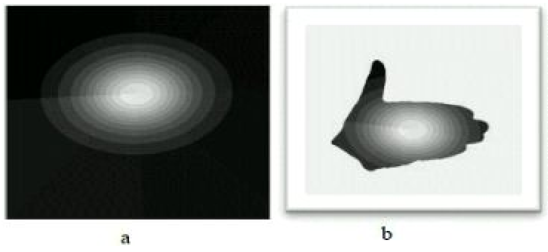


Figure 8. Features divisions . a) Terrace area in Gaussian. b) Terrace area in hand image.

After capturing the hand shape, two types of features are extracted to form the feature vector ; local feature, and global features. Local features using geometric central moments which provide two different moments μ00 , μ11 as shown by equation (1):

$$\mu_{pp} = \sum_{x}\sum_{y}(x-\mu_x)^p(y-\mu_y)^p\, f(x,y) \qquad (1)$$

$$\mu_{pp}^{(k)} = \sum_{y}\sum_{x}\left(x^{(k)}-\mu_x^{(k)}\right)^p\left(y^{(k)}-\mu_y^{(k)}\right)^p\, f\left(x^{(k)},y^{(k)}\right) \qquad (2)$$

$$\forall\, k \in \{1, 2, 3, \ldots, 88\}\ \&\ \forall\, p \in \{0, 1\}$$

Where μx and μy is the mean value for the input feature area , x and y are the coordinated,
and for this, the input image is represented by 88*2 features, as explained in detail in equation (2). While the global features are two features the first and second moments that are the computed for the whole hand features area . These feature areas are computed by multiplying feature area intensity plus feature area's map location . In this case, any input image is represented with 178 features . The system carried out using 20 different gestures , 10 samples for each gesture, 5 samples for training and 5 for testing, with 100% recognition percentage and it decreased when the number of gestures are more than 14 gestures . In 6 gestures are recognized with 10 samples for each gesture. Euclidian distance used for the classification of the feature .

Kulkarni recognize static posture of American Sign Language using neural networks algorithm. The input image are converted into HSV color model, resized into 80x64 and some image preprocessing operations are applied to segment the hand from a uniform background , features are extracted using histogram technique and Hough algorithm. Feed forward Neural Networks with three layers are used for gesture classification. 8 samples are used for each 26 characters in sign language, for each gesture, 5 samples are used for training and 3samples for testing, the system achieved 92.78% recognition rate using MATLAB language..

Hasan applied scaled normalization for gesture recognition based on brightness factor matching. The input image with is segmented using thresholding technique where the background is black. Any segmented image is normalized (trimmed), and the center mass of the image are determined, so that the coordinates are shifted to match the centroid of the hand object at the origin of the X and Y axis . Since this method depends on the center mass of the object, the generated images have different sizes see figure 9, for this reason a scaled normalization operation are applied to overcome this problem which maintain image dimensions and the time as well , where each block of the four blocks are scaling with a factor that is different from other block's factors. Two methods are used for extraction the features; firstly by using the edge mages,

and secondly by using normalized features where only the brightness values of pixels are calculated and other black pixels are neglected to reduce the length of the feature vector . The database consists of 6 different gestures, 10 samples per gesture are used, 5 samples for training and 5 samples for testing. The recognition rate for the normalized feature problem achieved better performance than the normal feature method, 95% recognition rate for the former method and 84% for the latter one .

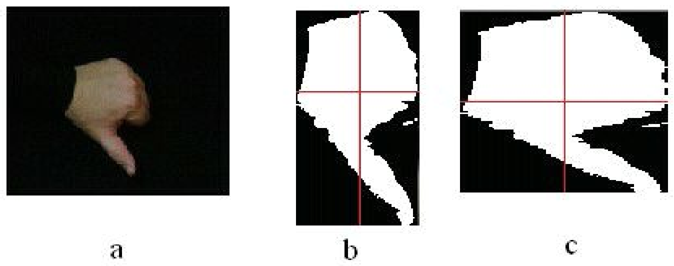


Figure 9. applying trimming process on the input image, followed by scaling normalization process .

Wysoski et al. [8] presented rotation invariant postures using boundary histogram. Camera used for acquire the input image, filter for skin color detection has been used followed by clustering process to find the boundary for each group in the clustered image using ordinary contourtracking algorithm. The image was divided into grids and the boundaries have been normalized. The boundary was represented as chord's size chain which has been used as histograms, by dividing the image into number of regions N in a radial form, according to specific angle. For classification process Neural Networks MLP and Dynamic Programming DP matching were used. Many experiments have implemented on different features format in addition to use different chord's size histogram, chord's size FFT. 26 static postures from American Sign

Language used in the experiments. Homogeneous background was applied in the work. Stergiopoulou [14] suggested a new Self-Growing and Self-Organized Neural Gas (SGONG) network for hand gesture recognition. For hand region detection a color segmentation technique based on skin color filter in the YCbCr color space was used, an approximation of hand shape morphology has been detected using (SGONG) network; Three features were extracted using finger identification process which determines the number of the raised fingers and characteristics of hand shape, and Gaussian distribution model used for recognition.

IV. RESEARCH PAPER

- PAPER 1

There are many types of Noise Models, as Noise is undesirable information in digital images. We need to get rid of it, here are some noise models :

1. Gaussian Noise Model
2. White Noise
3. Brownian Noise (Fractal Noise)
4. Impulse Valued Noise (Salt and Pepper Noise)
5. Periodic Noise
6. Quantization Noise
7. Speckle Noise
8. Photon Noise (Poisson Noise)
9. Poisson-Gaussian Noise
10. Structured Noise
11. Gamma Noise
12. Rayleigh Noise

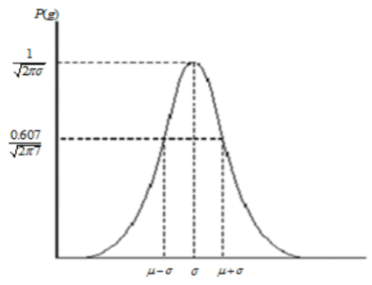
Let's discuss each Noise,

1. Gaussian Noise Model:It is also called as electronic noise because it arises in amplifiers or detectors. Gaussian noise caused by natural sources such as thermal vibration of atoms and discrete nature of radiation of warm objects.

Gaussian noise generally disturbs the gray values in digital images. That is why Gaussian noise model essentially designed and characteristics by its PDF or normalizes histogram with respect to gray value. This is given as

$$P(g) = \sqrt{\frac{1}{2\pi\sigma^2}}\, e^{-\frac{(g-\mu)}{2\sigma^2}}$$

Where g = gray value, σ = standard deviation and μ = mean. Generally Gaussian noise mathematical model represents the correct approximation of real-world scenarios. In this noise model, the mean value is zero, variance is 0.1 and 256 gray levels in terms of its PDF, which is shown



PDF of Gaussian Noise

Due to this equal randomness the normalized Gaussian noise curve look like in bell shaped. The PDF of this noise model shows that 70% to 90% noisy pixel values of degraded image in between μ σ μ σ − + and The shape of normalized histogram is almost same in spectral domain.

2. White Noise :Noise is essentially identified by the noise power. Noise power spectrum is constant in white noise. This noise power is equivalent to power spectral density function. The statement “Gaussian noise is often white noise” is incorrect [4]. However neither Gaussian property implies the white sense. The range of total noise power is -∞ to +∞ available in white noise in frequency domain. That means ideally noise power is infinite in white noise. This fact is fully true because the light emits from the sun has all the frequency components. In white noise, correlation is not possible because of every pixel values are different from their neighbours. That is why autocorrelation is zero. So that image pixel values are normally disturb positively due to white noise.

3. Brownian Noise (Fractal Noise): Colored noise has many names such as Brownian noise or pink noise or flicker noise or 1/f noise. In Brownian noise,

power spectral density is proportional to square of frequency over an octave i.e., its power falls on ¼ th part (6 dB per octave). Brownian noise caused by Brownian motion. Brownian motion seen due to the random movement of suspended particles in fluid. Brownian noise can also be generated from white noise, which is shown in Fig. 2. However this noise follows non stationary stochastic process. This process follows normal distribution. Statistically fractional Brownian noise is referred to as fractal noise. Fractal noise is caused by natural process. It is different from Gaussian process

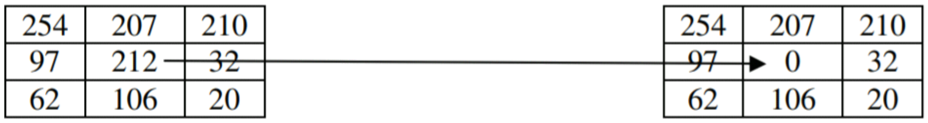
Although power spectrum of fractal noise, decaying continuously due to increase in frequency. Fractal noise is almost singular everywhere. A fractional Brownian motion is mathematically representing as a zero mean Gaussian process (BH) which is showing in equation (2) and equation (3), respectively

$BH(0)=0$..(2)

And expected value of fractional Brownian motion is

$E\{|BH(t)-BH(t-\Delta)|^2\}= \sigma^2|\Delta|^{2H}$..(3)

4. Impulse Valued Noise (Salt and Pepper Noise) :This is also called data drop noise because statistically its drop the original data values. This noise is also referred as salt and pepper noise. However the image is not fully corrupted by salt and pepper noise instead of some pixel values are changed in the image. Although in noisy image, there is a possibility of some neighbors does not changed [13-14]. This noise is seen in data transmission. Image pixel values are replaced by corrupted pixel values either maximum 'or' minimum pixel value i.e., 255 'or' 0 respectively, if number of bits are 8 for transmission. Let us consider 3x3 image matrices which are shown in the Fig. 3. Suppose the central value of matrices is corrupted by Pepper noise. Therefore, this central value i.e., 212 is given in Fig. 3 is replaced by value zero. In this connection, we can say that, this noise is inserted dead pixels either dark or bright. So in a salt and pepper noise, progressively dark pixel values are present in bright region and vice versa



254	207	210
97	212	32
62	106	20

254	207	210
97	0	32
62	106	20

The central pixel value is corrupted by Pepper Noise

Inserted dead pixel in the picture is due to errors in analog to digital conversion and errors in bit transmission. The percentagewise estimation of noisy pixels, directly determine from pixel metrics. PDF of noise is given below

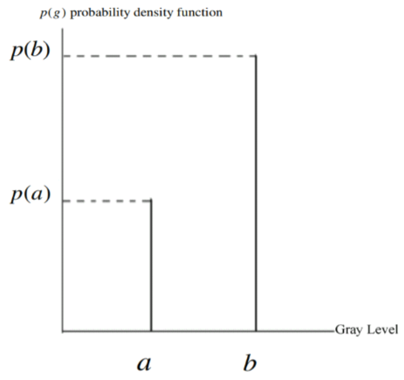


Figure 4. The PDF of Salt and Pepper noise

$$P(g)=\begin{cases} Pa \text{ for } g=a \\ Pb \text{ for } g=b \\ 0 \text{ otherwise} \end{cases} \quad (4$$

This shows the PDF of Salt and Pepper noise, if mean is zero and variance is 0.05. Here we will meet two spike one is for bright region (where gray level is less) called 'region a' and another one is dark region (where gray level is large) called 'region b', we have clearly seen here the PDF values are minimum and maximum in 'region a' and 'region b', respectively [16]. Salt and Pepper noise generally corrupted the digital image by malfunctioning of pixel elements in camera sensors, faluty memory space in storage, errors in digitization process and many more.

5. Periodic Noise: This noise is generated from electronics interferences, especially in power signal during image acquisition. This noise has special characteristics like spatially dependent and sinusoidal in nature at multiples of specific frequency. It's appears in form of conjugate spots in frequency domain. It can be conveniently removed by using a narrow band reject filter or notch filter.

6. Quantization noise : Quantization noise appearance is inherent in amplitude quantization process. It is generally presents due to analog data converted into digital data. In this noise model, the signal to noise ratio (SNR) is limited by minimum and maximum pixel value, Pmin and Pmax respectively. The SNR is given as

$$SNR_{dB} = 20\log_{10}(P_{\max} - P_{\min})/\sigma_n \quad (5)$$

Where σ_n = Standard deviation of noise, when input is full amplitude sine wave SNR becomes

$$SNR = 6n + 1.76 \text{ dB} \quad (6)$$

Where n is number of bits. Quantization noise obeys the uniform distribution. That is why it is referred as uniform noise. Its PDF is shown in Fig. 5.

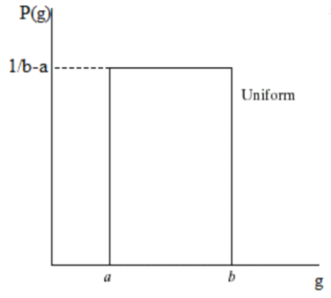


Figure 5 Uniform noise

$$P(g)=\begin{cases}\frac{1}{b-a} & \text{if } a \le g \le b\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

and their mean $\mu=\frac{a+b}{2}$ and variance $\sigma^2=\frac{(b-a)^2}{12}$

7. Speckle Noise :This noise is multiplicative noise. Their appearance is seen in coherent imaging system such as laser, radar and acoustics etc,. Speckle noise can exist similar in an image as Gaussian noise. Its probability density function follows gamma distribution, which is shown in Fig. 6 and given as in equation (8) [17-19].

$$F(g)=\frac{g^{\alpha-1}e^{\frac{-g}{a}}}{\alpha-1!a^{\alpha}} \tag{8}$$

Speckle Noise with Variance 0.04

8. Photon Noise (Poisson Noise): The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient’s body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise. This noise obeys the Poisson distribution and is given as

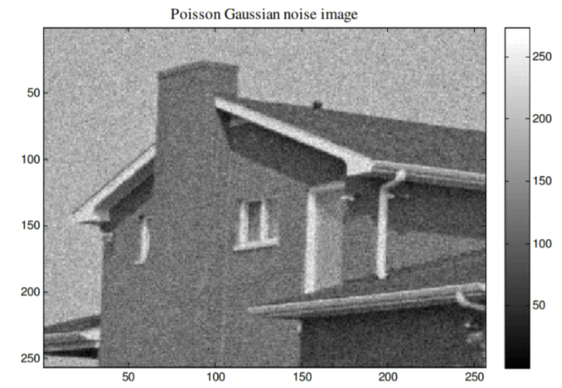
$$P(f_{(pi)}=k)=\frac{\lambda^{k}{}_{i}e^{-\lambda}}{k!} \tag{9}$$

9. Poisson-Gaussian Noise: In this section the proposed model work for removing of Poisson-Gaussian noise that is arose in Magnetic Resonance Imaging (MRI). Poisson-Gaussian noise is shown in Fig. 7. The paper

introduces the two most fantastic noise models, jointly called as Poisson-Gaussian noise model. These two noise models are specified the quality of MRI recipient signal in terms of visual appearances and strength [21]. Despite from the highest quality MRI processing, above model describes the set of parameters of the Poisson-Gaussian noise corrupted test image. The Poisson Gaussian noise model can be illustrated in the following manner.

$$Z(j,k) = \alpha * P_{\alpha}(j,k) + N_{\alpha}(j,k) \tag{10}$$

Where, the model has carried Poisson-distribution (p_{α}) follows with Gaussian-distribution (n_{α}). Poisson-distribution estimating using mean $(\mu_{\alpha 1})$ at given level $\alpha > 0$ and Gaussian-distribution counted at given level $\alpha > 0$ using zero mean $(\mu_{\alpha 2})$ and variance $\sigma^2_{\alpha 2}$. To evaluate $\mu_{\alpha 1}$, noisy Poisson image was quantitatively added to underlying original image. After all to get a noisy image $Z(j,k)$ from the Poisson-Gaussian model is based on [22-23].



Poisson-Gaussian Noise House Image

10. Structured Noise: Structured noise are periodic, stationary or non-stationary and aperiodic in nature. If this noise is stationary, it has fixed amplitude, frequency and phase. Structured noise caused by interferences among electronic components [24]. Noise presents in communication channel are in two parts, unstructured noise (u) and structured noise (s). structured noise is also called low rank noise. In a signal processing, it is more advantagable (more realistic) to considering noise model in a lower dimensionality space. Further, this model is mapped into full rank measurement space in physical system. So we can conclude that in the measurement space, resulting noise has low rank and exhibits structure dependent on physical system. Structured noise model is showing in equation (11) and equation (12), respectively .

$$y_{(n)} = x_{(n,m)} + v_{(n)} \tag{11}$$

$$y_{(n)} = H_{(n,m)} * \theta_{(m)} + S_{(n,t)} * \phi_{(t)} + v_{(n)} \tag{12}$$

Where, n = rows, m = columns, y = received image, H = Transfer function of linear system, S = Subspace, t = rank in subspace, ϕ = underlying process exciting the linear system (S), θ = signal parameter sets initial conditions or excites, linear system H is used to produce original signal x in terms of n vector random noise $\left(v_{(n)}\right)$.

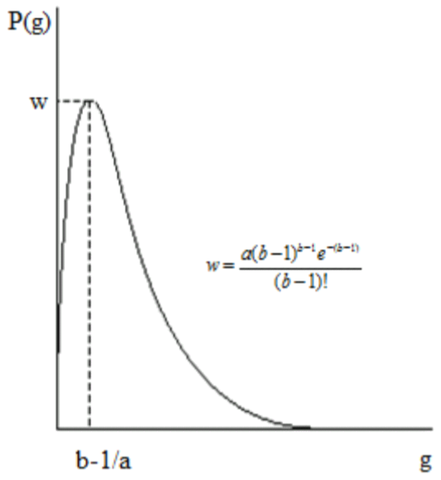
Structured Noise (when noise is periodic and non-stationary)

Structured Noise

11. Gamma Noise: Gamma noise is generally seen in the laser based images. It obeys the Gamma distribution. Which is shown below

$$P(g) = \begin{cases} \dfrac{a^b g^{b-1} e^{-ag}}{(b-1)!} & \text{for } g \geq 0 \\ 0 & \text{for } g < 0 \end{cases}$$

Where mean $\mu = \frac{b}{a}$ and variance $\sigma^2 = \frac{b}{a^2}$ are given as, respectively.

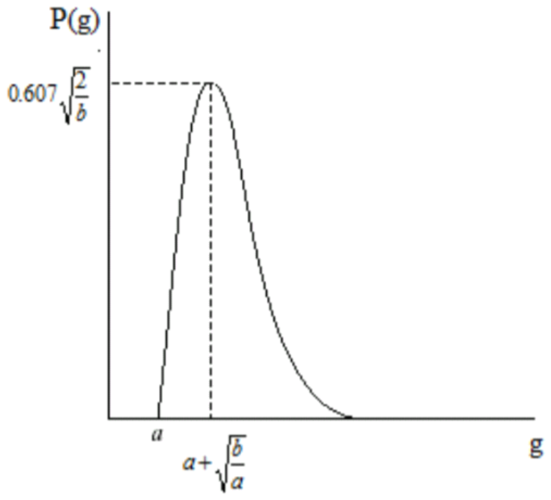


Gamma Distribution

12. Rayleigh noise Rayleigh noise presents in radar range images. In Rayleigh noise, probability density function is given as

$$P(g) = \begin{cases} \dfrac{2}{b}(g-a)e^{\frac{-(g-a)^2}{b}} & \text{for } g \geq a \\ 0 & \text{for } g < a \end{cases} \qquad (14)$$

Where mean $\mu = a + \sqrt{\frac{\pi b}{4}}$ and variance $\sigma^2 = \frac{b(4-\pi)}{4}$ are given as, respectively.



Rayleigh Distribution

CONCLUSION:

During image acquisition and transmission, noise is seen in images. This is characterised by noise model. So study of noise model is very important part in image processing. On the other hand, Image denoising is necessary action in image processing operation. Without the prior knowledge of noise model we cannot elaborate and perform denoising actions. Hence, here we have reviewed and presented various noise models available in digital images. We addressed that noise models can be identified with the help of their origin. Noise models also designed by probability density function using mean, variance and mainly gray levels in digital images. We hope this work will provide as a susceptible material for researchers and of course for freshers in the image processing field.

- PAPER 2

Filter

A filter is a device that discriminates according to one or more attributes at its input, what passes through it.

One example is the color filter which absorbs light at certain wavelengths.

By filter design we can create filters that pass signals with frequency components in some bands, and attenuate signals with content in other frequency bands.

Required Classification as per Requirement

1. Low Pass Filter

A low-pass filter is a filter that passes low frequencies but attenuates higher than the cutoff frequency.

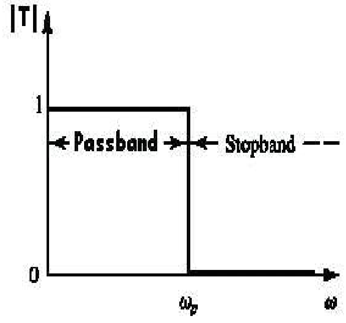


Figure 2

2. High Pass Filter

A high-pass filter is a filter that passes high frequencies well, but attenuates frequencies lower than the cut-off frequency.

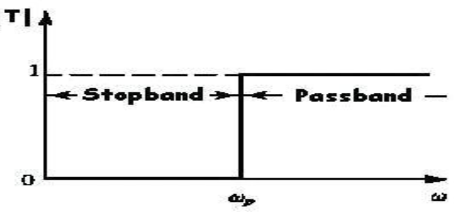


Figure 3

If we combine the above two together, we can design a filter that starts as a low-pass filter and slowly allows higher frequency components also and finally all frequencies can pass through that filter and we get the whole image.

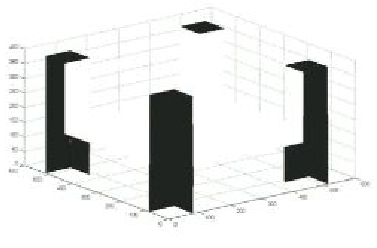


Figure 4

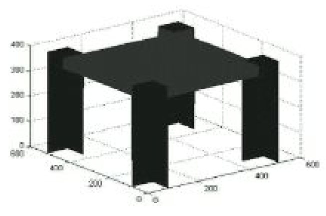
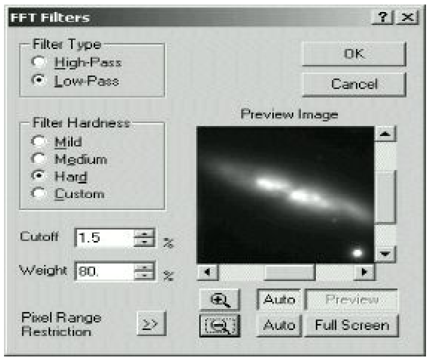


Figure 5

3. FFT Filter

FFT Filters provide precisely controlled low- and high-pass filtering (smoothing and sharpening, respectively) using a Butterworth characteristic. The image is converted into spatial frequencies using a Fast Fourier Transform, the appropriate filter is applied, and the image is converted back using an inverse FFT.



FFT Filters

Filter Type: High-Pass [✓ Low-Pass]

Filter Hardness: Mild Medium [✓ Hard] Custom

Cutoff 1.5 %

Weight 80. %

Image Selection

Many grey-level images are available for the purpose of showing the function and working of a filter but for standard conventions I decided to choose Lenna.

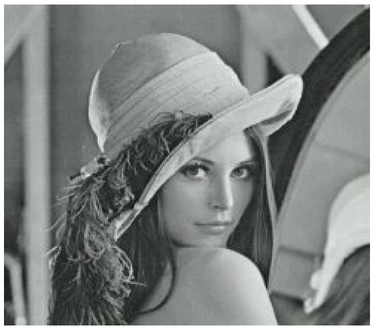


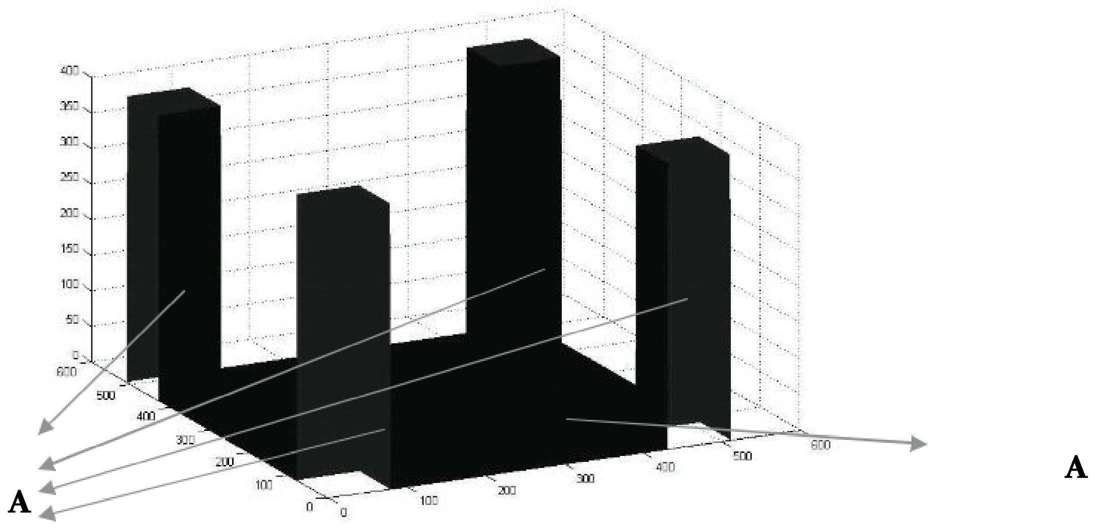
Figure 6

Matrix Representation

The image chosen is now scaled to a fixed size of (512x512) and represented as a matrix. One important thing that has to be kept in mind is that image must have its both dimensions of at least 512 or else there will be a run-time error.

Area Division

The (512x512) matrix is divided into two major portions. First of all we separate a fixed square size area (say 60x60) from all corners of the image and we assume that in any of the functions applied this area will nor be taken into account. This area will simply act as pillars of the filter. The rest of the area left comprises of the second major portion of the image matrix.



Fig

The Working

Phase - 1

Since MATLAB was very new to me, I started off by calculating the FFT of an image and observed some important things. When the FFT was calculated the entire image seemed to show noise all over. Since no boundary had been set for the frequencies, hence noise intrusion was very large and hence NOISE.

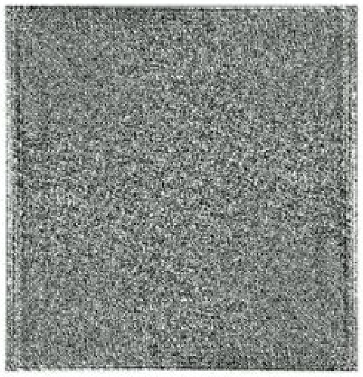
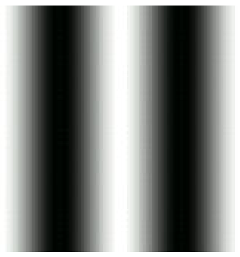


Figure 8

When 2-D FFT was calculated, proceedings made much more sense. Unless I processed a completely black image, a 2D Fourier transform of an image file (where all pixels have positive values) will always have a bright pixel in the center. That center pixel is called the DC term and represents the average brightness across the entire image.



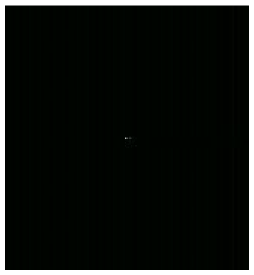


Figure 9 **Figure 10**

On the left side is an image of a sine wave where black pixels represent the bottom of the sine wave, white pixels the top and the gray pixels in between represent the sloping areas of the curve. On the right is the FFT of that image.

2D Fourier transforms are always symmetrical. The upper left quadrant is identical to the lower right quadrant and the upper right quadrant is identical to the lower left quadrant. This is a natural consequence of how Fourier transforms work.

When the IFFT of the image was calculated, a white screen appeared showing nothing. The reason was; since the initial 2-D FFT was nothing but noise so nothing appeared, but when the mesh/surface plot of the image was viewed, I saw the following interesting result.

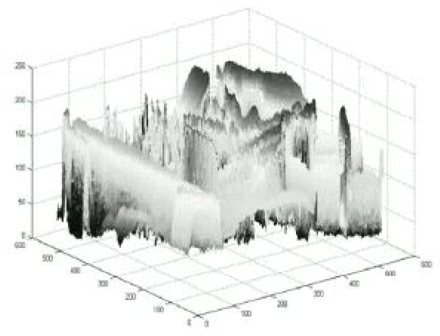


Figure11

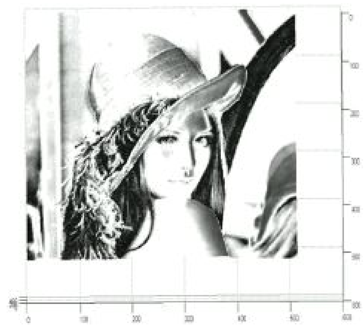


Figure 12

Figure 11 clearly depicts that when the MESH function was used, the colored parametric mesh defined by the matrix arguments was visible.

On rotating it in a 3-D manner, Lenna was clearly visible as in **Figure 12**.

The reason why Lenna could not be seen was that all the values after calculating the FFT were out of the specified range i.e. they were not NORMALIZED. The moment IFFT was calculated the values were back in range but there were some differences in the original values of the image matrix and the ones obtained after IFFT.

The values were not same because in the process of NORMALIZATION round-off functions are used which might change the values at some point of time.

A few important functions that had to be used were:

RGB2GRAY : Converts RGB image to grayscale by eliminating the hue and saturation information while retaining the luminance.

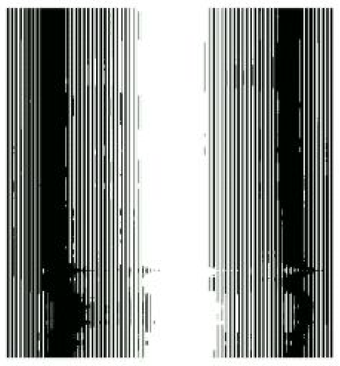
IM2DOUBLE : Convert image to double precision. IM2DOUBLE takes an image as input, and returns an image of class double. If the input image is of class double, the output image is identical to it. If the input image is not double, IM2DOUBLE returns the equivalent image of class double, rescaling or offsetting the data as necessary.

Once I understood the basic concepts, I moved ahead with the filter thing. After making the side pillars, now the sheet area that was in between those pillars had to be moved up slowly so that it acts as the frequency limit till where the frequencies could enter.

Phase – 3

To move the inner sheet up so that it acts as a frequency cut-off limiter, a code was written that limited only the inner area to move and not the entire block along with the sheet. Once this was achieved, I had to multiply the FFT of the image with this filter function that had been designed.

After multiplication the IFFT of the entire result was calculated and the filtered result was viewed for various values. The same result kept on repeating.



.

Figure 13

NORMALIZATION once again comes into picture. Since the final result had not been subjected to the specified range the result could not be viewed.

After all corrections results for various tests are displayed below:

Result-1

Filter window size 10 (Only very few low-range frequencies are allowed to pass)

Image:

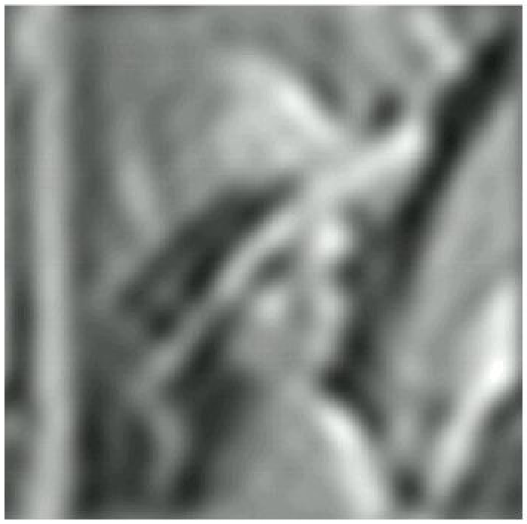


Figure 14

Result-2

Filter window size 50 (A few more frequencies are allowed to pass)

Image:

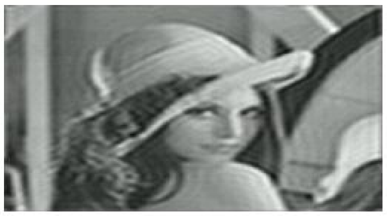


Figure 15

Result-3

Filter window size 180 (A lot more frequencies are allowed to pass)

Image:

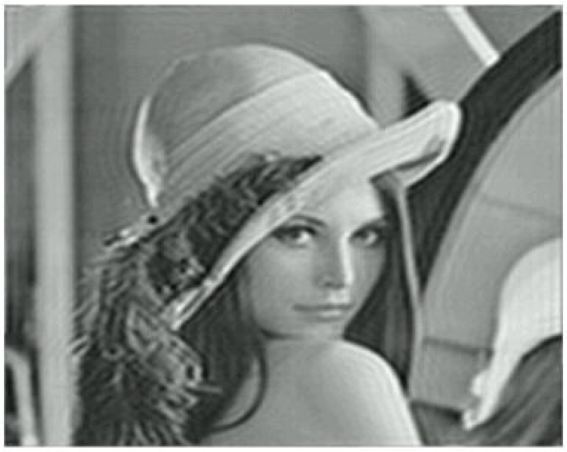


Figure 16

As you can see, by allowing the window size to increase and bringing the resultant frequencies within the range, the resulting filtered image becomes more and more clear.

Phase-4

Calculation of MSQE

Mean square quantization error (MSQE) is a figure of merit for the process of analog to digital conversion.

As the input is varied, the input's value is recorded when the digital output changes. For each digital output, the input's difference from ideal is normalized to the value of the least significant bit, then squared, summed, and normalized to the number of samples.

MSQE calculations were carried out for all the results abtained after filtering and were within permissible range i.e **-3.6% to +3.6%** of the original value.

Phase-5

Size of Outer Squares

The area that comprised of the four outer squares also plays a very important part in the final outlook of the image. Larger the square size, sharper is the final image.

The reason of such an outcome is that when the squares are chosen, the particular area is not being operated by any of the functions. Whenever a function acts on the matrix the image covered under the four squares is untouched and in the final output appears as it is. Hence there is a major change in the final image as we increase or decrease the size of the squares.

Observation : Larger is the size of the outer squares, better is the final image.

For an example, a few of the images have been shown below in increasing order of size of squares.

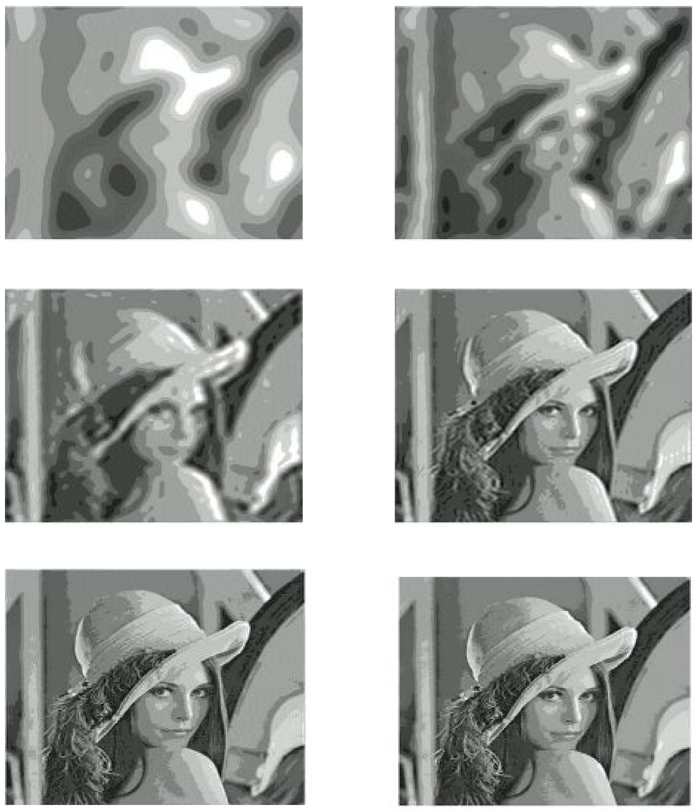


Figure-17

v. CODE

```
import cv2

import numpy as np

import math

cap = cv2.VideoCapture(0)

while(1):

    try:  #an error comes if it does not find anything in window as it cannot find contour of max
area

          #therefore this try error statement

        ret, frame = cap.read()

        frame=cv2.flip(frame,1)

        kernel = np.ones((3,3),np.uint8)

        #define region of interest

        roi=frame[100:300, 100:300]

        cv2.rectangle(frame,(100,100),(300,300),(0,255,0),0)

        hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

    # define range of skin color in HSV

        lower_skin = np.array([0,20,70], dtype=np.uint8)

        upper_skin = np.array([20,255,255], dtype=np.uint8)

     #extract skin colur imagw

        mask = cv2.inRange(hsv, lower_skin, upper_skin)

    #extrapolate the hand to fill dark spots within

        mask = cv2.dilate(mask,kernel,iterations = 4)

    #blur the image

        mask = cv2.GaussianBlur(mask,(5,5),100)
```

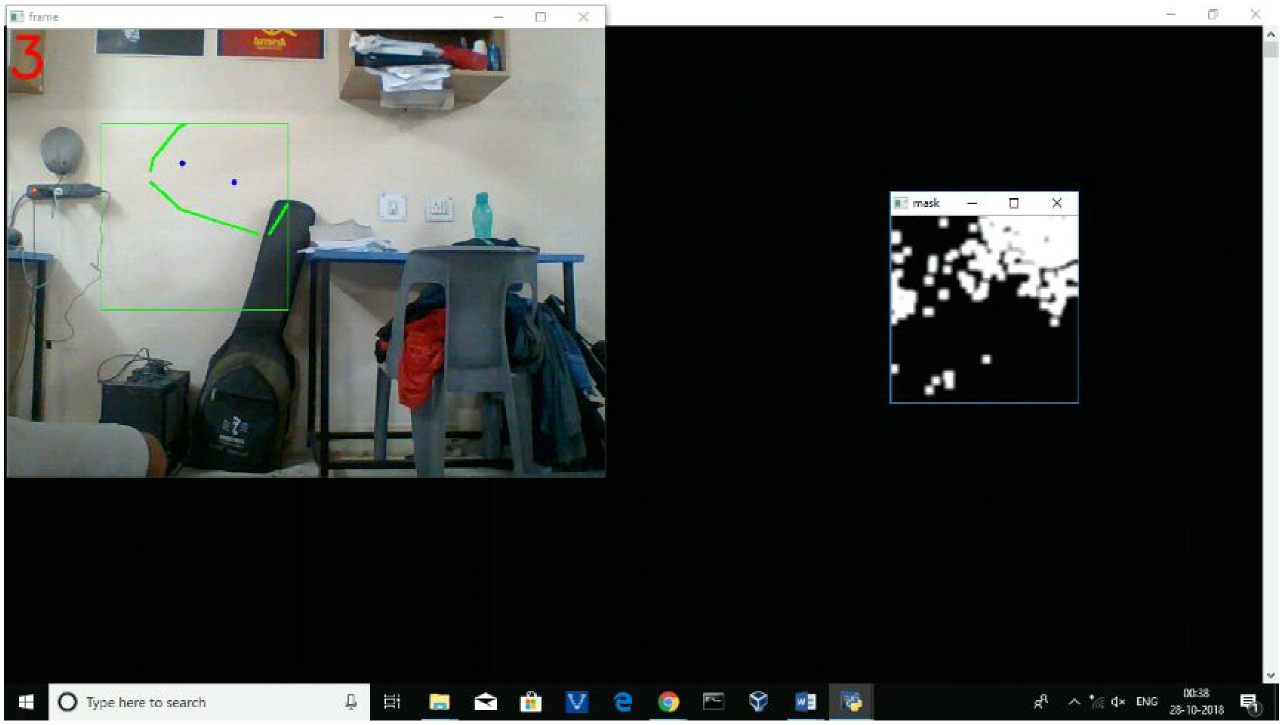
```
   #find contours

       _,contours,hierarchy=
cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE

  #find contour of max area(hand)

       cnt = max(contours, key = lambda x: cv2.contourArea(x))

   #approx the contour a little

       epsilon = 0.0005*cv2.arcLength(cnt,True)

       approx= cv2.approxPolyDP(cnt,epsilon,True)

   #make convex hull around hand

       hull = cv2.convexHull(cnt)

    #define area of hull and area of hand

       areahull = cv2.contourArea(hull)

       areacnt = cv2.contourArea(cnt

   #find the percentage of area not covered by hand in convex hull

       arearatio=((areahull-areacnt)/areacnt)*100

    #find the defects in convex hull with respect to hand

       hull = cv2.convexHull(approx, returnPoints=False)

       defects = cv2.convexityDefects(approx, hull)

   # l = no. of defects

       l=0

   #code for finding no. of defects due to fingers

       for i in range(defects.shape[0]):

           s,e,f,d = defects[i,0]

           start = tuple(approx[s][0])

           end = tuple(approx[e][0])

           far = tuple(approx[f][0])
```

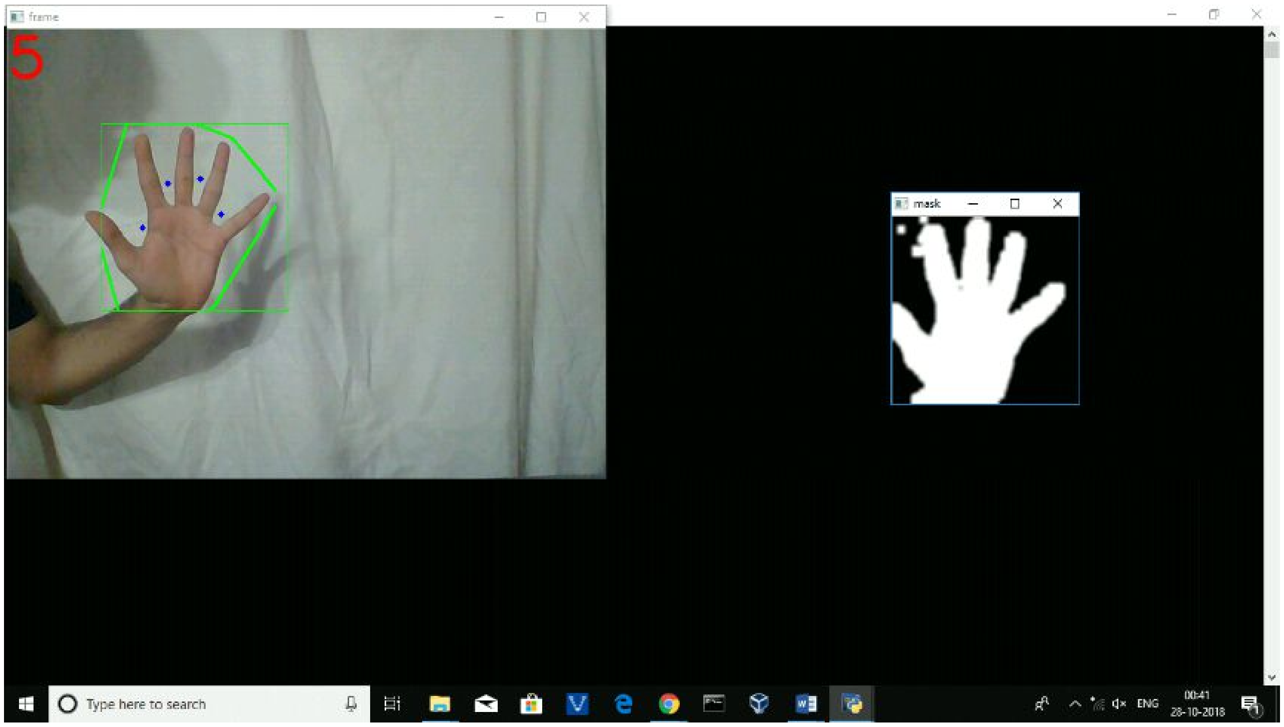
```
            pt= (100,180)

            # find length of all sides of triangle

            a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)

            b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)

            c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)

            s = (a+b+c)/2

            ar = math.sqrt(s*(s-a)*(s-b)*(s-c))

            #distance between point and convex hull

            d=(2*ar)/a

            # apply cosine rule here

            angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 57

            # ignore angles > 90 and ignore points very close to convex hull(they generally come due
    to noise)

            if angle <= 90 and d>30:

                l += 1

                cv2.circle(roi, far, 3, [255,0,0], -1)


            #draw lines around hand

            cv2.line(roi,start, end, [0,255,0], 2)

        l+=1

        #print corresponding gestures which are in their ranges

        font = cv2.FONT_HERSHEY_SIMPLEX

        if l==1:

            if areacnt<2000:

                cv2.putText(frame,'Put hand in the box',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

            else:
```

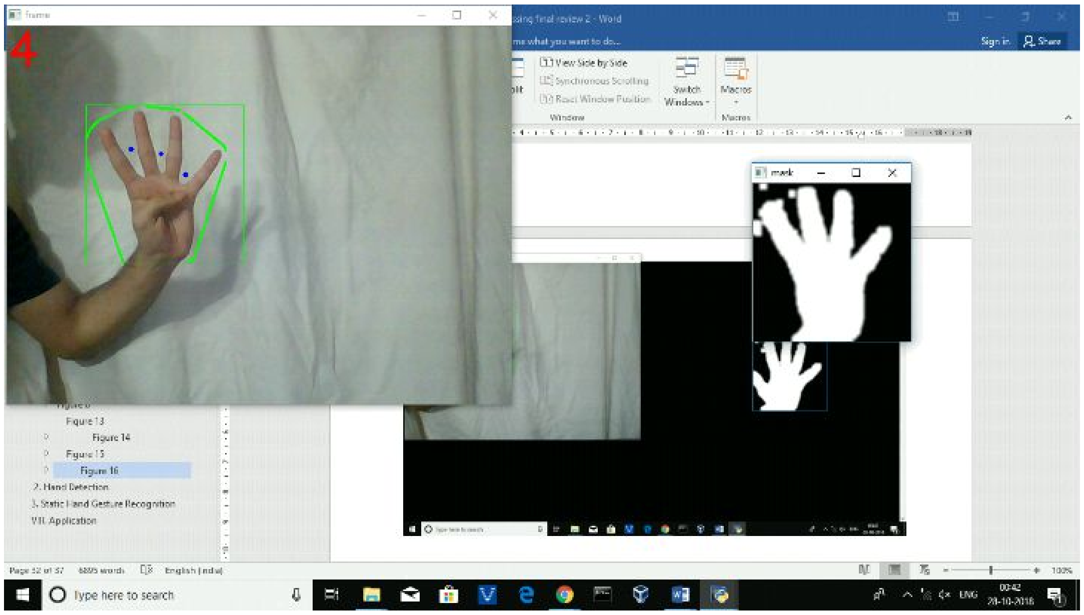
```
          if arearatio<12:

             cv2.putText(frame,'0',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

          elif arearatio<17.5:

             cv2.putText(frame,'Best of luck',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

          else:

             cv2.putText(frame,'1',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==2:

       cv2.putText(frame,'2',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==3:

         if arearatio<27:

             cv2.putText(frame,'3',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

         else:

             cv2.putText(frame,'ok',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==4:

       cv2.putText(frame,'4',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==5:

       cv2.putText(frame,'5',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    elif l==6:

       cv2.putText(frame,'reposition',(0,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    else :

       cv2.putText(frame,'reposition',(10,50), font, 2, (0,0,255), 3, cv2.LINE_AA)

    #show the windows

    cv2.imshow('mask',mask)

    cv2.imshow('frame',frame)

 except:
        pass

    k = cv2.waitKey(5) & 0xFF

    if k == 27:

        break

cv2.destroyAllWindows()

cap.release()
```

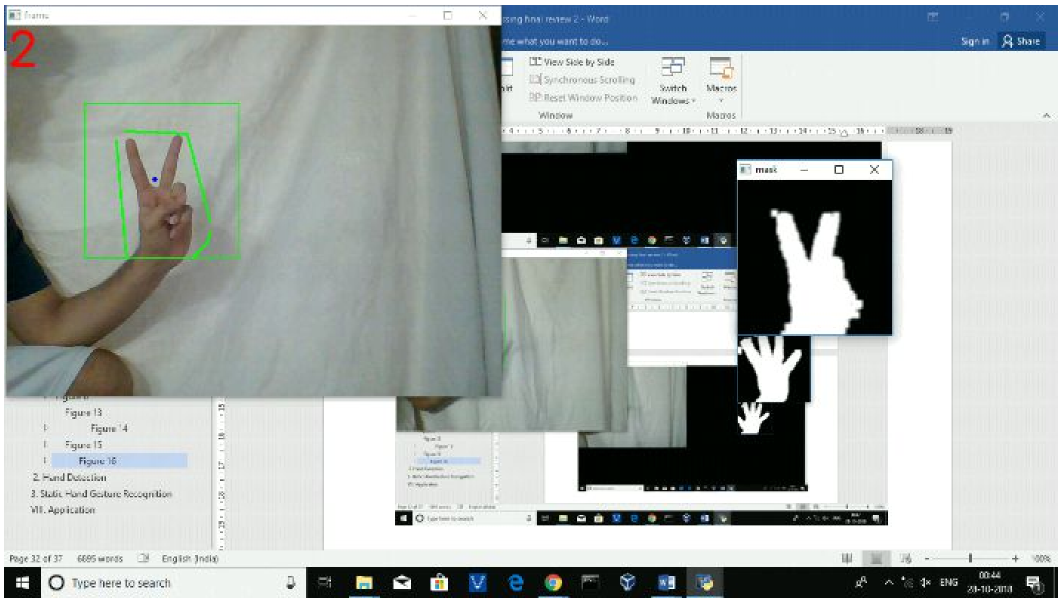
VI. OUTPUT

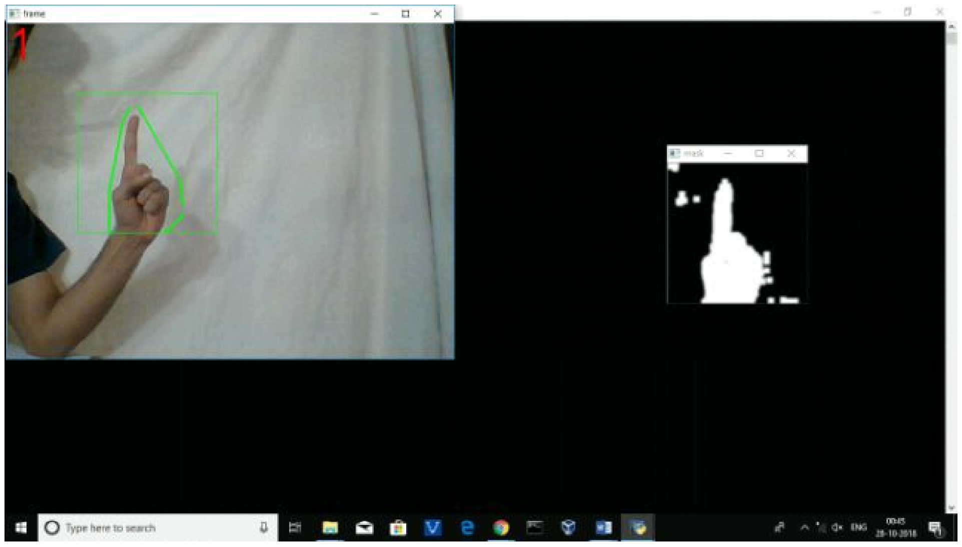
1. Skin Segmentation

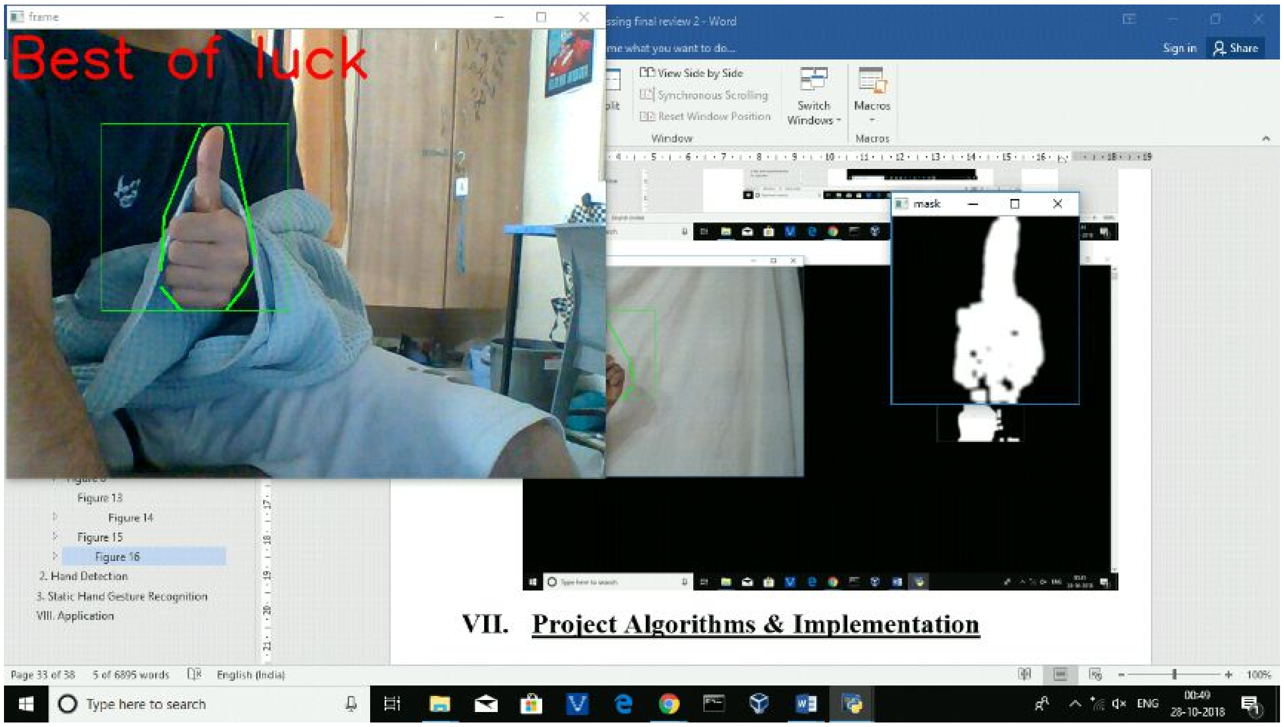


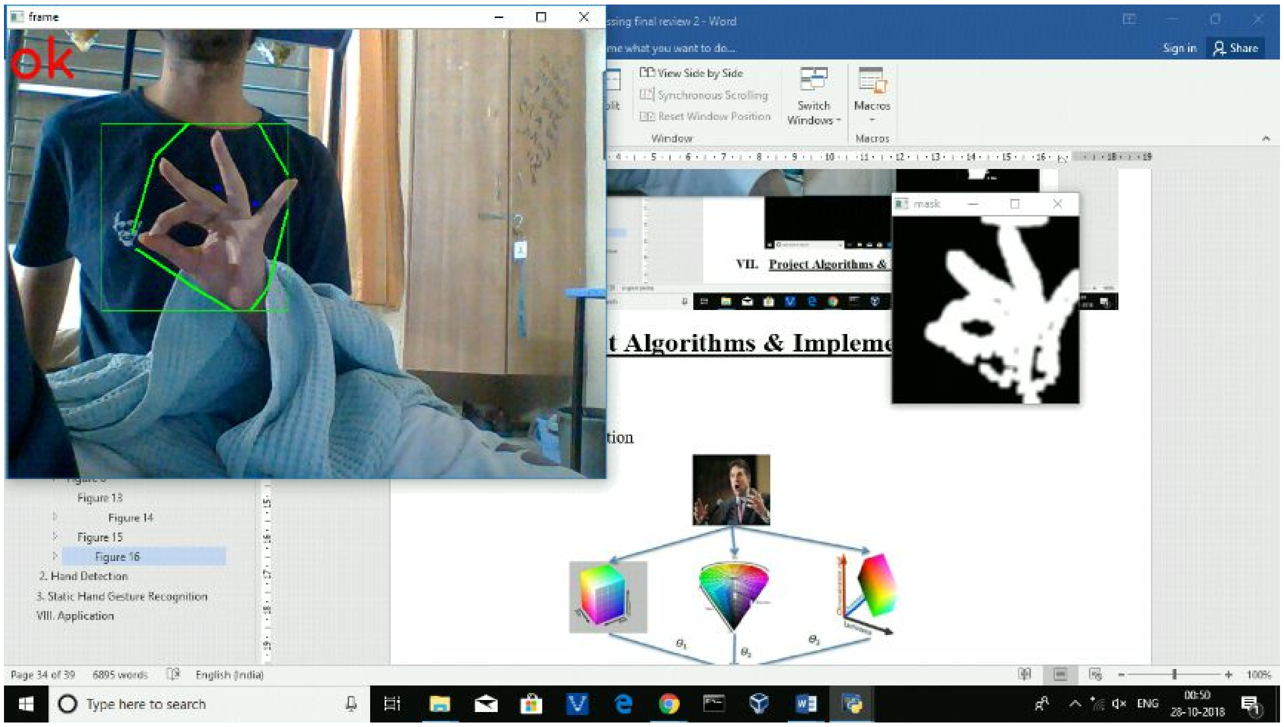



frame
4
mask
View Side by Side
Synchronous Scrolling
Reset Window Position
Switch Windows
Macros
Figure 13
Figure 14
Figure 15
Figure 16
2. Hand Detection
3. Static Hand Gesture Recognition
VII. Application
Page 32 of 37
6695 words
English (India)
Type here to search
00:42
28-10-2018


frame
2
mask
View Side by Side
Synchronous Scrolling
Reset Window Position
Switch Windows
Macros
Figure 13
Figure 14
Figure 15
Figure 16
2. Hand Detection
3. Static Hand Gesture Recognition
VII. Application
Page 32 of 37
6695 words
English (India)
Type here to search
00:44
28-10-2018


frame
1
mask
Type here to search
00:45
28-10-2018


frame
Best of luck
mask
Figure 13
Figure 14
Figure 15
Figure 16
2. Hand Detection
3. Static Hand Gesture Recognition
VIII. Application
VII. Project Algorithms & Implementation
Page 33 of 38
5 of 6895 words
English (India)
Type here to search


frame
ok
mask
Figure 13
Figure 14
Figure 15
Figure 16
2. Hand Detection
3. Static Hand Gesture Recognition
VIII. Application
Page 34 of 39
6895 words
English (India)
Type here to search

VII. Project Algorithms & Implementation

2. skin Segmentation

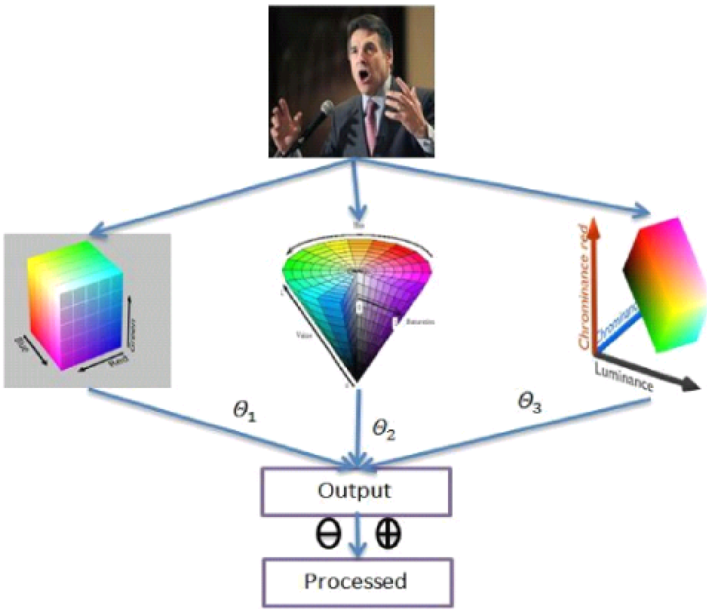


Figure 1. Skin segmentation system.

The figure above describes an overview of our skin segmentation system. Each input frame is converted to HSV and YCbCr color spaces (input frame is in RGB color space). Then, each color space is applied its thresholded using the specified boundaries found below. In RGB color space, the boundaries rules are the following:

$$R > 50 \ \&\& \ G > 40 \ \&\& \ B > 20 \ \&\& \ (\max(R,G,B) - \min(R,G,B)) > 10 \ \&\& \ abs(R-G) \geq 10 \ \&\& \ R > G \ \&\& \ R > B \qquad (1)$$

$$R > 220 \ \&\& \ G > 210 \ \&\& \ B > 170 \ \&\& \ abs(R-G) < 15 \ \&\& \ R > B \ \&\& \ G > B \qquad (2)$$

In the HSV and YCbCr color spaces, the boundaries are described below. In each of these color spaces, we only use two channels: H and V, Cb and Cr for HSV and YCbCr color space, respectively.

$$Cb \geq 60 \ \&\& \ Cb \leq 130 \qquad (3)$$

$$Cr \geq 130 \ \&\& \ Cr \leq 165 \qquad (4)$$

$$H \geq 0 \ \&\& \ H \leq 50 \qquad (5)$$

$$S \geq 0.1 \ \&\& \ S \leq 0.9 \qquad (6)$$

With the given boundaries rules above, a particular pixel is classified as a skin color if and only if it passes the following conditional test:

```
If( (1) || (2) ){
        If( (3) && (4) ){
                If( (5) && (6) ){
                        Pixel is skin
                }
        }
}
```

In order to correctly distinguish skin from non-skin pixels using the given boundary conditions, the right conversion methods between RGB and the HSV and YCbCr color spaces is essential. In the HSV color space, hue channel has to be in range of 0^o-360^o whereas saturation channel has to be between 0 and 1. In case of YCbCr color space, both Cb and Cr channels have to be between 16 and 240. The equations below give the proper transformations from RGB color space to HSV and YCbCr color space.

$$\begin{aligned} Cb &= 128 - 0.148 * R - 0.291 * G + 0.439 * B \\ Cr &= 128 + 0.439 * R - 0.368 * G + 0.071 * B \end{aligned} \qquad [4]$$

$$R' = \frac{R}{255}; G' = \frac{G}{255}; B' = \frac{B}{255}$$

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

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

$$H = \begin{cases} 0^\circ & ; \Delta = 0 \\ 60^\circ * \left(\frac{G' - B'}{\Delta} \% 6\right) & ; C_{max} = R' \\ 60^\circ * \left(\frac{B' - R'}{\Delta} + 2\right) & ; C_{max} = G' \\ 60^\circ * \left(\frac{R' - G'}{\Delta} + 4\right) & ; C_{max} = B' \end{cases}$$

$$S = \begin{cases} 0 & ; C_{max} = 0 \\ \frac{\Delta}{C_{max}} & ; C_{max} \neq 0 \end{cases} \qquad [5]$$

There do exist some false positive skin pixel values. In order to eliminate those false positives - generally in the form of static white noise - we apply morphological operations. We use OpenCV functions *erode* and *dilate* to perform closing operation. For these functions, we chose structure element with size of 3x3 and 5x5 for *erode* and *dilate*, respectively.

**Note: In order to ease the amount of processing power necessary for our program, we designate only a subregion of original input frame to do skin segmentation. This also provides the user with a specific region to actually present static gestures.

2. Hand Detection

From the procedure above, we are ensured a skin segmented mask. With that given segmented mask, we apply the centroid equation explained in the problem formulation section above. These equations ideally outputs the (x,y) coordinates for the center of the palm. Below you will find these equations.

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_i}{i}; \ \bar{y} = \frac{y_1 + y_2 + \cdots + y_j}{j}$$ [6]

Because the pixels in the mask are either white or black (i.e. 1 or 0), the equation is polarized to only skin pixel values.

3. Static Hand Gesture Recognition

As described in the problem formulation, we draw a rectangle around the center of the hand, whose coordinates are approximated by the centroid equation above. The size of this rectangle is calculated by finding distance from the center coordinates and the last counted skin pixel value in the matrix. Note that the “last” skin pixel value is simply the last set of coordinates with a white pixel when the matrix is initially traversed during the centroid calculations. i.e. Since the matrix is traversed using for loops - rows and then columns - the bottom row is usually where the last white pixel value resides. From this distance *d* calculated above, we draw a rectangle that has length of twice the calculated distance *d* with center of that rectangle corresponds to the center of the segmented hand. This proportion was tested and most often drew the edges of the rectangle in the correct location. Here, we make one large assumption: the user uses his right hand. Given that the box for computations is positioned to the right of the user, it is more intuitive. Still it is necessary to note.

Assuming the right hand is used and the rectangle is drawn such that its edges cut across the fingers of the hand - when the fingers are fully extended - we can simply count the number of white & black changes happen across the line:

```
Given:    Line of top edge of rectangle (x0,y 0) → (x0,y N)
          N = length of line
          SkinMask = Matrix corresponding to mask image
          pixCount = 0 , counts number of skin pixels in a row
          FingerThreshold = 10 , # of consecutive skin pixels needed in order to
               classify as a finger.
 for j = 0 --> N if
          SkinMask(x0,y j) == 1:
               pixCount++
          else:
               if pixCount > FingerThreshold:
                    number of fingers++
```

This method is not limited to horizontal counting, as it can also be applied to vertical lines of the rectangle. From this method, we can count the number of finger the user is holding up. From this feature, we can create a multitude of different gestures.

Figure 4. Snake game gesture command diagram

VIII. Application

After integrating the hand gesture recognition with the Snake game, we had a challenge of handling the gesture command processing delay. In the game implementation, every unit step movement of the snake is instructed by user's input in real time. In the earlier version, every frame is interpreted as a command, we found it creates some error as the unintended gesture can be captured during the finger movements of gesture change. Additionally, it is very hard to change hand gesture at the right moment in order to move the snake to the desired spot which leads to a less friendly control experiences.

To overcome this challenge, we first analyzed the frame-to-frame elapse time, which starts from the back-end of the game reading in one frame, performing skin detection and gesture recognition until the interpreted command is concluded, and then the front-end of the game updates the GUI. This process is then repeated again and again between frames. By testing and running the game with timestamps, we found the average frame processing time is about 130 milliseconds. We then adjusted the game to accept a command if the number of repeated hand gesture frames has met a threshold. For an example, if the threshold is set to 1, then one consecutive hand gesture frame with the same command as the previous hand gesture frame meets the threshold, thus will be send as a single command to the front-end of the game to move the snake. According to Humanbenchmark.com[6], a site has collected over 18 million human clicks for reaction benchmarks, the average human reaction time is 266 milliseconds and the median lays at 256 milliseconds. Based on this statistical data and our testing, setting the threshold to either 1 (every 2 repeated frames -> 260 milliseconds per command) or 2 (every 3 repeated frames -> 390 milliseconds per command) gives a comfortable command elapse time for user to change their hand gesture. We finally settled with repeated frame threshold of 1 because this is much closer to the human reaction time which makes the game fun and challenging. Also, for continuous snake movement in the same direction, this setting allows the GUI to refresh at a reasonable speed (which is roughly the same as the command elapse time of 260 milliseconds) compared to the threshold of 2 which refreshes quite slow and makes the game appears laggy.

IX. References

[1] V. Vezhnevets, V. Sazonov and A. Andreeva, 'A Survey on Pixel-Based Skin Color Detection Techniques'.

[2] S. Thakur, S. Paul, A. Mondal, S. Das and A. Abraham, 'Face Detection Using Skin Tone Segmentation', *IEEE*, 2011.

[3] A. Malima, E. Ozgur and M. Cetin, 'A Fast Algorithm for Vision-based Hand Gesture Recognition for Robot Control'.

[4] Wikipedia, 'YCbCr', 2015. [Online]. Available: http://en.wikipedia.org/wiki/YCbCr.

[5] 'RGB to HSV color conversion', 2015. [Online]. Available: http://www.rapidtables.com/convert/color/rgb-to-hsv.htm.

[6] Humanbenchmark.com, 'Human Benchmark - Reaction Time Statistics', 2015. [Online]. Available: http://www.humanbenchmark.com/tests/reactiontime/statistics.

[7] Snake, 'Qt Snake', *SourceForge*, 2013. [Online]. Available: http://sourceforge.net/projects/qtsnake/.