Chapter 3

Hand Detection System Architecture

A block diagram of my hand detection system is shown in Figure 3.1. A scan window is scanned over the input image at different scales and each of the resulting image patches is fed into a classifier cascade which rapidly determines whether the image patch is a hand. The classifier cascade eliminates more than 95% of the non-hand regions in a given image. However, due to the large number of candidate regions in one image, to be practical, the false positive rate must be further reduced. To serve this need, I add a post-processor to the system, denoted by dash-line block in Figure 3.1, to further reduce false positive detections. The post-processor takes advantage of a prior knowledge of hands color and geometry. Skin detection, feature extraction, and Mahalanobis classification are the essential building blocks of the post-processor.

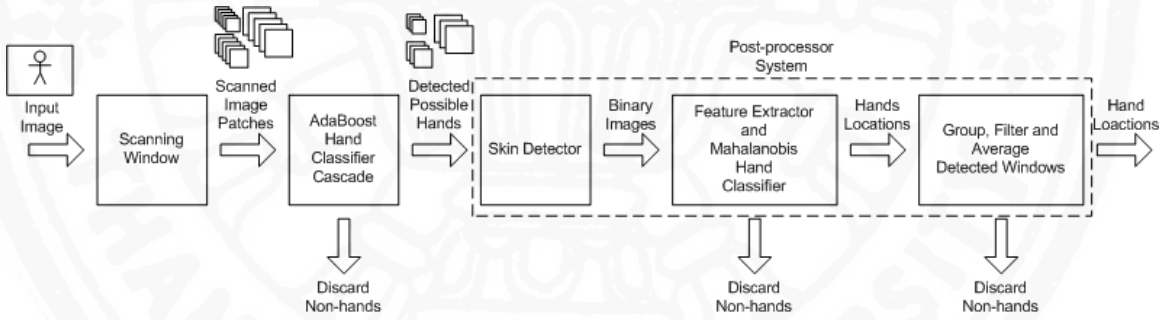


Figure 3.1: Hand detection system architecture. Scanned image pages are classified as hand or non-hand by a cascade based on Haar-like features then further filtered by a post-processor based on skin pixel blob analysis.

3.1 Scanning Window

When an image is presented to hand detection system (Figure 3.1), a detection window is scanned over the image at multiple scales, and each resulting image patch is passed to the boosted classifier cascade. The scanning process is to begin with a detection at the images original scale with the fixed scanning window size 24×24. After every possible image patch at that scale has run through the classifier cascade, the image is scaled down by 90% and the process is repeated until a minimum image size (maximum detection window size) is reached. The scanning window moves a pixel at a time in either x or y axis, while making sure that it scanned thorough out the whole image at every scale. The scanning process is shown as sudo-code in Figure 3.2.

Given : *Img*, an image
Return : *ImgSet*, a set of image patches

1. Initialize $w_i = 1/N$ where $i = 1, \ldots, N$.
2. Repeat while smallest side of the image *Img* is equal or grater than 24.
   (a) Scan 24×24 pixel window over *Img* with a step size of one pixel.
   (b) Add scanned image patches to the image set *ImgSet*.
   (c) Downscale *Img* to 90% of its current size.
3. Return the image set *ImgSet*.

Figure 3.2: Explanation of how the scanning window extracts image patches from the image.

3.2 Boosted Classifier Cascade

The research work on rapid object detection using boosted cascade of simple features [14] originally proposed the cascade of boosted classifiers as a real-time general object detector and applied it to face detection [3]. Their work showed that the system can robustly detect faces in static images independent of the background. The system runs in real-time since the feature detector is limited to a class of Haar-like filters that can be computed in constant time with the help of integral images, regardless of the spatial extent of the filters. The speed of the system is increased even further by arranging the classifiers in a cascaded fashion, so that the early stages reject most of the image patches unlikely to contain the object of interest. The cascade therefore only spends significant compute time on the image patches most likely to contain the object of interest. The illustration shown in Figure 3.3 will give a clear insight of how cascading helps faster classification.

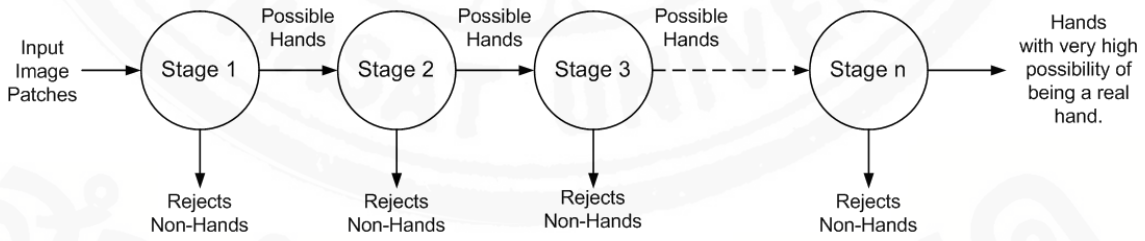
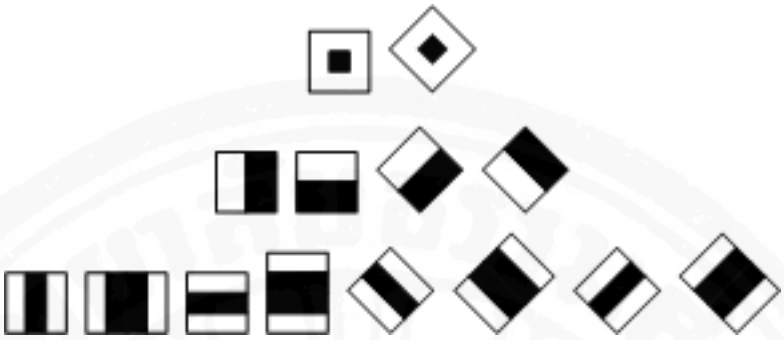


Figure 3.3: Classifiers in cascade arrangement speeds up the classification by rejecting most non-hand image patches

Each stage in the cascade is constructed from a set of simple Haar-like filters using AdaBoost algorithm [21]. AdaBoost builds a strong nonlinear classifier from multiple weak threshold classifiers, in this case each using a Haar-like filter, a threshold, and a weight, all of which are selected by AdaBoost to minimize the weighted error for the whole stage over the training set, while maintaining the desired detection rate. Original detector [14, 3] used the four types of Haar-like filters shown in Figure 3.4 (a). The filters can take on arbitrary positions and sizes within an image patch. The output of each filter is simply the difference between the

(a)



(b)

Figure 3.4: Haar-like features used to construct weak classifiers in the boosted classifier cascade. (a) Viola and Jones features. (b) Lienhart and Maydt features.

average pixel value within the clear rectangular regions and the shaded rectangular regions. They applied integral image technique to compute the value of Haar-like filters in constant time regardless of the type and size of filter.

An integral image is a special representation of image, on which rectangle Haar-like features can be computed very rapidly. The integral image is very similar to summed-area tables used in texture mapping [22] and it is one of the contributions to achieve object detector to operate in real-time. During detection, integral image for each input image is computed and then the classifier computes all required features with a few simple mathematical manipulations, only addition and subtraction.

Recently, an extended set of Haar-like features [16] are added to the original features [14, 3] for better performance. Their additional rotated Haar-like filter types are shown in Figure 3.4 (b). On a particular test set, they found that their modified system gave 10% fewer false positives than the original system for certain detection rates. The empirical analysis of detection cascade of boosted classifier [17] compared Discrete AdaBoost (the algorithm used by [14, 3]), Real AdaBoost [23], and Gentle AdaBoost[23]. The results from this comparison analysis showed that classifiers trained with Gentle AdaBoost performed the best.

Since the results from the work of [17] are very promising, I decided to used their system as a core boosted classifier cascade in my hand detection system. Due to outstanding performance over other boosting algorithms, Gentle AdaBoost is used as booting algorithm for training process with all full features set, shown in Figure 3.4 (b). The Gentle AdaBoost is very similar to AdaBoost algorithm but different in the way weights are updated. The brief idea of Gentle AdaBoost is shown in Figure 3.5 and please refer to the original work [23] for more detail.

As boosting algorithms are supervised learning algorithms, a large number of labeled positive and negative examples must be input to the training process. Positive examples manually located hands in the images and negative examples are extracted from are *background images*. The process to get positives examples and background images will be discuss in

Given : α_{tp} the desired true positive rate
α_{fp} the desired false positive rate
N examples $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ where $x \in \Re^k$ and $y \in \{-1, 1\}$
Return : A boosted classifier $H(x)$

1. Initialize $w_i = 1/N$ where $i = 1, \ldots, N$.
2. Repeat until required α_{tp} and α_{fp} is met, while increasing M by 1 at each time.
   (a) Fit the regression function $f_M(x)$ by weighted least-squares of y_i to x_i with weights w_i.
   (b) Set new weights $w_i \leftarrow w_i e^{-y_i f_M(x_i)}$ where $i = 1, \ldots, N$.
   (c) Renormalize new weights sum of all weight is equal to 1, $\sum_{i=1}^{N} w_i = 1$.
   (d) Find the α_{tp} and α_{fp} of current classifier $H(x) = sign\left[\sum_{m=1}^{M} f_m(x)\right]$ on the given N examples.
3. Output the classifier, $H(x) = sign\left[\sum_{m=1}^{M} f_m(x)\right]$.

Figure 3.5: Gentle AdaBoost algorithm used to train the classifier cascade of my hand detector system.

Chapter 4, Section 4.1. Besides those examples, two important learning parameters must be specified. Those parameters are the desired true positive and false positive rate for each stage of the cascade, and one stage is trained at a time until that stage achieves the specified true positive and false positive rates. Then a new stage is begun, and the process continues until some stopping criterion is reached. Only the positive examples that are correctly classified by the previous stages and the negative examples that are incorrectly classified by the previous stages are used to train each new stage. Figure 3.6 clearly shows how the training process train and arrange boosted classifiers in cascade order to improve detection speed.

Once we have trained the classifier cascade, image patches can be fed to the cascade for classification. Each stage in the classifier cascade classifies incoming image patch and passes to the next stage only if it is classified as a hand until the end of the cascade. The image patches that are positively classified by the last stage of cascade do have very high possibility to be true hands but high percentage of false positives are present. So, further post-processors are needed to reduce those false positives making the whole system to be practically usable.

The complete object detection system techniques mention above [14, 3, 16, 17] are implemented in Open Computer Vision Library [24]. The boosted classifier cascade training program allows users to train cascade with several choices of Haar-like feature sets and various boosting algorithms. I use this utility to construct boosted classifier cascade for my prototype hand detector system.

Given : N_{stage} the desired number of stage in the cascade
α_{tp} the desired true positive rate for each stage of cascade
α_{fp} the desired false positive rate for each stage of cascade
NP_{stage} number of positive example to train each stage of cascade
NN_{stage} number of negative example to train each stage of cascade
P_{total} a set of NP_{total} positive examples
where $NP_{total} \geq [NP_{stage} + (N_{stage} \times \alpha_{tp} \times NP_{stage})]$
B_{image} back ground images from which negative examples $N_{examples}$ will be extracted

Return : A cascade of boosted classifier $H_{cascade}$

1. Extract NP_{stage} of positive examples P_{train} from given positive example set P_{total}.
2. Extract NN_{stage} of negative examples N_{train} from background images B_{image}.
3. Repeat until desired number stage N_{stage} is exceeded in the cascade.
   (a) Train single boosted classifier H that meets desired α_{tp} and α_{fp} with positive examples P_{train} and negative examples N_{train} as described in Figure 3.5.
   (b) Add resulting boosted classifier H to the cascade $H_{cascade}$.
   (c) Test resulting boosted classifier H on training examples P_{train} and N_{train}.
   (d) Positive examples from P_{train}, misclassified by H, are replaced with new positive examples from positive example set P_{total}.
   (e) Negative examples from N_{train}, correctly classified by H, are replaced with new negative examples extracted from background images B_{image}.
4. Output the boosted classifier cascade $H_{cascade}$.

Figure 3.6: Cascading algorithm arranges boosted classifiers in cascade order to improve detection speed.

3.3 Skin Detector

There are many approaches to segmenting regions with similar color and texture from other regions. To extract skin color blobs from images, color information is the obvious choice. Nowadays, there are many skin detectors [25, 26, 27, 28, 29] available and some of those are very robust. The skin detector for my hand detection system need not be extremely robust but it should be fast.

The Bayesian maximum likelihood classifier based on color histograms [28], meets all of these needs. Based on their results and my own follow-up study, I selected the *HS* (*hue* and *saturation*) color model. The main reason to use only *H* and S while ignoring *I* (*intensity*) from *HSI* color space is to eliminate non-uniform illumination problem. Using only *HS* color model also saves computational cost and helps skin detector to work on different tone, i.e. different skin tones over Caucasian, Asian, African, and so on. Histograms used in the skin detector have two dimensions, namely hue and saturation. Each axis of the plane is quantized into 16 bins, so that each histogram will have $16\times16=256$ bins. I selected 16-bin quantization based on comparison experiments with different bins counts of 8, 16, 32, and 64. I found that 16 bins along each axis gave the best performance for my application.

I construct histograms for skin and non-skin pixels from a large training set. The normalized histogram counts are used to construct a discrete class-conditional likelihood for a Bayesian maximum likelihood classifier which I then use to determine whether a given pixel is most likely skin or not skin. To classify whether a given pixel $\overline{p}$ is skin pixel or not, H and S values are computed and quantized into 16 bins for each values. Then probability of being skin pixel $P(s|\overline{p})$ is found from skin histogram using those quantized H and S. From non-skin histogram, the probability of being non-skin pixel $P(\neg s|\overline{p})$ is found. According to the Bayesian maximum likelihood classification, pixel $\overline{p}$ is classified as skin if:

$$\frac{P(s|\overline{p})}{P(\neg s|\overline{p})} > 1.$$

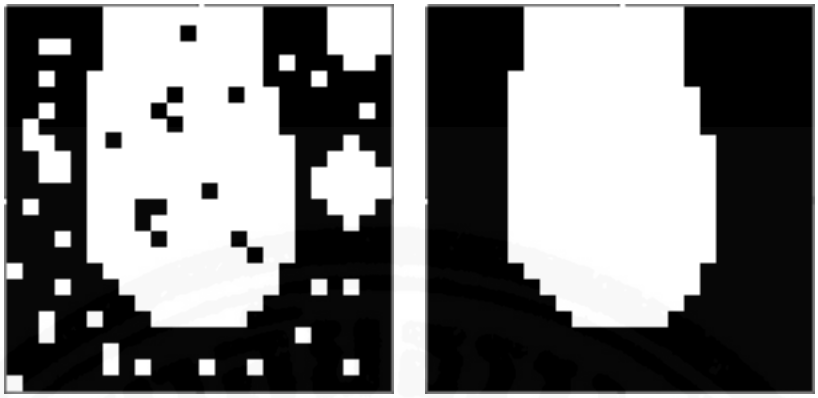
Otherwise, $\overline{p}$ is considered as non-skin pixel.

Each image patch which is classified as a hand by the cascade is scaled to a standard size 24×24 pixels and then fed to the skin detector, which produces a binary image, in which the value 1 represents a putative skin pixel and the value 0 represents a non-skin pixel.

3.4 Features Extractor

The shape and relative size of the skin blob within the detection window give useful information for discriminating image patches containing hand from those not containing hand. There are a many popular and classical shape descriptors like shape signatures, Fourier descriptors and curvatures [30]. Even more powerful shape analysis and classification technique called shape context [31] is available. But those techniques demand high computational cost and are not suitable for classification pretty noisy shape in a small 24x24 pixels binary image. I did research to find the efficient features best suited to my application.

Finally, I got four simple features which can be used to discriminate between hand image patches and on-hand image patches. The binary images produced by skin detector may be



(a) (b)

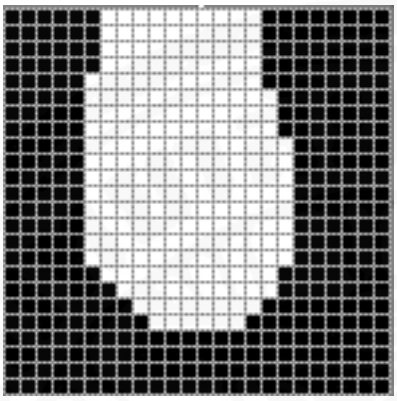
Figure 3.7: (a) Ideal skin detected image of proper hand, which may be produced by skin detector. (b)Largest connected skin blob of image (a), on which holes are filled. (Images are synthesized to model problem.)

noisy and probably contain multiple connect skin blobs. So, the largest connected blob is selected and fill the holes on it. The ideal skin detected image of proper hand is is shown in Figure 3.7 (a) and image (b) shows the hole filled largest connected skin blob. Then, feature extractor computes four simple features from the largest skin blob that are surprisingly useful for accurate classification:

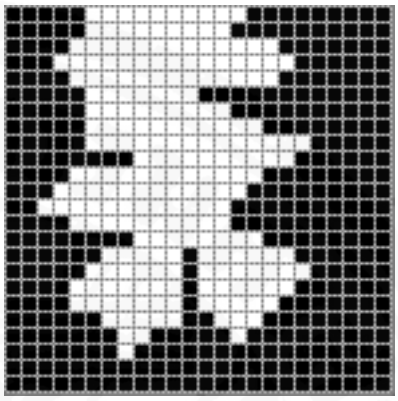
1. The area of the blob f_{area}.
2. The length of the perimeter of blob $f_{perimeter}$.
3. The eccentricity of the blob $f_{eccentricity}$.
4. The number of pixels on the boundary of blob that intersects the detection window boundary $f_{boundary}$.

The area feature f_{area} is simply the number of pixels in the largest connected skin component; it is normalized by the total number of skin pixels $24 \times 24 = 576$ in the image patch. It is very obvious that the given image patch is unlikely to contain a hand if the area feature is very large or very small. As shown in Figure 3.8, image patch containing very large (c) or small (d) can be easily discriminate from the image patch containing real hand (a) based only area feature. But (b) can not be discriminated from (a) only using area feature since the area value for both images are the same. Fortunately, there are other features which are able to handle this kind of problem.

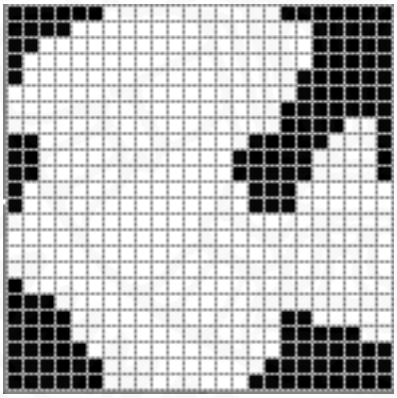
The perimeter feature $f_{perimeter}$ is the total number of pixels on the perimeter of the largest connected skin component; it is normalized in the same way as the area feature. I selected perimeter feature based on the fact that although area and other region properties of two shape may be the same, the perimeter can be different. Figure 3.8 (a) and (b) clearly demonstrate this idea. Both images are having slightly same area, eccentricity and boundary but perimeter feature helps to eliminate image containing non-hand (b) from the positive image patch (a). However, this feature is usable only for detecting hand in low-resolution condition where detail of hand is not visible. If detail of hand is visible the range of perimeter for



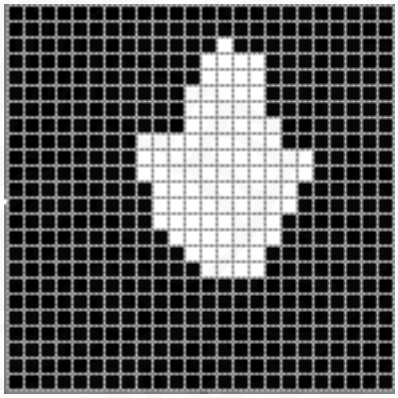
(a) Ideal Hand
f_{area} - 226
$f_{perimeter}$ - 62
$f_{eccentricity}$ - 0.7783
$f_{boundary}$ - 10



(b) Hand-like skin color blob
f_{area} - 226
$f_{perimeter}$ - 120
$f_{eccentricity}$ - 0.7755
$f_{boundary}$ - 10



(c) Non-hand very large skin blob
f_{area} - 424
$f_{perimeter}$ - 122
$f_{eccentricity}$ - 0.4885
$f_{boundary}$ - 37



(d) Non-hand small skin blob
f_{area} - 98
$f_{perimeter}$ - 47
$f_{eccentricity}$ - 0.7357
$f_{boundary}$ - 0

Figure 3.8: Illustration to demonstrate how useful four features are in the discrimination between image patches contain hand and those do not contain proper hand. Here, feature values for each image are absolute values, i.e. they are not normalized. (Images are synthesized to model problem.)

hand will be very large due to unlimited deformations and poses of hand in high-resolution imagery.

The eccentricity feature $f_{eccentricity}$ is basically the eccentricity of the ellipse having the same second moments as the largest connected skin component, i.e., the ratio of the distance between the foci of the ellipse and its major axis length. Eccentricity is computed with technique used by function *regionprops* in Image Processing Toolbox of MATLAB 6.5 [32]. We know x_i and y_i, locations of $x-y$ location of n skin pixels, from the skin detected binary image. Then, the second order moments relative to x axis μ_{xx}, y axis μ_{yy} and both axis μ_{xy} can computed as

$$\mu_{xx} = \sum_{i=1}^{n} \frac{(x_i - \overline{x})^2}{n} + \frac{1}{12}$$

$$\mu_{yy} = \sum_{i=1}^{n} \frac{(y_i - \overline{y})^2}{n} + \frac{1}{12}$$

$$\mu_{xy} = \sum_{i=1}^{n} \frac{(x_i - \overline{x}) \times (y_i - \overline{y})}{n}$$

where $\overline{x}$ is the mean value of x_i

$$\overline{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

and $\overline{y}$ is the mean value of y_i

$$\overline{y} = \sum_{i=1}^{n} \frac{y_i}{n}.$$

From those second order moments, the eccentricity is computed by using following mathematical formula.

$$f_{eccentricity} = \sqrt{\frac{2\sqrt{(\mu_{xx} - \mu_{yy})^2 + (4\mu_{xy}^2)}}{\mu_{xx} + \mu_{yy} + \sqrt{(\mu_{xx} - \mu_{yy})^2 + (4\mu_{xy}^2)}}}$$

The eccentricity value range is between 0 and 1, with 0 indicating a circle and 1 indicating a line segment. This feature helps to discriminate face skin regions and other round shape skin colored objects, which tend to be quite round, from true hand skin regions, which tend to be more eccentric.

Finally, the boundary feature $f_{boundary}$ helps to discriminate between arm skin regions, which tend to intersect the boundary of the detection window in two places, from true hand skin regions, which only intersect the detection window at the wrist. The boundary feature value is normalized by the total number of pixel on the perimeter of the detection window, $(24\times2)+[(24-2)\times2]=92$ in our case.The boundary feature provides information about how wrist-like the boundary is. Boundary feature also helps eliminate negative image patches containing skin color blob that intersects with detection window at several places or does not intersect at all. This can be seen clearly in Figure 3.8 (a), (c) and (d). Since skin-color blob in (c) intersects with detection windows at several places its boundary value is very much larger than that of image patch containing real hand (a). Moreover, boundary value of negative image patch (d) is zero, very much lesser than that of positive image patch (a) because skin-color blob in image patch (d) does not intersect with detection window boundary.

3.5 Mahalanobis Classifier

No matter how good those four features are, they will not be efficiently utilized for classification without a suitable classifier. There are many classification techniques available ranging from very simple statistical Gaussian classifier to complex state of art classifiers such as support vector machine (SVM). Among those, classical artificial neural networks are very popular choices.

I prefer classifiers that are simple with few parameters to tune and training process of the classifier to be simple. The computation cost of should be low so that there will be some possibility that hand tracking system or higher level applications based on my hand detector will be able to run in real-time. I found that a simple classifier based on Mahalanobis distance is a reasonable choice. The advantage of using Mahalanobis distance over Euclidean distance is that it take account the patterns of covariance that exist in the data during computation [33].

Each image patch can be represented by a feature vector consisting of the area, perimeter, eccentricity, and boundary features. To classify a given feature vector $\vec{f}$ as a true hand or not a hand, we calculate the Mahalanobis distance

$$d_{Mahalanobis} = (\vec{f} - \vec{\mu})^T \Sigma^{-1} (\vec{f} - \vec{\mu})$$

between the feature vector $\vec{f}$ and the mean feature vector μ, then we classify $\vec{f}$ as a hand if $d_{Mahalanobis}$ is less than some threshold θ. Here the mean hand feature vector $\vec{\mu}$, the covariance matrix Σ and they are computed from the four features of given sample positive examples matrix:

$$f_{sample} = \begin{bmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \vdots \\ \vec{f}_n \end{bmatrix} = \begin{bmatrix} f_{area_1} & f_{perimeter_1} & f_{eccentricity_1} & f_{boundary_1} \\ f_{area_2} & f_{perimeter_2} & f_{eccentricity_2} & f_{boundary_1} \\ \vdots & \vdots & \vdots & \vdots \\ f_{area_n} & f_{perimeter_n} & f_{eccentricity_n} & f_{boundary_n} \end{bmatrix}.$$

So, f_{sample} is n×4 matrix, where n is the total number of *sample positive image patches*. The procedure to get *sample positive image patches* will be described in Chapter 4, Subsection 4.2.3. The mean hand feature vector is simple computed from f_{sample} as:

$$\vec{\mu} = \begin{bmatrix} \mu_{area} & \mu_{perimeter} & \mu_{eccentricity} & \mu_{boundary} \end{bmatrix} = \frac{1}{n} \sum_{i=1}^{n} \vec{f}_i.$$

Then, covariance matrix Σ is computed by using following formula:

$$\Sigma = \frac{\chi^T \times \chi}{n}$$

where χ is simply computed by subtracting $\vec{\mu}$ from each row of the matrix f_{sample}:

$$\chi = \begin{bmatrix} \vec{f}_1 - \vec{\mu} \\ \vec{f}_2 - \vec{\mu} \\ \vdots \\ \vec{f}_n - \vec{\mu} \end{bmatrix}$$

and dimensions is same as dimensions of matrix f_{sample}. The resulting covariance matrix σ is 4×4 matrix. The distance threshold θ are estimated from the training set and will be discussed more in Chapter 4, Subsection 4.2.3.

Once classification for each possible detection windows is done, the positively detected hands are fed to the final module, the grouping, filtering, and averaging module. Further reduction of false positives is done there.

3.6 Grouping, Filtering, and Averaging Module

The Mahalanobis classifier produces a few very sparsely distributed false positives and densely distributed true detections around the actual targets. Since it produces several true detections around each of the actual detections, grouping and averaging is necessary to ensure only one detection for each target. I use the existing implementation of this technique in the OpenCV.

Grouping process groups the nearby detections together. Given square shape detection D_1 and D_2 are taken to be in the same group if the following requirements are met.

1. D_{2x} is in the range of $[D_{1x} \pm (D_1 w \times 0.2)]$,
2. D_{2y} is in the range of $[D_{1y} \pm (D_1 w \times 0.2)]$,
3. $D_{2w} \leq (D_{1w} \times 1.2)$, and
4. $D_{1w} \leq (D_{2w} \times 1.2)$.

Where D_{1x} and D_{1y} is the x, y location of D_1
D_{2x} and D_{2y} is the x, y location of D_2
D_{1w} is the width of D_1
D_{2w} is the width of D_2.

Once grouping is done, a group which contains less than some number of detections can be disposed of on the assumption it is a false positive. Then the averaging process takes the average of detection windows in each remaining group resulting only one detection window for each group. The positively detected hands output from this module could then be forwarded to another component in an integrated application, for example a gesture recognition module. But, in this thesis, I simply evaluate the performance and efficiency of the proposed algorithm on a series of video sequences. The following chapter describes my experiments in detail.