CHAPTER IV

Facial Feature Detection

The goal of facial feature detection is to detect the presence and location of features (left eye, right eye, nose, and mouth). Before applying facial feature detection algorithms, the size of the face from the face detection process is normalized to 128×128 pixels. The proposed approach consists of three main steps. First, facial features are coarsely detected by a neural visual model (NVM). To be free from the usage of intensity information, the inputs of the NVM are defined from the position and face shape information. The results from the NVM are further enhanced by applying image dilation algorithm in the second step. In the last step, Radon transform is used to evaluate the face angle of a rotated image.

4.1 Neural Visual Model (NVM)

Neural visual model (NVM) is used for detecting facial features. The model consists of three neural networks for detecting facial features in frontal, left-profile, and right-profile views of faces. For each view, there are two parts. The first part is input preparation. The network inputs are facial parameters obtained from position and face shape information. The other part is model construction.

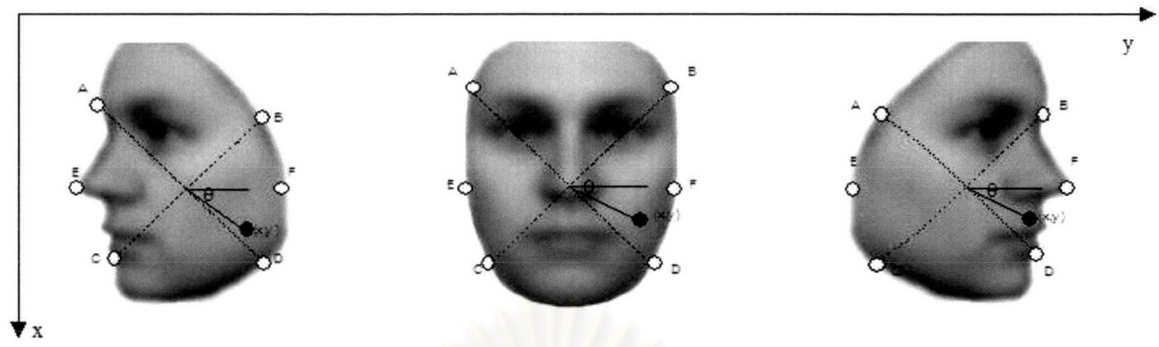

Figure 4.1: Notation used for the facial parameters. O is a face center. Line EF is parallel to the Y-axis. The length of line EF is called face width. Lines AD and BC are perpendicular to each other. The angles $B\hat{O}D$ and $A\hat{O}C$ are divided into two halves by line EF. A, B, C, and D are at four corners of the face.

4.1.1 Facial Parameters

Seven facial parameters obtained from face databases are extracted from the position and face shape information. The first two parameters denote the position of any given image pixel with respect to the center of face image in polar coordinates. The remaining five parameters represent the width of the face and the Euclidean distances measured from the four corners, i.e., upper left, upper right, lower left, and lower right, to the center of face image, respectively. The parameters are illustrated in Figure 4.1. The normalized size of the training image is 128×128 pixels.

The following geometric values and notations are referred in the NVM.

1. (x_o, y_o) : Coordinate of the center point O. x_o is measured in terms of the number of rows and y_o is measured in terms of the number of columns. The values of x_o and y_o are dependent on the length and width of the image, respectively. Since the length of a given image has no effect on the analysis of facial features by NVM, the value of x_o is set to $l/2$. However, the value of y_o depends upon the actual width of a given face and must be derived from the image. Discussion of this derivation

will be given later.

2. (r, θ) : Polar coordinate at any point (x, y) with respect to the center point calculated by

$$r = \sqrt{(x - x_o^2) + (y - y_o^2)} \tag{4.1}$$

and

$$\theta = arctan\frac{y - y_o}{x - x_o} \tag{4.2}$$

3. L_w : Face width defined as the distance of a line connecting two boundary points (E and F) of the face. Line EF is parallel to the Y-axis. The actual value of L_w depends on the actual width of a given face.

4. L_A, L_B, L_C, L_D : Distance between the center (O) and the far upper left(A), upper right(B), lower left(C), and lower right corner(D), respectively. These values depend upon the actual region of the face. Discussion of how to compute these values will be given later.

The values of L_w and y_o can be derived from a given image by the following algorithm.

Algorithm 2: Finding Face Width

1. Extract face by the face detection algorithm and normalize it to 128 × 128 pixels.

2. Convert the face image from step 1 into edge face image.

3. Generate face template of size 128 × 128 from resizing face template of size 24 × 24 and dilate the face boundary of the face template.

4. Apply *AND* operation to the dilated template and the edge face image.

5. Dilate the face boundary from step 4 in order to connect all pixels.

6. Apply thinning method to dilated face boundary.

7. Consider two pixels at the 64^{th} row (because the face is length-fitted with normalized image of size 128×128 and the face width is measured from two pixels at the half of thinned image length), the first column from the left side of the face image, y_l, having a white pixel is called *left boundary point* and the first column from the right side of the face image, y_r, having a white pixel is called *right boundary point.* The values of L_w and y_o can be computed by the following equations.

$$L_w = y_r - y_l \tag{4.3}$$

and

$$y_o = y_l + \lceil 0.5 \times L_w \rceil \tag{4.4}$$

An example of locating the face center point and computing the face width is shown in Figure 4.2.

Lines AD and BC are perpendicular to each other. The angles $B\widehat{O}D$ and $A\widehat{O}C$ are divided into two halves by line EF. We can find L_A, L_B, L_C, L_D by counting the number of pixels in four directions (upper left, upper right, lower left, and lower right) from the center point until a white pixel is found for each direction in the thinned image (step 6 of face width algorithm).

These facial parameters are independent of the facial color, light condition, and any face occlusions such as wearing the spectacles, sunglasses, and covering the mouth by a scarf. Participating parameters in locating essential facial features are r, θ, L_w, L_A, L_B, L_C, L_D.

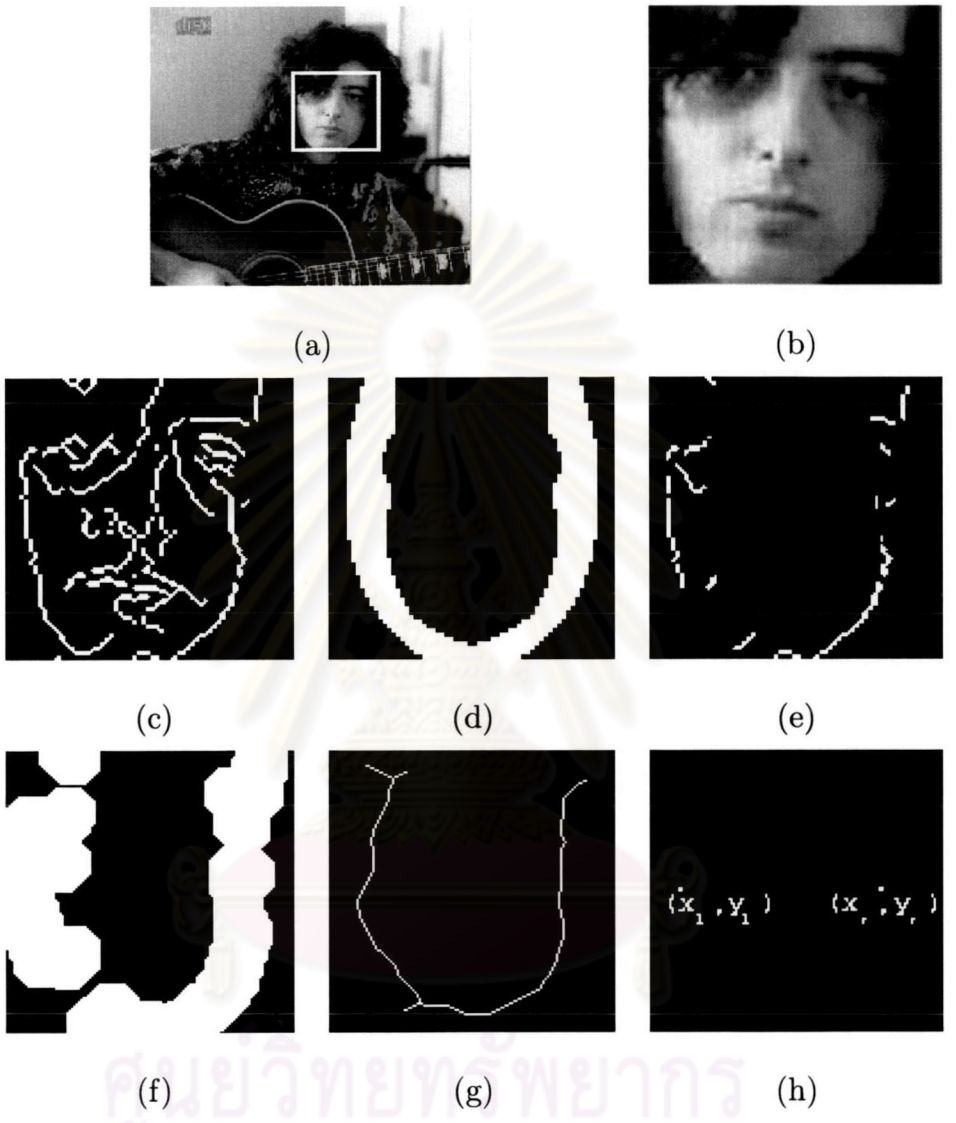

Figure 4.2: Steps for finding face width and face center point. (a) Face detection. (b) Normalized face from (a). (c) Edge version of (b). (d) Dilated boundary from face template. (e) Result of applying *AND* operation to (c) and (d). (f) Dilated version of (e). (g) Thinned version of (f). (h) Two points at the 64^{th} row. The face width is the distance between these points and the face center is at the center of the line between these points.

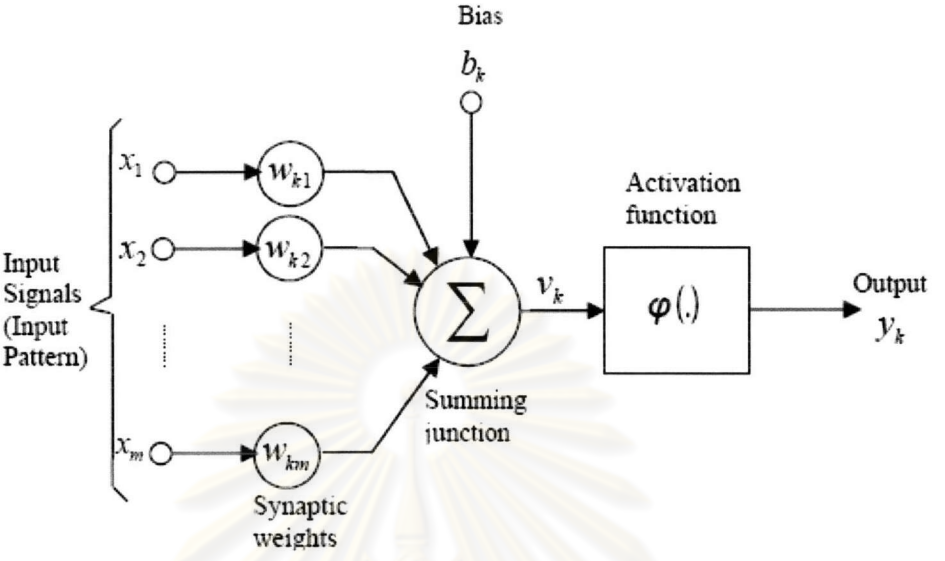

Figure 4.3: Nonlinear model of a node.

4.1.2 The Feedforward Neural Networks

A node(neuron) is an information processing unit that is fundamental to the operation of a neural network. The block diagram of Figure 4.3 shows the model of a node which forms the basis for designing neural networks. In mathematical terms, a node k is described by the following pair of equations.

$$u_k = \sum_{j=1}^{m} w_{kj} x_i \tag{4.5}$$

and

$$y_k = \psi(u_k + b_k) \tag{4.6}$$

where

$x_1, x_2, \ldots, x_m$	:	input signals
$w_{k1}, w_{k2}, \ldots, w_{km}$	:	synaptic weights of node k
u_k	:	linear combiner output of node k due to the input signals
b_k	:	bias of node k
$\psi(\cdot)$	:	activation function
y_k	:	output signal of node k

The use of bias b_k has the effect of applying an affine transform to the output u_k of the linear combiner in the model of Figure 4.3, as shown by

$$v_k = u_k + b_k \tag{4.7}$$

A popular activation function is sigmoidal activation function shown in Eq.(4.8).

$$\psi(x) = \frac{1}{1 + exp(-ax)} \tag{4.8}$$

A feedforward neural network is a form of interconnected nodes without having a feedback loop. There are two different classes of network architectures [58]:

1. Single-layer feedforward networks: In the simplest form of a layered network, an input layer of source nodes that projects onto output layer of computaional nodes, but not vice versa.

2. Multilayer feedforward networks: This class of a feedforward neural network distinguishes itself by the presence of one or more hidden layers of computaional node.

4.1.3 Implementation of the Neural Visual Model

Typically, a multilayer feedforward network (Figure 4.4) consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation

nodes, and an output layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptron (MLP) network [58].

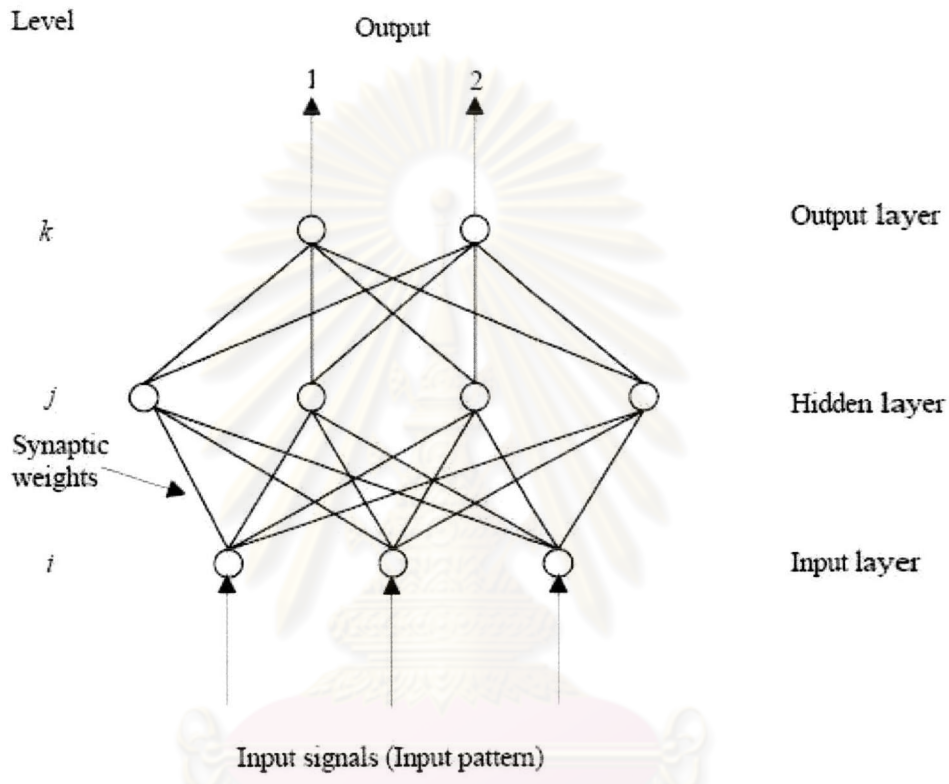

Figure 4.4: Architecture of multi-layer perceptron.

Facial feature detection model is constructed using MLP networks with back-propagation (BP) learning algorithm. The model consists of three neural visual networks for frontal, left-profile, and right-profile views. Four essential facial features, i.e., left eye, right eye, nose, and mouth, are considered. Each feature will be located using one MLP network. Hence, four MLP networks must be introduced for all features in each neural visual network. The problem of locating facial feature is considered as a problem of classifying a given set of facial parameters into two classes, interested facial feature class and uninterested facial feature class.

BP learning [58] consists of two phases, namely, learning phase and operating phase. In learning phase, a set of input patterns (training set) is presented to the input layer together with their corresponding desired output patterns. Each input pattern consists of seven facial parameters as described in the previous section. In this study, there are two classes of desired output patterns. The first class corresponds to the input pattern embodied in a facial feature under focus. The Other class corresponds to the input pattern not embodied in a facial feature under focus. A small random initial weight value is assigned to each connection between nodes in the input layer and the hidden layer because it is gradually adjusted to obtain the appropriate weight value. As each input pattern is applied, the actual output is recorded. The weights between the output layer and the hidden layer are re-calculated using the delta rule. The weight adjustment reduces the difference between the network actual outputs and the desired outputs for a given input pattern.

BP learning errors are resulted from two passes through different layers of network [58], namely,

1. Forward pass, an activity pattern that is applied to the input node of the network. Its effect propagates through the network layer by layer and produces a set of outputs as the actual response of the network.

2. Backward pass, an error signal that is produced by subtracting the actual response of the network from the desired response. It then propagates backward through the network. The synaptic weights, which are all fixed in the forward pass, are adjusted to minimize the difference between actual and desired outputs of the network.

The input to each node for successive layers is the sum of the scalar products of the incoming vector components with their respective weights. Thus the input to a node j

(Figure 4.4) is given by

$$input_j = \sum_i w_{ji} \cdot out_i \tag{4.9}$$

where

w_{ji} : weight connecting node i to node j

out_i : output node i.

No calculation is performed at the input layer since this stage merely feeds input patterns to the network. For $input_j$ x, the output of a node j is

$$out_j = \psi(x) \tag{4.10}$$

and this output is sent to all nodes in the hidden layer. This computation is continued until the output layer is reached. At that point, the output vector is generated.
The function $\psi(\cdot)$ is the activation function of each node. In this study, the sigmoidal activation function is employed. Based on the difference error term or δ term in the output layer, the weight can be computed by [58]

$$w_{kj}(n+1) = w_{kj}(n) + \eta(\delta_k out_k) \tag{4.11}$$

where $w_{kj}(n+1)$ and $w_{kj}(n)$ are the weights connecting nodes k and j at iteration $(n+1)$ and n, respectively, and η is a learning rate parameter. The δ term of the nodes in hidden layer are computed and the weights connecting the hidden layer to the previous layer (another hidden layer or input layer) are updated accordingly by

$$w_{ji}(n+1) = w_{ji}(n) + \eta(\delta_j out_j) \tag{4.12}$$

where $w_{ji}(n+1)$ and $w_{ji}(n)$ are the weights connecting nodes j and i at iteration $(n+1)$ and n, respectively. These calculations are repeated until all weights have been adjusted to decrease the difference error.

The δ term in previous equation is often referred to as the rate of change of error with respect to the input of node k, and can be written as

$$\delta_k = (d_k - out_k)\psi'(input_k) \tag{4.13}$$

for nodes in the output layer, and

$$\delta_j = \psi'(input_k) \sum_k \delta_k w_{kj} \tag{4.14}$$

for nodes in the hidden layers, where d_k is the desired output of node k. The fundamental of BP learning algorithm employed in training phase is a gradient descent optimization procedure which minimizes the mean squared error between network output and the desired output of all input patterns P as follows:

$$E_{msq} = \frac{1}{2P} \sum_p \sum_k (d_k - out_k)^2 \tag{4.15}$$

The training set is presented iteratively to operate the network, whereby the weights are updated until their values become stabilized according to the following criteria:

- a user-defined error tolerance is achieved, or
- a maximum number of iterations is reached.

4.1.4 Applying Cross Validation to Stopping Criteria

The important goal in the learning phase of the neural network is to obtain an optimal generalization performance. Generalization performance means small errors on examples not seen during training [60–62]. Because standard neural network architectures such as the fully connected multilayer perceptron (MLP) network almost always have too large a parameter space, such architectures are prone to overfitting. While the network seems to get better and better, i.e., the error on the training set decreases, at some point during

training the error actually begins to get worse again, i.e., the error on unseen examples increases. There are two methods to solve this situation [60–62]. The first method is to reduce the number of dimensions of the parameter space or the effective size of each dimension. Details on the first method can be found in [63–65]. The other method is early stopping. Early stopping is widely used because it is simple to understand and implement. It can be used either interactively or automatically, i.e., based on human judgement or automatic stopping criteria.

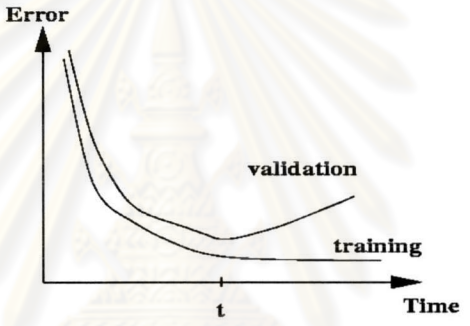

Figure 4.5: Idealized training and generalization error curves.

Figure 4.5 shows the evolution overtime of the per-example error on a training set and on a test set not used in training. The following steps explain how to perform early stopping using cross validation [60–62].

Algorithm 3: Cross Validation Training

1. Split the training data into a training set and a cross validation set.

2. Train only on the training set and evaluate the per-example error on the validation set once in a while.

3. Stop training as soon as the error on the cross validation set is higher than it was checked the last time.

4. Use the weights in the previous step as the result of the training run.

This method uses the cross validation set to simulate the behavior on the test set, assuming that the errors on both sets will be similar.

There are a number of plausible proposed stopping criteria where Prechelt [60] classified them into three classes based on the following definitions. Given

- E be the objective function (error function) of the training algorithm,
- $E_{tr}(t)$ be the average error per example over the training set, measured after epoch t,
- $E_{va}(t)$ be the corresponding error on the validation set and is used by the stopping criterion,
- $E_{te}(t)$ be the corresponding error on the test set, and
- $E_{opt}(t)$ be the lowest validation set error obtained in epochs up to t,that is, $E_{opt}(t) = \min_{t' \leq t} E_{va}(t')$

The generalization loss at epoch t is the relative percentage increase of validation error over the minimum-so-far (in percent) [60]:

$$GL(t) = 100 \times \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right) \tag{4.16}$$

A high generalization loss directly indicates overfitting, hence the stopping criteria of training. Consequently, the first class of stopping criteria (GL_ς) is to stop as soon as the generalization loss exceeds a certain threshold [60], that is,

GL_ς: stop after first epoch t with $GL(t) > \varsigma$

As training error still decreases quickly, no overfitting takes place. When the error decrease rate begins to level, overfitting will likely occur. A training strip of length k is defined as a sequence of k epochs numbered $n+1, ..., n+k$, where n is divisible by k. The training progress (in per thousand) measured after such a training strip is [60]

$$TP_k = 1000 \times \left(\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k \times \min_{t'=t-k+1}^{t} E_{tr(t')}} - 1 \right) \tag{4.17}$$

The above measures the average training error for the duration when the strip becomes larger than the minimum strip.

The second class of stopping criteria is defined as the quotient of generalization loss and progress as follows [60]:

$$PQ_\varsigma\text{: stop after first end-of-strip epoch } t \text{ with } \frac{GL(t)}{TP_k(t)} > \varsigma$$

Cross validation errors are measured only at the end of each strip, depending on the sign of changes in the generalization.

The third class of stopping criteria is based on the notion of stopping when generalization errors increase in successive steps [60], that is,

$$UP_s\text{: stop after epoch } t \text{ if } UP_{s-1} \text{ stops after epoch } t-k \text{ and } E_{va}(t) > E_{va}(t-k)$$

$$UP_1\text{: stop after first end-of-strip epoch } t \text{ with } E_{va}(t) > E_{va}(t-k)$$

This definition means that when validation error increases not only once, but during s consecutive strips and assumes that such increase indicates the beginning of final overfitting, independent of how large the increases actually are.

From all stopping criteria classes (GL, PQ, and UP), the classes GL and PQ are employed as the stopping criteria of the training under back-propagation learning approach because model construction applying the classes GL and PQ takes less time than applying UP although the results of applying all stopping criteria classes are similar.

If the error on validation set is not consistent with the error on test set, the data in training set, validation set ,and test set must be rearranged so that the errors on validation set and test set are similar. The other choice is adding inputs in order to give more information for training.

4.2 Image Dilation

The field of mathematical morphology [67] contributes a wide range of operators to image processing, all based around a few simple mathematical concepts from set theory. The operators are particularly useful for the analysis of binary images and common usages including edge detection, noise removal, image enhancement, and image segmentation. The most basic operation in mathematical morphology is dilation. This operator takes two pieces of data as input: an image to be dilated and a kernel. For a binary image, white pixels are normally taken to represent foreground regions while black pixels denote background. The set of coordinates corresponding to that image is simply the set of two-dimensional Euclidean coordinates of all the foreground pixels in the image, having an origin normally taken in one of the corners so that all coordinates have positive elements. The kernel is just a set of point coordinates although it is often represented as a binary image. The kernel differs from the input image coordinate set in that it is normally much smaller, and its coordinate origin is often not in a corner. Thus, some coordinate elements will have negative values. Note that in many implementations of morphological operators, i.e., dilation and erosion, the kernel is a particular shape (e.g. a 3×3 square) and so is hardwired into the algorithm.

The basic effect of dilation on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller. The mathematical

definition of dilation for binary images is as follows:

Suppose that X is the set of Euclidean coordinates corresponding to the input binary image, and that K is the set of coordinates for the kernel. Let Kx denote the translation of K so that its origin is at x. Then the dilation of X by K is simply the set of all points x such that the intersection of Kx with X is non-empty.

In this dissertation, image dilation [67] is chosen to eliminate some irrelevant regions remaining after using NVM. The algorithm for eliminating the irrelevant regions [61,62, 66] is described as follows:

Algorithm 4: Eliminating Irrelevant Regions

1. Filter the original image by a threshold, and invert the threshold filtered image. The black-and-white face image is obtained from this step.

2. Apply dilation to the image from step 1 by

$$I_{Dilated}(x, y) = I_{BW}(x, y) \oplus K(x, y) \tag{4.18}$$

where

$\oplus$	:	dilation sign
(x, y)	:	position of pixel in rectangular coordinate
$I_{BW}(x, y)$	:	black-and-white face image from step 1
$K(x, y)$	:	kernel defined by the 3×3 square image with white intensity
$I_{Dilated}(x, y)$	:	dilated image

3. Combine the dilated image and the detected image from the NVM with an *AND* operation by

$$I_{Detected}(x, y) = \text{AND}(I_{Dilated}(x, y), I_{NVM}(x, y)) \tag{4.19}$$

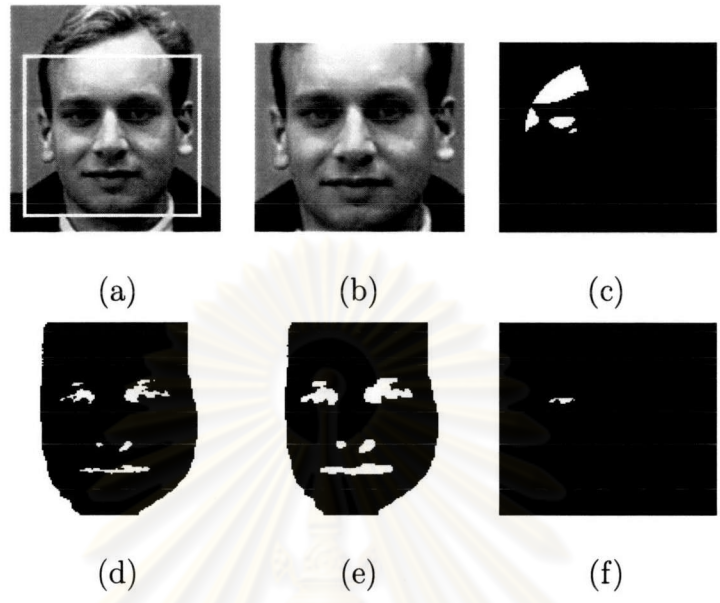

Figure 4.6: Left-eye region detection steps using image dilation. (a) Face detection. (b) Normalized face from (a). (c) Left-eye region detection from NVM. (d) Black-and-white image obtained from original image with thresholding filter (step 1). (e) Dilated image. (f) Left-eye extraction from detected region using the image dilation technique.

where

$I_{NVM}(x, y)$: detected image from NVM

$I_{Detected}(x, y)$: detected image after eliminating the irrelevant regions

An example of how the algorithm works is shown in Figure 4.6.

4.3 Rotational Invariance

Radon transform [68] transforms a 2-dimensional image with lines into a domain of possible line parameters. Each line in the image gives a peak position at the corresponding line parameters such as the distance to the center of image (ρ) and the angle of a line (θ). This leads to many line detection applications within image processing and computer vision. For algorithm, some lines of rotated edge image provide an important line

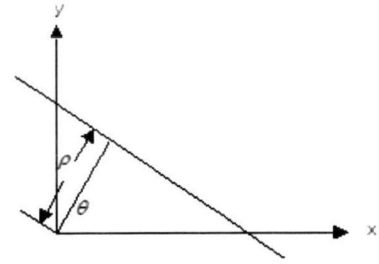

Figure 4.7: Line expression.

parameter called *face angle*. The lines in an edge face image can be expressed in the form:

$$\rho = xcos(\theta) + ysin(\theta) \tag{4.20}$$

where θ is the angle of a line and ρ is the smallest distance to the center of the face image. An example of line expression is shown in Figure 4.7. Radon transform of a set of parameters is the line summation through the edge image $I_{edge}(x, y)$, where the line is positioned correspondingly to the value of (ρ, θ) and δ is the Kronecker delta as follows:

$$\widetilde{I_{edge}}(\rho, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I_{edge}(x, y)\delta(\rho - x\cos(\theta) - y\sin(\theta))dxdy \tag{4.21}$$

In the face detection stage, a face is detected by face templates with eight angles (Figures 3.5 and 3.6). However, the real face angle is re-calculated from the maximum intensity of the transformed image by Eq.(4.21). The algorithm for facial feature detection with rotational invariance is as follows:

Algorithm 5: Rotational Invariant Facial Feature Detection

1. Construct the edge image using Canny edge detector.

2. Eliminate the face boundary by removing outside white pixels from possible face regions. A face region is in the circle region with radius α and center at face center (x_o, y_o). The radius is calculated by

$$\alpha = 0.3 \times \max(m, n) \tag{4.22}$$

where $m \times n$ is image size. The coefficient 0.3 in the above equation is a yield value which retains the eyes, nose, and mouth regions but face boundary. This coefficient is statistically tuned for good implemention.

3. Apply Radon transform to the edge image from the previous step. The face angle, β, is evaluated from the positions having the highest intensity on the transformed image.

4. Rotate the original image by $-\beta^\circ$ in clockwise direction using a bilinear interpolation.

5. Detect face and facial features from the re-rotated image using face and facial feature detection algorithms in Sections 2 and 3.

6. Bound the regions of facial features by rectangular blocks.

7. Inversely rotate the rectangular blocks by (β°) in clockwise direction and superimpose the blocks on the original image.

An example of rotated face and facial feature detection is shown in Figure 4.8.

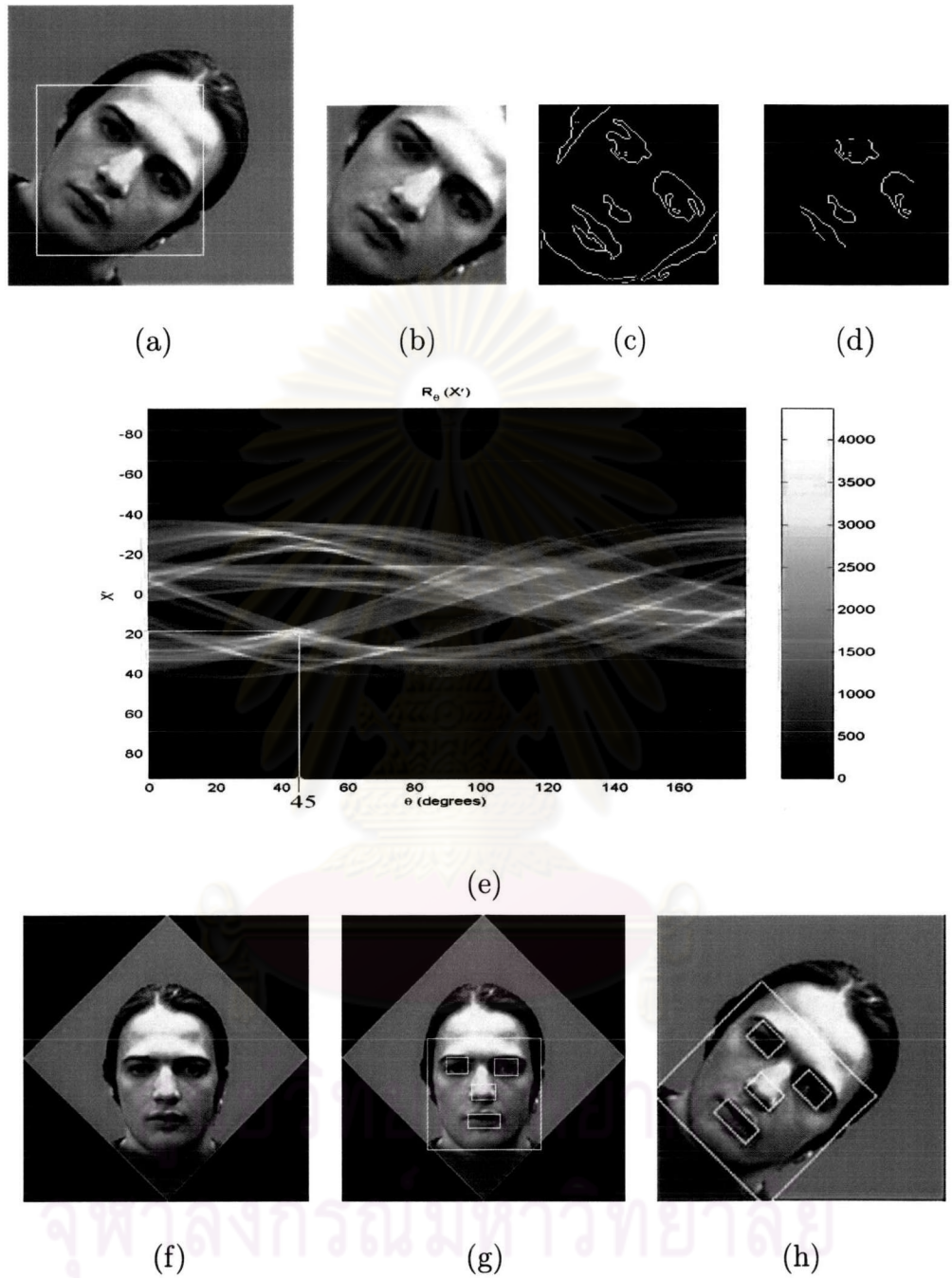

Figure 4.8: Detection steps of rotated image. (a) Face detection. (b) Detected face. (c) Edge image. (d) Edge image eliminating face boundary. (e) Radon transform of (d) which the highest intensity is at 45°. (f) Re-rotated image obtained by rotating 45° in a counter-clockwise direction. (g) Face and facial feature detection of re-rotated image. (h) Detected image.