2001

Algorithm design for smart vision sensors.

Hongmei. Gao
University of Windsor

ALGORITHM DESIGN FOR SMART VISION SENSORS

by

Hongmei Gao

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the Department of Electrical & Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2001

Abstract

Design of smart vision sensors attracts a lot of academic and industrial interests, as solutions to many existing production problems require the inclusion of intelligent sensors into the manufacturing processes. The goal of this thesis is to provide algorithm design for smart vision sensors.

Two algorithms are developed in this thesis. The first one is based on the correlation analysis method for images and can be implemented to find the 2-dimensional position of a target. In particular, the problem of finding the deviations along both X and Y directions is formulated as a matching process between the saved template, which represents the reference position, and the picture of a real static target captured by a 'vision' element such as a *CCD* camera, through correlation analysis of the 2-D spatial shift. It is an efficient and simple method for deviation identification of target position with high noise rejection ability. Using this method, 2-D position deviation can be found very accurately with high reliability.

The second algorithm, which is based on locating the critical points of a planar shape, can be applied to identifying the pattern and finding the 2-dimensional position deviation and rotation angle of a target. The problems of finding the deviations along both horizontal (X direction) and vertical (Y direction) directions and the rotation angle are formulated as a process of locating the critical points of the planar shape. And the shape recognition can be achieved by comparing the shape numbers between the saved template image and the image of a real target. It is an efficient, accurate and effective method.

It is noted that the size of the captured image by the vision element can be reduced in our design in order to accommodate fast real-time application with reasonable accuracy and reliability.

It is also pointed out that the procedures for both methods can be applied to the design of

a smart vision sensor. Also, because we mainly use software solution for the design, it is very flexible to update the sensor functions and to augment this sensor into a closed-loop architecture design.

The examples show that our design idea exhibits good performance and can be applied to the design of a real high-performance and cost-saving smart vision sensor.

Acknowledgments

My sincere appreciation goes to my supervisor Dr. Xiang Chen for his guidance, support and encouragement throughout the course of this research.

I would also like to thank Dr. James Soltis and Dr. Robert Gaspar for their support, comments and suggestions and for their agreeing to serve as committee members. Thanks Dr. Govinda Raju for attending my defense.

Thanks are also directed to other faculty members, fellow graduate students and friends for their encouragement.

Finally, I would like to thank my husband, my daughter and my parents for their constant love, encouragement, and support.

Table of Contents

List of Figures

Number *Page*

List of Tables

Chapter 1

Introduction

1.1 Motivation of The Research

Design of smart vision sensors attracts a lot of academic and industrial interests, as solutions to many existing production problems require the inclusion of intelligent sensors into the manufacturing processes. The current research and design interests concentrate on hardware implementations and vision processing algorithms (see [1, 12, 20, 23, 24, 38, 42]).

The optical hardware currently used in non-contact smart vision sensors includes laser scanners, photo-diodes and cameras. Among them, the camera systems capture the image information of objects through a non-scanning approach and, hence, provide an efficient information source for conducting real-time software based position measurement and pattern recognition. Another advantage of using cameras is their low cost compared with laser scanners.

The significance of this thesis is that the algorithms developed can be used to design a real smart vision sensor, which can not only analyze and recognize the pattern of a product, but also find the 2-dimensional deviation or rotation angle of the product position compared with its reference template.

The significance of the design is also illustrated by a real engineering problem existing in the foam barrier assembly line for automobile door handle escutcheons. The motivation of the research is originally from this real engineering problem.

The story is that assembling is an important process in manufacturing door handle escutcheons. A door handle escutcheon consists of two parts: an escutcheon and a sheet

foam barrier. The escutcheon is the cover plate that lies behind the door handle in an automobile, while the foam barrier serves as a sealing layer on door handle escutcheon to avoid the ingress of dust and other unwanted material during the opening and closing of doors. This procedure requires that the foam barrier be fitted on the inner surface of the escutcheon through the hole on the escutcheon. Locating the foam barrier on the conveyor, which carries foam barriers in stream, picking up the barrier and pressing it onto the escutcheon are surprisingly difficult and time-consuming work. The practical difficulty is that the positions of foam barriers may deviate from their expected position due to the possible stretching behavior of foam material. The existing assembly platform lacks automatic locating and actuating functions and hence can not handle the assembly line automatically. The pre-configured robotic arm can not handle the foam barriers with position uncertainty because there is no position sensor involved in this process. On the other hand, simply adding a traditional position sensor to the robotic arm system can not solve this problem because:

1. Finding the correct positions of foam barriers involves a searching process, which can not be performed by the existing sensors.

2. It is difficult to identify special features on the foam material as the position indication.

3. The existing vision sensors are also not a suitable economic solution to this process and many other similar manufacturing problems because these sensors are usually designed to passively capture vision related features, but do not possess the capability of auto-searching for a target.

Clearly, new effective and economic solutions, one of which is shown in Figure 1.1, need to be figured out to accommodate such kind of engineering problems. A smart vision sensor needs to be developed to guide the robotic arm intelligently in lieu of the position changes of the foam barriers on the conveyor because of the possible stretching behavior of the foam material or other disturbance. In order to enhance the productivity of the door handle escutcheons, the new smart vision sensor should possess the following capabilities.

1. It can recognize the pattern of a product.

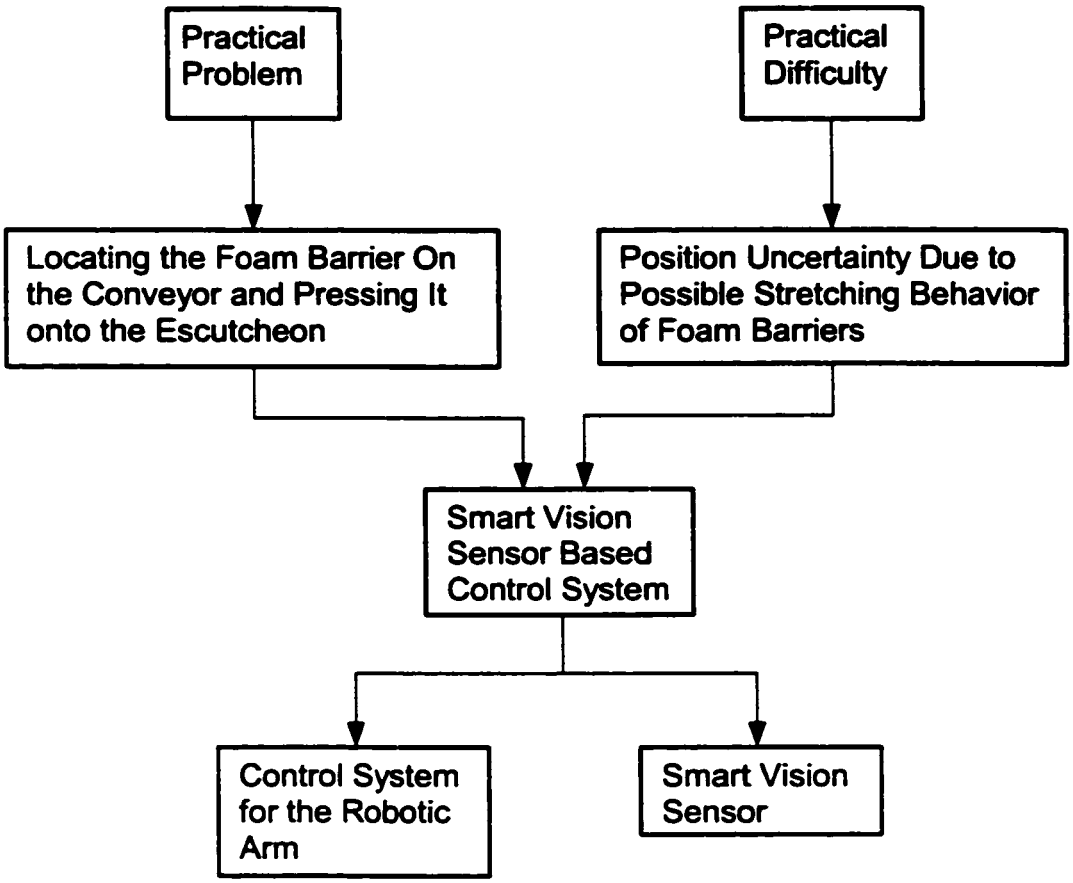

Figure 1.1 A Real Engineering Problem and The Solution

2. It can find the 2-*D* deviation and the rotation angle of the product position compared with its reference position.

3. It possesses searching capability to lock in the position (pattern) if there exists deviation between the real pattern and the template due to the position change of the object.

4. It should be in modular form in the sense that it can be easily re-configured to adapt to the changing products on the assembly line.

The proposed design in this thesis is targeted on providing such kind of smart vision sensor.

1.2 Literature Survey

In this section, some literature survey results for correlation method and shape pattern recognition methods are summarized.

1.2.1 Correlation Method

The correlation method for image processing has been applied to solving problems in different fields [3, 7, 8, 27, 30, 35], such as:

1. Automatic defect classification [7] —It requires that consistent quality images are captured on all tools. Image metrics have been developed and the variance of these metrics have been correlated to matching classifiers.

2. Obtaining a color motion stereo [27] —3-*D* shape recovery in motion stereo is formulated as a matching optimization problem of multiple color stereo images, which can be carried out with circular decorrelation of a color signal.

3. Chemical structure matching [3] —It is a process of matching the 3-*D* structure of molecules in chemical database within the framework of binary correlation matrix memories.

4. Image registration in remote sensing [35] —Registration is the process, which makes the pixels in two images precisely coincide to the same points on the ground.

5. Template matching [30] —Localization of a known pattern within the given images represents one of the most important tasks of digital image processing. This is carried out via correlating the pattern with the image in various positions (relevant sub-regions).

There are also some other applications. But the common point of all these applications is that the correlation analysis result actually returns the singularity in spatial comparison of two images or patterns. The template matching provides a good motivation of applying the correlation method to designing a smart vision sensor for position measurement which is going to be addressed in this thesis, since the position deviations, when reflected in images, are featured exactly by the singularity in spatial comparison.

1.2.2 Shape Recognition

In many applications of robotics and computer vision, object recognition is of paramount importance [10, 29, 43, 45, 46]. With regard to object recognition, shape representation and recognition is fundamental. Planar shapes arise in a variety of important computer vision applications including character recognition, biomedical application, manufacturing inspection, etc. This problem has, not surprisingly, received considerable attention in literatures [6, 25, 36, 37]. Some of the current research results are summarized as follows:

1. Curvature function [4, 26, 34] —It is an information preserving representation of a shape. It has invariant properties with regard to scaling, translation and rotation. It is a 1-*D* function.

2. Fourier descriptors [34, 39] —A shape is characterized by interpreting its boundary points as complex numbers and generating spectra from those representations using *FFT*.

3. Coordinate function [34] —It uses the contours of the objects directly. It does not need the representation of the shape.

4. Contour coding by chain codes [5, 15] —It is a more compact form to represent the shape.

5. Critical points [22, 44] —The major features of a shape are almost concentrated at the critical points with high curvature. It is a good idea to use the relatively simple features of the polygon formed by connecting every two neighboring critical points to characterize the original contour.

There are also some other methods such as: medial axis transformation, etc., which can also be applied in shape recognition.

In human perception, the major features of a shape are almost concentrated at the critical points with high curvature. The basis of the method is to divide a curve into segments and then use relatively simple features to characterize the shape. There are many methods based on locating the critical points for shape recognition in the past [15, 16, 17, 22, 28,

31, 40, 44]. Using the critical points to characterize the contour of a shape provides a good motivation of applying it to designing a smart vision sensor for pattern recognition, position measurement and rotation angle calculation. The position deviation and the rotation angle, when reflected in images, can be featured by the coordinates of the critical points. The polygon formed by connecting every two neighboring critical points can be used to characterize the contour of a shape, hence can be used to identify the pattern of the shape. In this thesis, Directional Flow-Change Method [22] is used for this purpose.

1.3 Design of A Smart Vision Sensor

The block diagram of a smart vision sensor is shown in Figure 1.2.

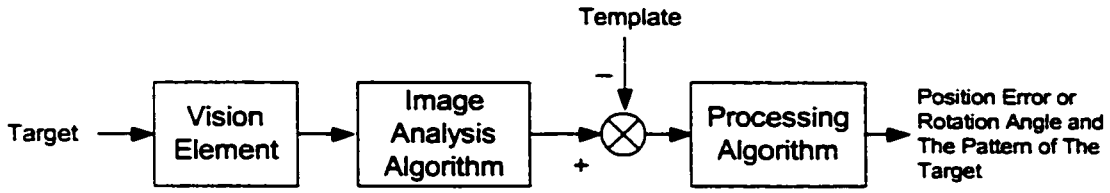

Figure 1.2 Block Diagram of A Smart Vision Sensor

In this configuration, a vision element (for example, a *CCD* camera) is needed to capture the image of an object. Then the image analysis algorithm is applied to processing the captured image. The processing algorithm is used to compare the processed image with the template, which represents both the reference position of the object and the pattern of the object.

The spatial shift between the two images will be obtained and translated into the physical position deviation of the object compared with the reference position. The rotation angle between the two images and the pattern of the object can also be obtained.

In this thesis, two algorithms based on Correlation Method and Directional Flow-Change Method are presented. It is pointed out that many other analysis methods such as Fast Fourier Transform [34, 39], etc., can also potentially be applied to the sensor design.

1.4 Thesis Organization

This thesis is organized as follows: Chapter 1 presents the research background, motivation for the research, literature survey results and the design idea. Chapter 2 introduces the correlation analysis method. A review of the Directional Flow-Change Method [22] is also provided. At the end of the chapter, some preliminary results are given. Chapter 3 introduces the Histogram, Cumulative Probability Density Function and Gray-level Transformation. The algorithm, which is based on the Correlation Method, is developed for finding the 2-*D* deviation of the input image compared with the template image. Chapter 4 presents the image binarization, edge detection and noise reduction, contour tracing, pattern recognition, deviation calculation and rotation angle calculation of the input image compared with the template. The algorithm based on the Directional Flow-Change method, which can identify the pattern of a planar shape and find the 2-*D* deviation or the rotation angle, is developed. Chapter 5 concludes this thesis by summarizing the two algorithms and making recommendations on future work.

Chapter 2

Methods for Image Processing and Shape Pattern Recognition

2.1 Introduction

In this chapter, the correlation analysis method is introduced first. Then, the *DFCM* method, which can identify the pattern of a planar shape, is reviewed. Finally, some preliminary results are given at the end of the chapter.

2.2 Correlation Analysis

One of the principle applications of correlation method in image processing is in the area of template matching. The maximum correlation value indicates the closest match. This is called the method of maximum likelihood. For dissimilar signals, the peak of the correlation function is an indicator of good or bad matches between two signals [35].

Definition 2.1 Given two $M \times N$ matrices G and F,

$$G = \begin{bmatrix} g_{11} & \cdots & g_{1N} \\ \cdots & \cdots & \cdots \\ g_{M1} & \cdots & g_{MN} \end{bmatrix}, \quad F = \begin{bmatrix} f_{11} & \cdots & f_{1N} \\ \cdots & \cdots & \cdots \\ f_{M1} & \cdots & f_{MN} \end{bmatrix},$$

the cross correlation of matrices G and F is a $(2M-1)\times(2N-1)$ matrix R, $R=[r_{ij}]$, $i=-M+1,\ -M+2,\ \ldots,\ -1,\ 0,\ 1,\ \ldots,\ M-1$; $j=-N+1,\ -N+2,\ \ldots,\ -1,\ 0,\ 1,\ \ldots,\ N-1$, and

$$r_{ij} = \sum_{k=1}^{M} \sum_{l=1}^{N} f_{kl} g_{(i+k)(j+l)} . \tag{2.1}$$

Note that, in the above expression, the terms, the subscript indices of which are out of range, are zeros.

The cross-correlation describes the general dependence between two data sets. It has its maximum value when the two matrices are aligned so that they are shaped as similarly as possible.

For the correlation analysis between two images, F and G represent the gray levels of the pixels. If two images are sufficiently similar, except for a relative spatial shift, it should result in a maximum value at the point of the best alignment. Therefore the spatial shifts along X and Y directions between two images can be calculated as follow:

$$deviations(X,Y) = \max_{i,j}(r_{ij}), \quad \begin{cases} i = -M+1, \ -M+2, \ ...., \ -1, \ 0, \ 1, \ ...., \ M-1; \\ j = -N+1, \ -N+2, \ ...., \ -1, \ 0, \ 1, \ ...., \ N-1. \end{cases} \tag{2.2}$$

However, there is a problem with equation (2.1) when there is a large change in the image brightness. Suppose image F is the template image, which is constant. But large changes in the image brightness in the input image G can lead to false correlation peaks. When applying the equation (2.1) to the images, the value of R will then largely depend on G and will not give a correct indication of the match. This problem can be solved by using normalized cross-correlation. The match measurement can be computed as follows:

$$c_{ij} = \sum_{k=1}^{M} \sum_{l=1}^{N} f_{kl} g_{(i+k)(j+l)} . \tag{2.3}$$

$$k1 = [\sum_{k=1}^{M} \sum_{l=1}^{N} f_{kl}^{\ 2}]^{1/2} . \tag{2.4}$$

$$k2 = [\sum_{k=1}^{M} \sum_{l=1}^{N} g_{kl}^{\ 2}]^{1/2} . \tag{2.5}$$

$$r_{ij} = c_{ij} / k2 . \tag{2.6}$$

Since $k1$ is a constant as long as the template does not change, it can be ignored in

locating the relatively maximum correlation. This definition is less sensitive to the mean value and it can be used to prevent false correlation peaks arising from the changes in the mean image gray level. Without this, the correlation will be the highest where the image has the highest gray levels, not where the pattern match is the best. There are also some other definitions for normalized cross-correlation.

Using the edged images to do correlation can also solve this problem.

2.3 Directional Flow-change Method

The Directional Flow Change Method (*DFCM*) [22] is based on the directional flow-change concept, which extracts shape features for shape recognition. It can locate critical points, evaluate angles, code angles, and finally get a set of shape numbers for shape recognition. It has good performance in efficiency, accuracy and effectiveness. The original purpose of the *DFCM* method is to identify the pattern of a planar shape. In this thesis, the method is improved to do pattern recognition, deviation calculation and rotation angle calculation.

This method is based on the concept of directional flow-changes along the contour points. When tracing the contour clockwise starting from any arbitrary point, all the points on the contour can be treated as an array:

$$\phi = \{p_0, p_1, p_2, \ldots, p_{N-1}\},$$

where N is the total number of contour points, p_0 and p_{N-1} are the starting point and the ending point, respectively. Since the contour of an object is a closed curve, ϕ becomes a periodic function of period N, which means $p_{-k} = p_{N-k}$ and $p_k = p_{N+k}$ for $0 \le k \le N-1$. A portion of a contour consisting of $2J+1$ points $p_{i-J}, \ldots, p_i, \ldots, p_{i+J}$ is denoted by $\phi_i(J)$, where $J = \gamma N$ with $0 < \gamma < 1$. J is called the 'Supported Length' and γ is called the 'Supported Rate' [16, 17].

2.3.1 Concept of Chain Codes

Chain code [15] can be used to represent the boundary of an object by a connected sequence of straight-line segments of specified direction. It is a more compact form to represent the shape. A chain code representation based on the 8-connectivity of the segments is adapted. Two consecutive pixel points define a segment. The direction of each segment is coded by using a numbering scheme, which is show in Figure 2.1.

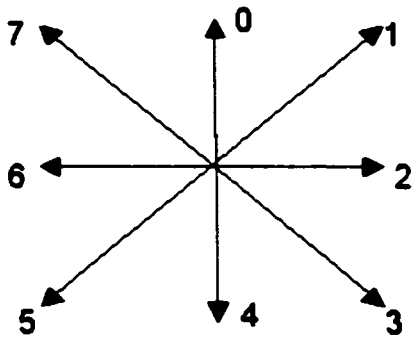

Figure 2.1 Numbering Scheme for 8-Connectivity Chain Code

Assume that $\{d_0, d_1, ..., d_{N-1}\}$ represent the 8-directional chain codes of the contour $\phi = \{p_0, p_1, p_2, ..., p_{N-1}\}$, where d_i is the direction of the segment from point p_i to p_{i+1}. A function $G_\alpha(d_i)$, $0 \le \alpha \le 7$, is defined for testing whether chain code d_i is in the α direction, i.e.

Definition 2.2 Function $G_\alpha(d_i)$ is defined as follow:

$$G_\alpha(d_i) = \begin{cases} 1 & \text{if} \quad d_i = \alpha, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

2.3.2 Concept of Directional Flow-Change

Definition 2.3 The input flow in the α direction of a contour segment with length J at

point $p_i \in \phi$ is defined as $\sum_{k=i-J}^{i-1} G_\alpha(d_k)$.

Definition 2.4 The output flow in the α direction of a contour segment with length J at point $p_i \in \phi$ is defined as $\sum_{k=i}^{i+J-1} G_\alpha(d_k)$.

Definition 2.5 The flow-change in the α direction at point $p_i \in \phi$ with contour segment of length J on both sides of p_i is defined as:

$$\delta_\alpha(i,J) = \left| \sum_{k=i}^{i+J-1} G_\alpha(d_k) - \sum_{k=i-J}^{i-1} G_\alpha(d_k) \right|. \tag{2.8}$$

Definition 2.6 The directional flow-change at point $p_i \in \phi$ with contour segment of length J on both sides of p_i is defined as:

$$\delta(i,J) = \sum_{\alpha=0}^{3} |\delta_\alpha(i,J) - \delta_{\alpha+4}(i,J)|. \tag{2.9}$$

To compute the directional flow-changes for all points on the contour is very efficient because the input and output flow of a point can be obtained from the input and output flow of its previous point with additional simple update operations as follows:

$$\sum_{k=i+1-J}^{i} G_\alpha(d_k) = \sum_{k=i-J}^{i-1} G_\alpha(d_k) - G_\alpha(d_{i-J}) + G_\alpha(d_i), \tag{2.10}$$

$$\sum_{k=i+1}^{i+J} G_\alpha(d_k) = \sum_{k=i}^{i+J-1} G_\alpha(d_k) - G_\alpha(d_i) + G_\alpha(d_{i+J}). \tag{2.11}$$

From physical scenario's point of view, the existence of curvature at a point p_i of a curve segment $\phi_i(J)$ will cause a directional flow-change at p_i. In other words, if there is a significant amount of directional flow-change $\delta(i,J)$ detected at point p_i, the curve segment $\phi_i(J)$ should have non-ignorable curvature at this point and point p_i is probably a critical point.

Definition 2.7 A critical point $p_i \in \phi$ is a point, which satisfies the following conditions

simultaneously:

Step 1. $\delta(i,J) > t \times J$ for some t with $0.8 \le t \le 1$.

Step 2. $\delta(i,J) \ge \delta(k,J)$ for all k with $i-L \le k \le i+L$, where L is an integer such that $0.5J \le L \le J$.

Step 3. If p_k is the next critical point after p_i, then there exists at least a point p_j with $i < j < k$ such that $\delta(j,J) \le \theta \times \delta(i,J)$, where $0.5 \le \theta \le 1$.

Step 4. Let $p_{i'}$ be the previous critical point before p_i; $p_{k'}$ be the first point after $p_{i'}$ such that $\delta(k',J) \le \theta \times \delta(i',J)$; and p_k be the first point after p_i such that $\delta(k,J) \le \theta \times \delta(i,J)$; If there are m points $\{p_{i1},\ldots,p_{im}\}$ between $p_{i'}$ and p_i such that $\delta(i_1,J) = \cdots = \delta(i_m,J)$, then $i = [\frac{1}{2}(i_1 + i_m)]$.

Step 5. $|i-k| \ge L$ If p_k is a critical point other than p_i.

2.3.3 Algorithm for Detecting Critical Points

Based on Definition 2.7, the critical point detecting algorithm is presented as follows:

Step 1. Pick a number γ between 0.01 and 0.05. Let $J = \gamma \times N$.

Step 2. Choose the values for θ, t and L such that $0.5 \le \theta \le 1$, $0.8 \le t \le 1$ and $0.5J \le L \le J$.

Step 3. *Critical_points* $\leftarrow$ *NULL*.

Step 4. Starting from p_0, search for the first two points p_p and p_k with $p < k$ such that $\delta(p,J) \ge t \times J$; $\delta(j,J) \le \delta(p,J)$, for all $0 \le j \le k$ and $\delta(k,J) \le \theta \times \delta(p,J)$.

Step 5. $i \leftarrow k$; $d \leftarrow k$.

Step 6. Increase i while searching for the first point $p_{d'}$ and all points $p_{p'}$ with $p' < d'$ such that $\delta(p',J) > \delta(j,J)$ for all j's with $d \le j \le d'$; $\delta(p',J) > t \times J$ and

$\delta(d', J) \leq \theta \times \delta(p', J)$. At every new point p_i, check if (i mod N) = k. If true, then output *Critical_points* and terminate.

Step 7. If more than one such $p_{p'}$ points between p_d and $p_{d'}$ are found in step 6, say $\{p_{i1}, \ldots, p_{im}\}$, then $candidate \leftarrow [\frac{1}{2}(i_1 + i_m)]$; otherwise $candidate \leftarrow p'$.

If *Critical_points* = *NULL*, then

(a) Add $p_{candidate}$ to *Critical_points*;

(b) $previous \leftarrow candidate$;

(c) $d \leftarrow d'$; Go to step 6.

Step 8. If $(candidate - previous) \geq L$, then

(a) Add $p_{previous}$ to *Critical_points*;

(b) $previous \leftarrow candidate$;

else if $\delta(previous, J) < \delta(candidate, J)$, then

$previous \leftarrow candidate$.

Step 9. $d \leftarrow d'$; Go to step 6.

2.3.4 Shape Description

By locating the critical points, a polygon with these critical points as vertices can be formed to approximate the contour of the shape. Thus, the sides of the approximating polygon are those line segments connecting every two neighboring critical points. Two connecting sides form an angle at a critical point. Consequently, the angles can be computed easily and the resulting sequence of successive angles can be used to characterize the contour. The possible range of an angle computed by the method is from 0° to 360°.

To facilitate shape matching, a sequence of angles are converted into a sequence of angle codes, called 'Angle String', based on the coding scheme shown in Table 2.1.

In this angle coding scheme, if an angle is between 160° and 200°, it will be removed

from the sequence because it is close to a straight line and is too sensitive to be treated as a curving feature.

Table 2.1 Angle Coding Scheme

Angle Size	Angle Code	Angle Size	Angle Code
0~29	0	200~209	8
30~39	1	210~239	9
40~69	2	240~249	A
70~79	3	250~279	B
80~109	4	280~289	C
110~119	5	290~319	D
120~149	6	320~329	E
150~159	7	330~359	F

An angle string such as *42B22* can be treated as a hexadecimal number for comparison.

To reduce possible over-sensitivity in shape recognition, the following rules are applied to angle strings:

1. Code 7 can be either removed from the string or changed into code 6.
2. Code 8 can be either removed from the string or changed into code 9.
3. Code 1 can be changed to either code 0 or code 2.
4. Code 3 can be changed to either code 2 or code 4.
5. Code 5 can be changed to either code 4 or code 6.
6. Code *A* can be changed to either code 9 or code *B*.
7. Code *C* can be changed to either code *B* or code *D*.

8. Code E can be changed to either code D or code F.

By applying the above rules, some additional angle strings can be generated.

Definition 2.8 A 'Shape Number' is defined as a permutation of an angle string such that this permutation forms an integer of the minimum angle.

Thus, the above angle strings can be converted into the corresponding shape numbers.

2.3.5 Shape Recognition

The shape recognition scheme can be described as follows:

Given any shape, a finite set of shape numbers can be generated by the method described above. Let $A=\{A_1,....,A_n\}$ and $B = \{B_1,....,B_m)$ be the sets of shape numbers associated with shape S_A and shape S_B, respectively. Then S_A and S_B are of the same shape if there exist some $A_i \in A$ and some $B_j \in B$ such that $A_i = B_j$.

2.4 Preliminary Results

Definition 2.9 The directional flow-change at point $p_i \in \phi$ with contour segment of length J on both sides of p_i is defined as:

$$\delta(i,J) = \sum_{\alpha=0}^{7} |\delta_\alpha(i,J)|. \tag{2.12}$$

This is the revision of Definition 2.6, which could cause some problems in some cases. For example, suppose there is a curve as shown in Figure 2.2. There are total 11 pixels with J=5 (Supported Length). For the 6th pixel p_i, the flow-changes $\delta_\alpha(i,5)$ in the direction 0, 1, 2, ... , 6, 7 are:

$$\delta_0(i,5)=0, \quad \delta_1(i,5)=0, \quad \delta_2(i,5)=5, \quad \delta_3(i,5)=0,$$

$$\delta_4(i,5)=1, \quad \delta_5(i,5)=4, \quad \delta_6(i,5)=0, \quad \delta_7(i,5)=0.$$

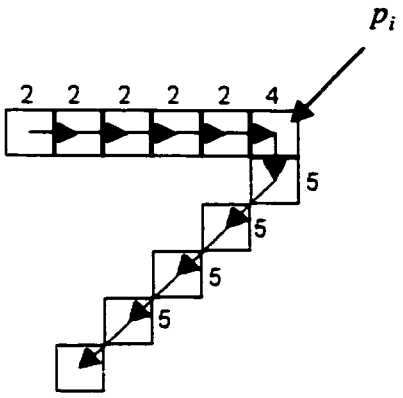

Figure 2.2 δ Function of A Curve

According to Definition 2.6, the directional flow-change at point p_i is:

$$\delta(i,5) = \sum_{\alpha=0}^{3} | \delta_\alpha(i,5) - \delta_{\alpha+4}(i,5) | = | 5-1-4 | = 0.$$

No directional flow change is directed at point p_i. But obviously, it could be a critical point. Using Definition 2.8, the δ value of this point can be calculated as:

$$\delta(i,5) = \sum_{\alpha=0}^{7} | \delta_\alpha(i,5) | = | 5+1+4 | = 10.$$

Non-ignorable directional flow-change is detected at this point. Therefore, it could be a critical point.

Definition 2.10 Given any shape, a finite set of shape numbers can be generated by the modified method. Let $A=\{A_1,....,A_n\}$ and $B = \{B_1,....,B_m\}$ be the sets of shape numbers associated with shape S_A and shape S_B, respectively. Then S_A and S_B are of the same shape if there exist some $A_i \in A$ and some $B_j \in B$ such that $A_i = B_j$ and if the corresponding side lengths of the polygons, which are used to represent the contours the shapes, are also

approximately the same.

Two supplemental procedures as described below are implemented for the algorithm of critical point detection. The third one will be introduced in Chapter 4.6. Comparing with the *DFCM* method developed in [22], the modified method (with the new definitions and the supplemental procedures) has better performance and is more accurate.

1. A Low Pass Filter Introduced to The δ Function

If the δ Function is processed through a low pass filter, a smoother δ function, which will result in better performance for the algorithm, can be obtained. This filter just takes the average value of δ Function for *L/2* points (*L/4* points on both sides of a certain point).

2. A Supplemental Procedure Implemented

The *DFCM* method could cause some problems. For example, suppose there is a δ function as shown in Figure 2.3. According to step 4 of the algorithm for detecting the critical points in Chapter 2.3.3, two points p_p and p_k, which satisfy the following conditions, can be found:

$\delta(p,J) \geq t \times J$; $\quad \delta(j,J) \leq \delta(p,J)$, for all $0 \leq j \leq k$ and $\delta(k,J) \leq \theta \times \delta(p,J)$.

Therefore, point p_p is a critical point. And according to step 6, two points $p_{d'}$ and $p_{p'}$ are also found, which satisfy the following conditions:

$$\delta(p',J) > t \times J, \delta(d',J) \leq \theta \times \delta(p',J).$$

But the condition $\delta(p',J) > \delta(j,J)$ for all j's with $d \leq j \leq d'$ (d=k) is not satisfied because at least $\delta(d,J) > \delta(p',J)$. But actually, $p_{p'}$ could be a critical point. So if the value of k is changed such that k locates on the point *MIN*, which has the local minimum value, this problem can be solved.

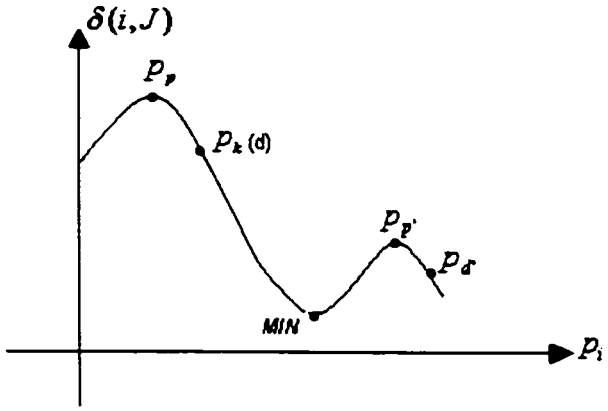

Figure 2.3 δ Function of A Shape

Therefore, in the algorithm for detecting the critical points, a new procedure is added following step 4 to change the value of k so that k will be the local minimum value. This procedure will make the algorithm more complete and accurate. The same thing should be done for step 6.

Chapter 3

Algorithm Design for Smart Vision Sensors Based on The Correlation Method

3.1 Introduction

In this chapter, the Histogram and Cumulative Probability Density Function, and Gray-Level Transformation are introduced first. Then the algorithm, which is based on the correlation method, is developed for finding the deviation of the target in the input image compared with the reference position of the template. Finally, two examples will be given to demonstrate the effectiveness of the algorithm.

3.2 Histogram and Cumulative Probability Density Function

The histogram and the Cumulative Probability Density Function (*CPDF*) of an image are defined as follows:

Definition 3.1 Given an image with $M \times N$ pixels, let $f(i, j)$ be the discrete gray-level at the pixel (i, j), $i=1, 2, ..., M$; $j=1, 2, ..., N$. The histogram $H(z)$ of this image is defined as the total number of pixels at which $f(i, j)=z$. The *CPDF* is defined as:

$$CPDF(z) = \sum_{i=0}^{z} [H(i)/(M \times N)]. \qquad (3.1)$$

The image histogram describes the statistical distribution of the gray-level intensities over the image without reference to their locations, but only to their frequencies of occurrence. It is calculated simply by counting the number of pixels in each gray level. With the histogram, a distribution of gray levels of pixels in an image can be described. It

is a useful tool for image contrast enhancement. The appropriate gray-level thresholds can be obtained from the histogram as the percentage of the total number of the pixels in the image.

The *CPDF* of an image is the fraction of pixels with gray-levels up to a given value over the all pixels. It is a single-valued monotonic function since it can only increase as each histogram value is accumulated from the minimum gray level. The asymptotic maximum value for the *CPDF* is one.

3.3 Gray-Level Transformation

Point processing in an image means that the processing of a certain pixel in the image depends on the information on the pixel itself without considering the status of its neighborhood. One method for point processing is the Gray-Level Transformation. It is a quite straightforward method. A conversion table or algebraic expression will be stored, and the gray-level transformation for each pixel will be carried out either by table lookup or by algebraic computation. Figure 3.1 shows a gray-level transformation, which can be used in image contrast enhancement.

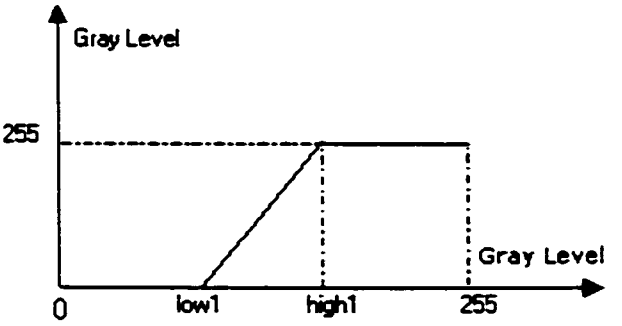

Figure 3.1 Gray-Level Transformation

Clearly, it can be seen that the intensity of pixels with gray-level lower than '*low1*' are set to 0 and those with gray-level higher than '*high1*' are set to 255. Then do linear

stretching for the remaining pixels. By doing such transformation, the dark part of the image becomes darker and the bright part becomes brighter.

3.4 Algorithm Design Based on The Correlation Method

To explain the idea, a sheet foam barrier on a conveyor is used as an example to show the algorithm design of the smart vision sensor. A typical image of sheet foam barriers on the conveyor is shown in Figure 3.2.

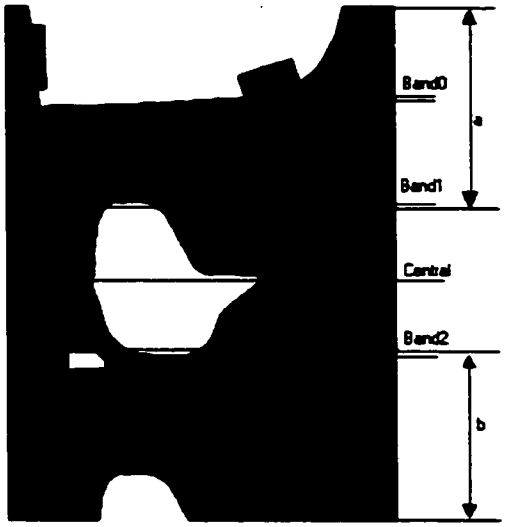

Figure 3.2 Image of Foam Barriers

Figure 3.3 shows the template image indicating the reference position of a complete foam barrier within a captured image. The vision sensor is designed to identify the position deviation of the real foam barrier on the conveyor by comparing the real image with the template image.

It is assumed that the input image covers up to the range of two foam barriers so that it is guaranteed that at least one complete foam barrier is included in the image. There are three cases for the position relationship between the reference position and the on-line foam barrier position:

Figure 3.3 Image of The Template

Case 1. The target foam barrier locates around the middle of the image (Figure 3.2).

Case 2. The target foam barrier locates at the top of the image (Figure 3.4). This extreme case shows that the vertical deviation is '*-a*' (Figure 3.2).

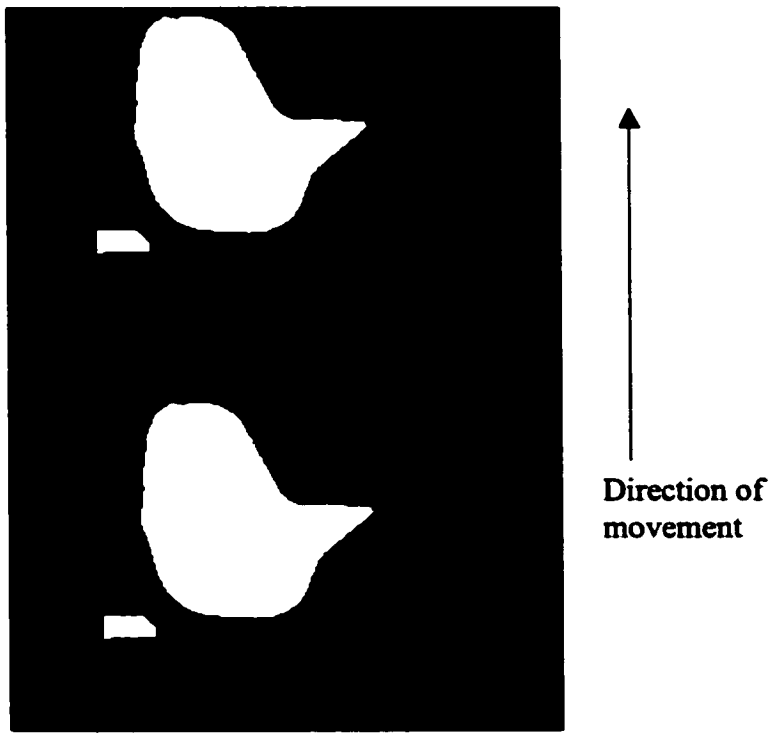

Figure 3.4 An Extreme Case Showing That The Deviation Is '*–a*'

Case 3. The target foam barrier locates at the bottom of the image (Figure 3.5). And this

extreme case shows that the deviation is '$+b$' (Figure 3.2).

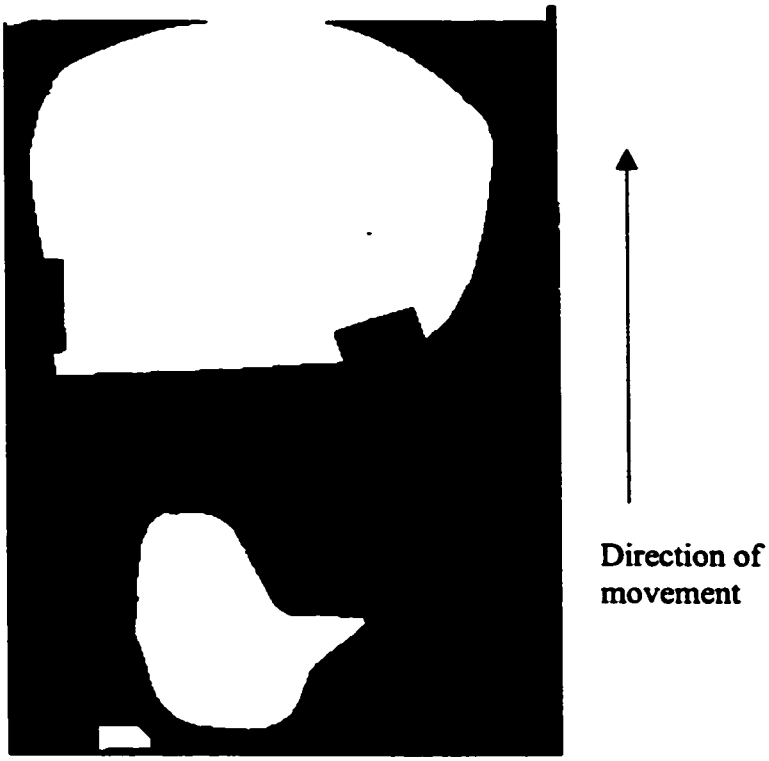

Figure 3.5 An Extreme Case Showing That The Deviation Is 'b'.

Thus, for this algorithm, the measurable range is $-a$~$+b$ along the vertical direction.

Six steps are conducted to implement the algorithm, the flow chart of which is also shown in Figure 3.6.

Step 1. Enhance the contrast of the input image.

Step 2. Find the approximate position of the complete foam barrier in the input image.

Step 3. Crop the input image to the same size as that of the template image.

Step 4. Find the edges of the template image and the cropped image.

Step 5. Calculate the correlation between the edged template image and cropped image.

Step 6. Calculate the deviations in both X-direction and Y-direction.

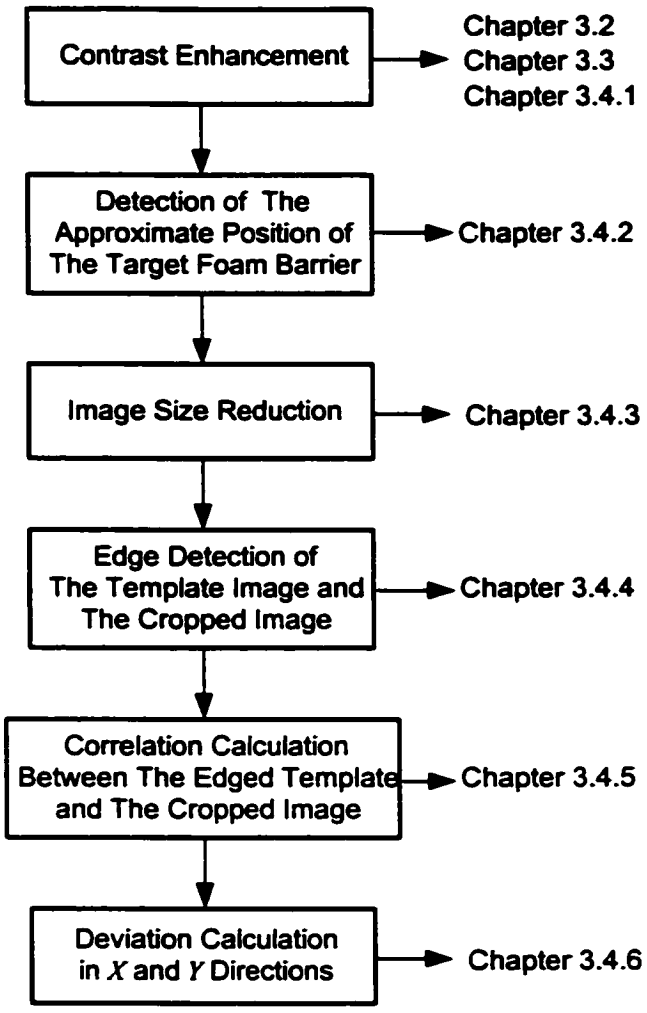

Figure 3.6 Flow Chart of The Algorithm Based on The Correlation Method

3.4.1 Enhance The Contrast of The Input Image

Because of the difficulties experienced in evaluating the criteria for image enhancement, and the fact that image enhancement is so problem-oriented, no general approaches are available that can be used universally. There are several ways to address this problem, one of which is the deterministic gray-level transformation. This method will be used together with the *CPDF* of the input image to enhance the contrast of the image.

A preferred captured image is an image, the pixels of which in the 'White Part' have the maximum gray level 255 (called value '*high*'), and the pixels of which in the 'Black Part' have the minimum gray level 0 (called value '*low*'). But in most real cases, because of the variable light luminance, different reflection factor of the target material and some other disturbance, the captured image could be very bad with the following situation.

1. Neither the value of '*high*' is 255, nor the value of '*low*' is 0.

2. Not all the pixels in the 'White Part' of the image have the same brightness; neither do the pixels in the 'Black Part'.

3. Sometimes, the contrast might be very low, which means the values of '*high*' and '*low*' are very close.

Therefore, enhancing the contrast of the image is necessary. The distribution of the intensity of an image is shown in Figure 3.7.

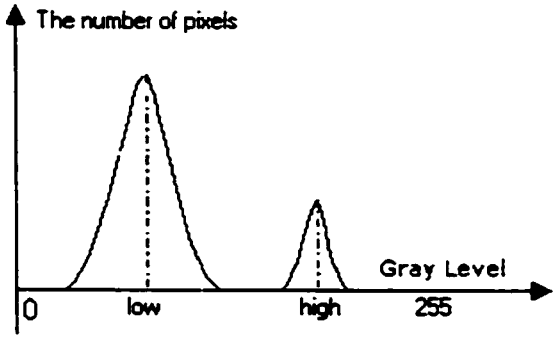

Figure 3.7 Intensity Distribution of The Input Image

In order to enhance the contrast of an image, the histogram and the *CPDF* of the image need to be obtained first. Then the values of '*high1*' and '*low1*' in Figure 3.1 can be found. Starting from the lowest gray-level, the number of pixels increases until the sum reaches 20% of the total number of the pixels in the image, which can be expressed as:

$$CPDF(low1) = 20\% . \qquad (3.2)$$

The corresponding gray-level is called '*low1*'. Using the formula as shown in equation (3.3), the value of '*high1*' can also be obtained. Starting from the highest gray-level, the

number of pixels increases until the sum reaches 20% of the total number of the pixels in the image, which can be expressed as:

$$CPDF(high1) = 1 - 20\% = 80\%. \tag{3.3}$$

The corresponding gray-level is called '*high1*'. With the values of '*low1*' and '*high1*' found, the gray-level transformation method mentioned in Chapter 3.3 can be used to process the image and the contrast-enhanced image can be obtained.

3.4.2 Find The Approximate Position of The Complete Foam Barrier in The Input Image

In order to reduce the computing time required to run the program, as well as to improve the positioning accuracy and the reliability, the input image should be cropped to the same size as that of the template.

One obvious characteristic of the foam barriers is that a narrow black band lies between every two neighboring foam barriers. The following 4 steps can be conducted to find the approximate position of the target foam barrier.

Step 1. Looking for the 1st 'Black Band' in the input image

Suppose $G(i,j)$ represents the gray level of a certain pixel (i,j) in the input image ($M \times N$ pixels). Using equation (3.4), the summation of the gray levels of all the pixels in each row is carried out.

$$y(i) = \sum_{j=1}^{N} G(i,j). \tag{3.4}$$

Then the maximum value '*MaxValue*' can be found according to the following equation.

$$MaxValue = max(y(i)). \tag{3.5}$$

If a certain $y(i)$ is less than or equal to 20% of the '*MaxValue*', the corresponding row is called a 'Black Line'; otherwise, it is called a 'White Line'.

The following procedures are conducted to find the 1st 'Black Band'.

First of all, the 1st 'Black Line' should be found. Then check the succeeding 4 lines to see if there are at least 3 'Black Lines'. If this condition is satisfied, it means that the 1st 'Black Band', which has at least 4 'Black Lines' out of 5 lines, is found. The allowable one 'White Line' may be caused by some noise. Therefore, this algorithm has the ability of fault tolerance. The 1st 'Black Band' is 'Band0' as shown in Figure 3.2. Obviously, the part above this band does not include the target foam barrier, but is an empty area, from which a foam barrier was just picked up.

It is explained implicitly in step 1 why the contrast of the image needs to be enhanced. In the case when the contrast is too low, and 20% is still used to look for the 'Black Band', it will result in a failure.

Step 2. Looking for the 1st 'White Band' in the input image

'White Band' is defined as a band, which has at least 4 'White Lines' out of 5 consecutive horizontal lines. Following a similar procedure as step 1, the 1st 'White Band' can be found, which is shown as 'Band1' in Figure 3.2. This band should be the approximate beginning of the target foam barrier.

Step 3. Looking for the 2nd 'Black Band'

Following the same procedure as step 1, the 2nd 'Black Band', which is shown as 'Band2' in Figure 3.2, can be found. This should be the approximate end of the target foam barrier.

Step 4. Looking for the approximate 'Central Line' of the target foam barrier

A foam barrier may be found between the 1st 'White Band' and the 2nd 'Black Band'. Now, the 'Central Line' needs to be found, which represents the approximate vertical central position of the target foam barrier. The position of the 'Central Line' (Figure 3.2) can be calculated as:

$$Central=(Band1+Band2)/2. \quad (3.6)$$

3.4.3 Crop The Input Image to The Same Size as That of The Template Image

In the 1st case as shown in Figure 3.2, it is easy to crop the image. Both below and above the 'Central Line', two image sections with *M/2* rows respectively (*M* is the number of total rows of the template image) are retained.

But special attention needs to be paid to case 2 (Figure 3.4) and case 3 (Figure 3.5). In case 2, the width of the part above the 'Central Line' is less than *M/2* rows and in case 3, the width of the part below the 'Central Line' is less than *M/2* rows too. Therefore, in these 2 cases, a different method needs to be used. In case 2, the image section with *M* rows starting from the first row is retained, and in case 3, the image section with *M* rows starting from the last row is retained.

3.4.4 Find The Edges of The Template Image and The Cropped Image

In order to prevent false correlation peaks arising from the changes in the mean image gray-level, the edges of the template and the cropped image have to be detected first. Then the correlation is performed between the two edged images. The built-in function 'edge' in *Matlab* can be used to detect the edge of an image. This function normalizes any edge values to 1 and non-edge values to 0. That is why the false correlation peak problem can be solved.

The normalized definition of the cross-correlation can also be used to solve the problem.

3.4.5 Calculate The Correlation Between The Template Image and Cropped Image

In equation (2.1), if *F* and *G* represent the gray-levels of two images, then the summation is carried out for those values of *k* and *l*, such that the computation is restricted within the

region of supported area only, the overlapping area between the two images. This corresponds to the shaded area in Figure 3.8. The location of (i, j) for which r_{ij} has the maximum value denotes the correct displacement between images G and F.

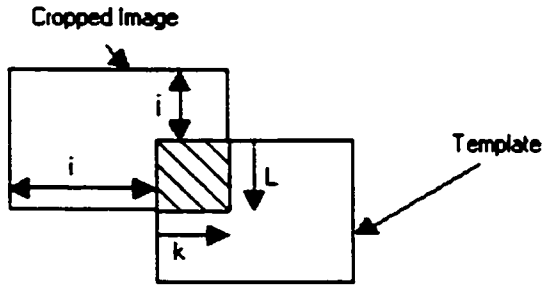

Figure 3.8 The Correlation Between The Template and The Cropped Image

The cross-correlation between the template and the cropped image is calculated by sliding the cropped image over the template, multiplying the two arrays pixel-by-pixel, and summing the result. A useful way to think of area correlation is that, if the two images are identical, the correlation becomes a spatial statistic on a single image and behaves similarly, with a peak for zero lag (shift) and a decrease as the lag increases. The point, which has the maximum correlation value, indicates the X and Y deviations. For the complete overlap, the correlation reaches the maximum. It is anticipated that the correlation will decrease fairly rapidly away from this peak as the shift between the 2 images increases.

The correlation calculation is performed between the edged template image and the edged cropped image in this chapter.

3.4.6 Calculate The Deviations in Both *X*-direction and *Y*-direction

As long as step 3.4.5 is finished, it is not difficult to find the deviations in both X-direction and Y-direction. The point with the maximum correlation value should be found first. Then some corrections need to be done because the final deviations are related to the value of '*Central*', M (number of the total rows of the template image), MI (number

of the total rows of the input image), and the *X-Y* deviations found in step 3.4.5.

3.5 Examples

To demonstrate the effectiveness of the method, let us look at an example. Figure 3.9 is an image with very low contrast. The histogram and the *CPDF* of the image are illustrated in Figure 3.11 and Figure 3.12 respectively.

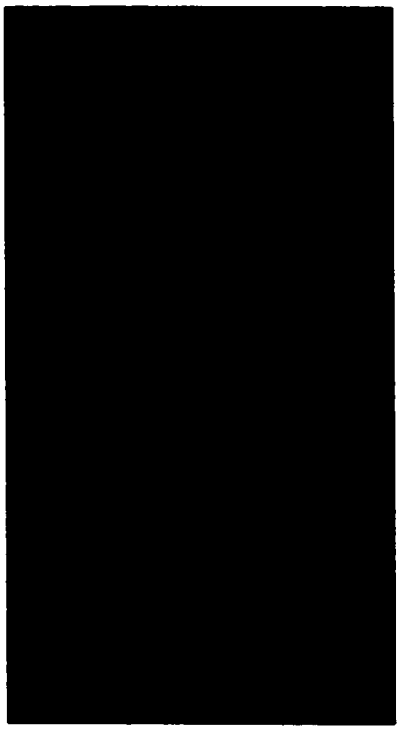

Figure 3.9 An Image with Very Low Contrast

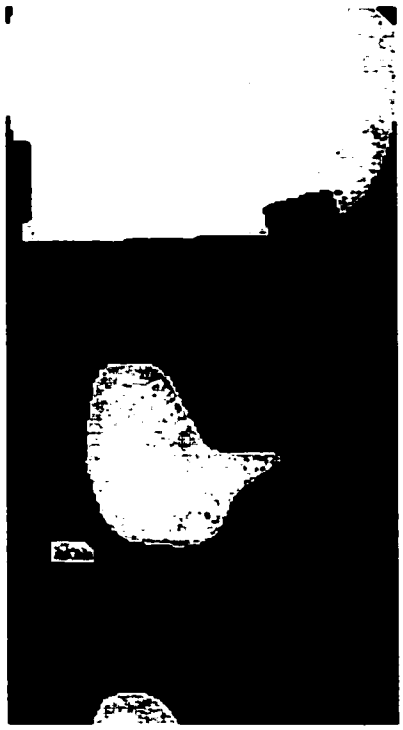

Figure 3.10 Contrast-Enhanced Image

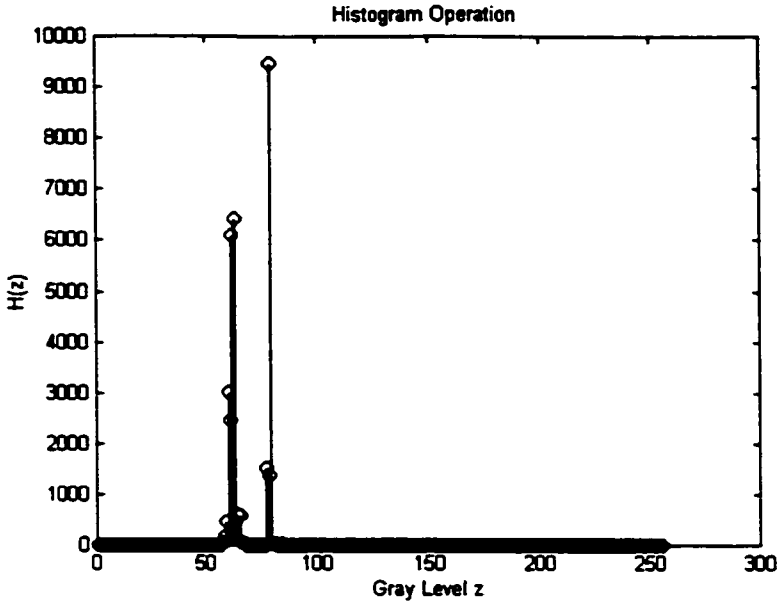

Figure 3.11 Histogram of The Input Image (Figure 3.9)

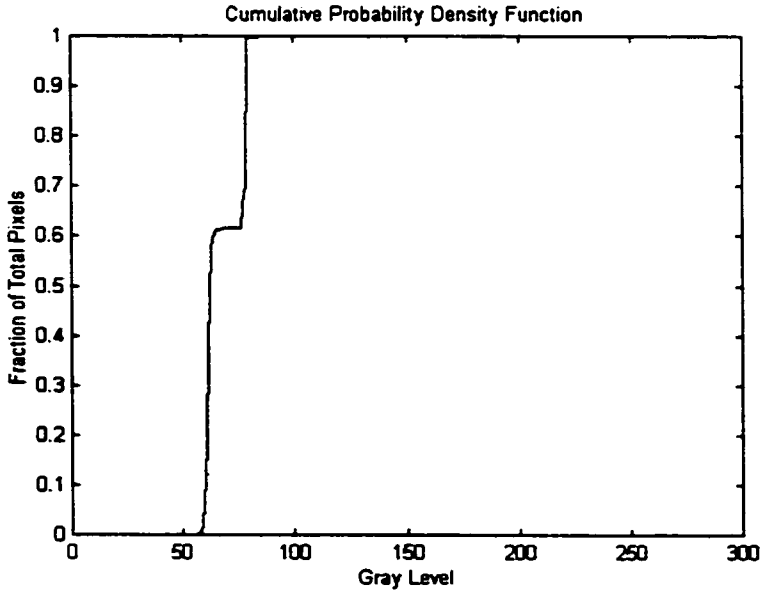

Figure 3.12 Cumulative Probability Density Function of The Input Image

Following the algorithm mentioned in Chapter 3.4.1, the contrast-enhanced image as shown in Figure 3.10 can be obtained. It can be seen that the contrast of the image is

greatly enhanced although the original image has very low contrast. The dark part in the image becomes darker and the bright becomes brighter.

The histogram and the *CPDF* of the contrast-enhanced image are illustrated in Figure 3.13 and Figure 3.14 respectively.

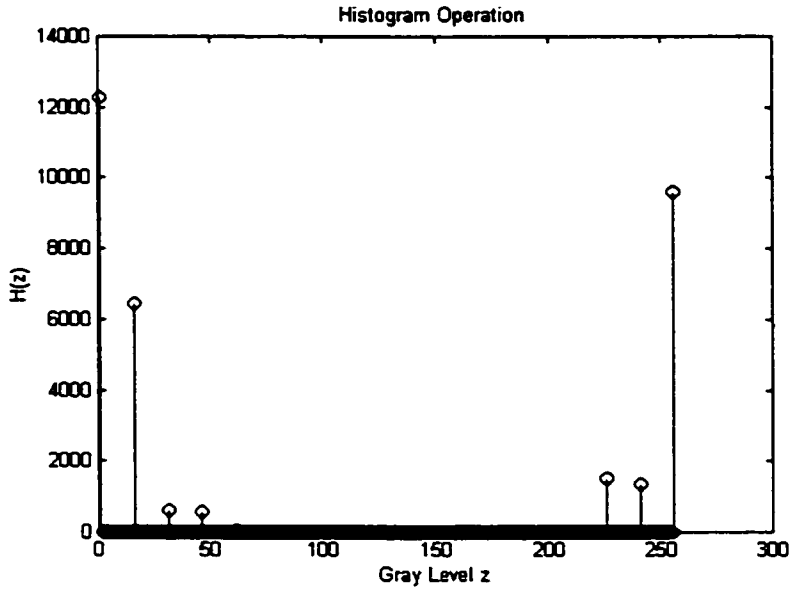

Figure 3.13 Histogram of The Contrast-Enhanced Image

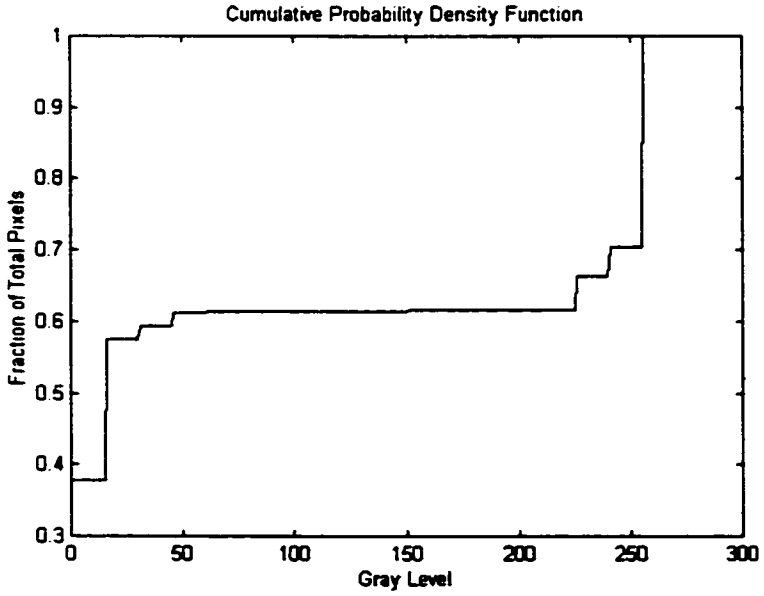

Figure 3.14 Cumulative Probability Density Function of The Contrast-Enhanced Image

Figure 3.15 and 3.16 are the edged images of the template and the cropped image respectively, which are going to be used to perform the correlation calculation.

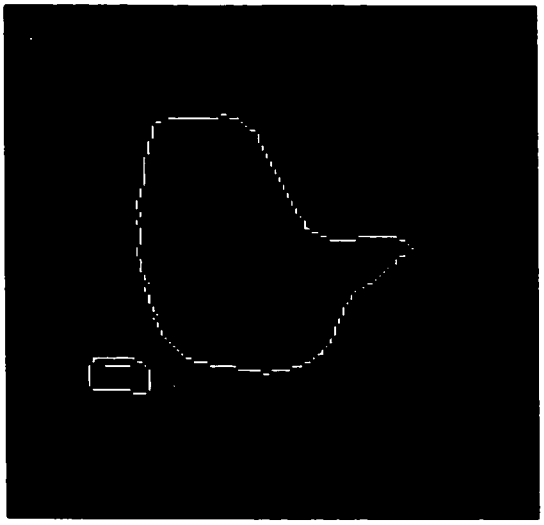

Figure 3.15 The Edged Image of The Cropped Image

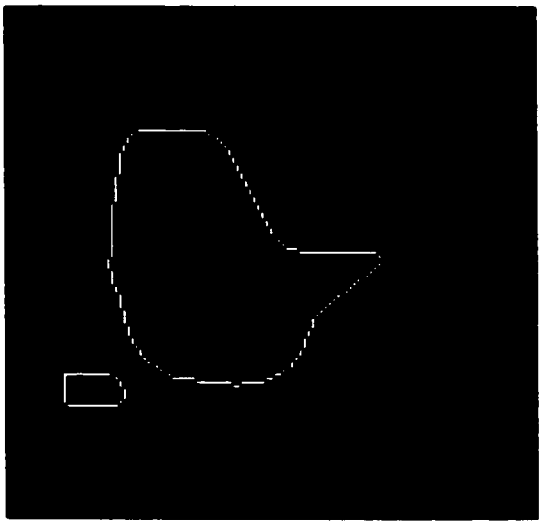

Figure 3.16 The Edged Image of The Template

Suppose the maximum value of the autocorrelation of the template image is

'*MaxAutoCor*'. In order to show the degree of the match between the input image and the template image, all the correlation values are scaled by a factor '1/*MaxAutoCor*'. If the correlation value after the scaling is 1, it means that the input image and the template have the best match. From Figure 3.17, which is the correlation result, it can be seen that the correlation reaches the maximum on the best match point, whereas on the other points, the correlation values decrease rapidly.

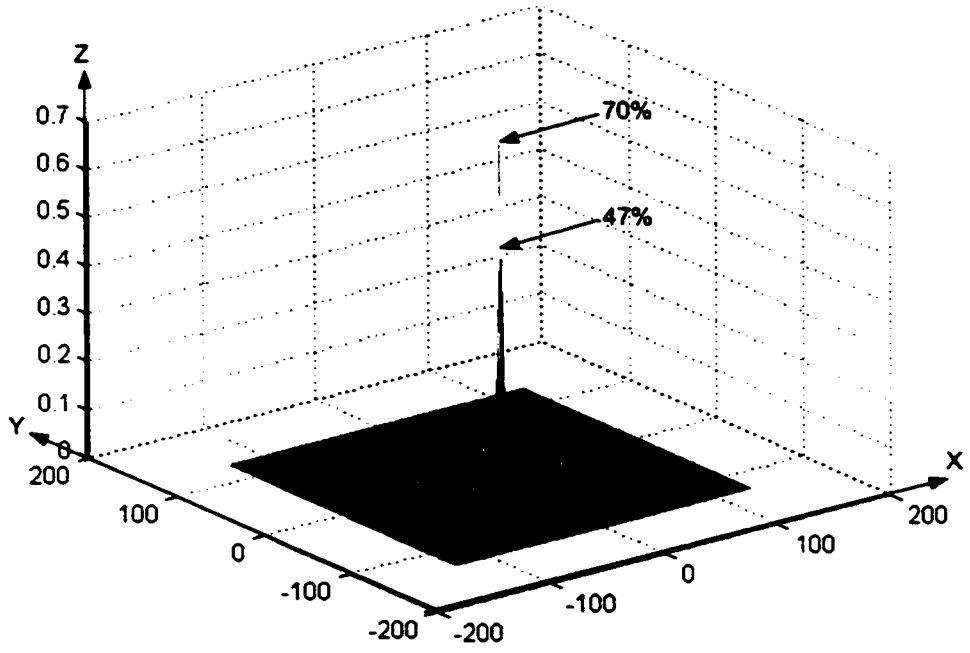

Figure 3.17 Correlation Result

Position errors found by the algorithm are:

X-deviation=34 (*pixels*), *Y-deviation*=-7 (*pixels*).

Position errors found manually are:

X-deviation=33 (*pixels*), *Y-deviation*=-7 (*pixels*).

Figure 3.18 is a diagram showing the position relationship between the contrast-enhanced image and the template image.

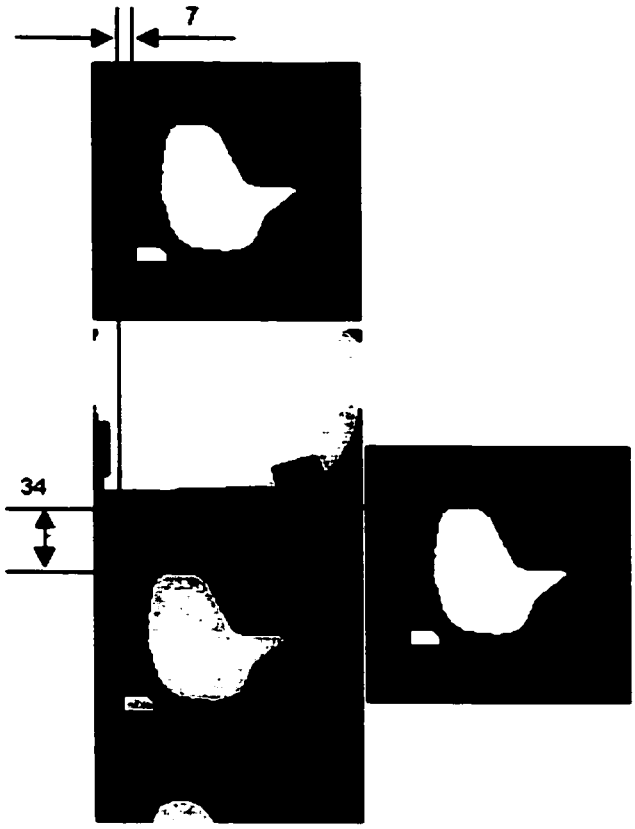

Figure 3.18 Contrast-Enhanced Image Compared with The Template Image

The result shows that the algorithm developed in this thesis has very high solution and noise rejection ability.

Let us look at another example shown in Figure 3.19, 3.20 and Figure 3.21.

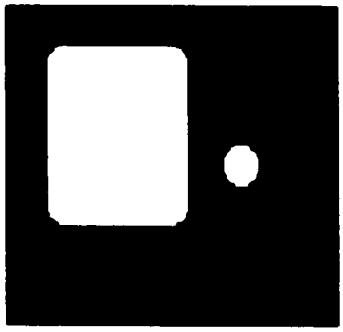

Figure 3.19 Image of A Different Shape

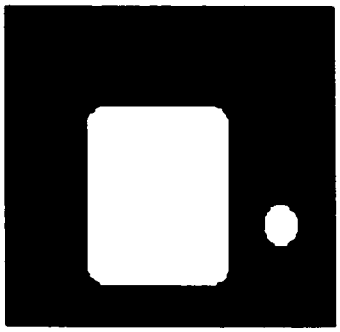

Figure 3.20 Image of The Template

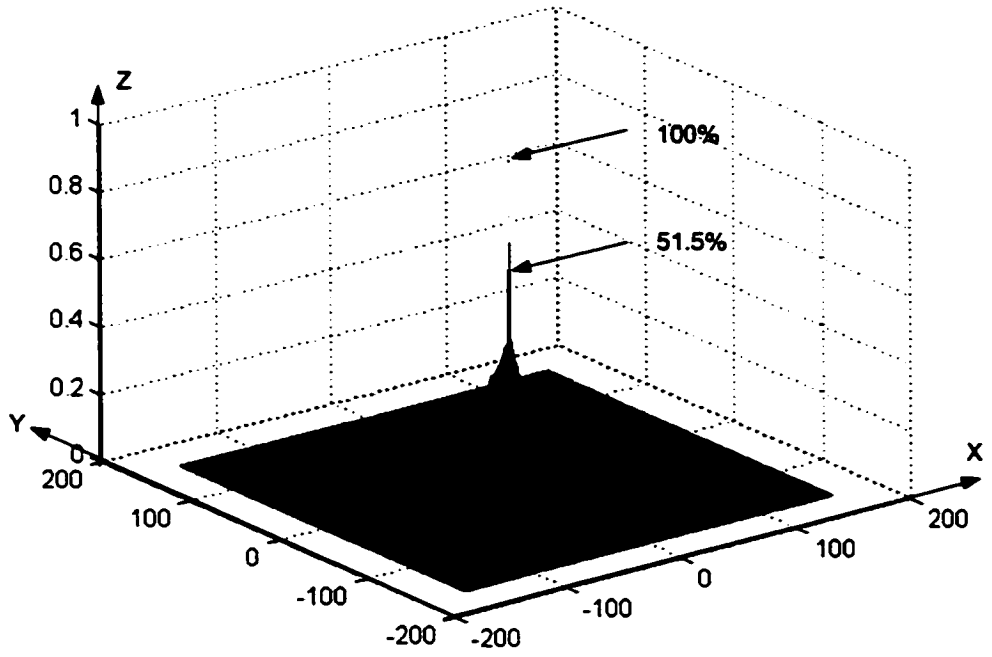

Figure 3.21 Correlation Result

The algorithm also works well for a different shape.

Position errors found by the algorithm are:

X-deviation=-30 (*pixels*), *Y-deviation*=-20 (*pixels*).

Position errors found manually are:

X-deviation=-30 (*pixels*), *Y-deviation*=-20 (*pixels*).

Chapter 4

Algorithm Design for Smart Vision Sensors Based on The Directional Flow-Change Method

4.1 Introduction

Some algorithms are developed in this chapter for image binarization, edge detection, noise reduction, contour tracing, pattern recognition, deviation calculation and rotation angle calculation. Based on the modified *DFCM* method, an algorithm, which can be applied to the smart vision sensor design, is obtained. Implementation of the algorithm is presented in details. At the end of this chapter, 4 examples are given to demonstrate the efficiency of the algorithm.

4.2 Image Binarization

The operation that converts a grayscale image into a binary image is known as binarization, which is carried out using a threshold value. Each pixel in the input image is assigned a new value (1 or 0) according to its intensity value. The threshold can be obtained from the histogram of the image.

4.3 Image Edge Detection

By using differentiation, a sharpening effect can be expected on the edge of the image. For a 2-*D* image function $f(x,y)$, where $f(x,y)$ represents the intensity value of a certain point (x,y), a vector gradient $G_f(x, y)$ can be formed as [9]:

$$G_f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}. \tag{4.1}$$

The direction of this vector is toward the maximum rate of increase of the image function $f(x,y)$, while its magnitude is represented by:

$$|G| = [(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2]^{1/2}. \tag{4.2}$$

For discrete images, the coordinate system is chosen as shown in Figure 4.1, with $\partial f / \partial x$ pointing to the horizontal rightward direction and $\partial f / \partial y$ pointing to the vertical downward direction. Equation 4.2 can then be implemented digitally by:

$$|G| = [(f(j,k) - f(j+1,k))^2 + (f(j,k) - f(j,k+1))^2]^{1/2}, \tag{4.3}$$

which is commonly called 'Three-point Gradient'. This implementation is accurate, but is computationally expensive. If the absolute values of the terms inside the brackets under the square root are taken for the value of |G|, i. e.,

$$|G| \approx |f(j,k) - f(j+1,k)| + |f(j,k) - f(j,k+1)|, \tag{4.4}$$

computational advantages can be achieved.

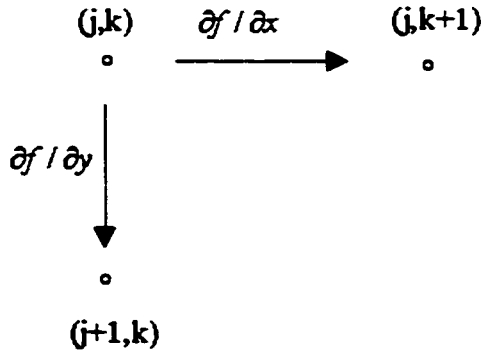

Figure 4.1 Digital Implementation of A Gradient Operator

By comparing the value $|G|$ of each pixel with the threshold value, a new value 1 or 0 will

be assigned to each pixel.

In order to implement the following Noise Reduction algorithm, the edge of a planar shape must be a closed contour. Therefore, when the edge detection algorithm is implemented, the threshold value is chosen as 1, which guarantees the edge will be a closed curve. After the gradient operation and subsequent thredholding operation have been performed, a closed contour is extracted.

4.4 Noise Reduction

A filling algorithm is developed and applied to reducing the noise presented in the binary image.

When the image section inside the contour (edge) is filled with 'white' and the rest image section is converted into 'black', an image without noise can be obtained. But if there is too much noise presented in the input image, a further step should be performed.

The following steps are conducted to remove the noise presented in the input image.

Step1: Randomly choose a point inside the contour as the starting point

Step 2: Process the adjacent 4 pixels of the starting point

Checking the adjacent 4 pixels of the starting point along vertical and horizontal directions, if the pixel is 'black' (The gray level is 0), change it to 'white' (The gray level is 1) and record its coordinates; otherwise, keep it 'white' and do not record its coordinates. This procedure is called 'Filling the 1^{st} layer'.

Step 3: Repeat the procedure until all the pixels inside the closed contour become 'white'

The same procedure should be conducted for the adjacent 4 pixels of each pixel, whose coordinates are recorded in step 2. If the pixel is 'black', change it to 'white' and record its coordinates until all the adjacent pixels around the recorded pixels become 'white'.

This procedure is called as: Filling the 2nd layer.

An example image is shown in Figure 4.2 with the chosen pixel inside the contour. Figure 4.3 shows the image after the first layer is filled and Figure 4.4 shows the image after the second layer is filled. All the Figures are color-inverted for display purpose.

The above procedure should be repeated until all the pixels inside the closed contour become 'white'. After the previous 3 steps are performed, the noise, which appears as white spots or lines inside the closed contour, will be removed.

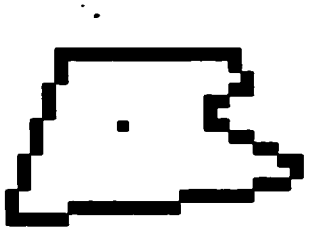

Figure 4.2 The Original Contour and The Starting Point

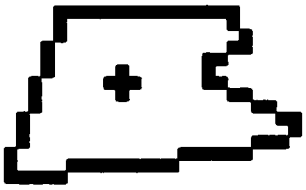

Figure 4.3 The Image After The First Layer Is Filled

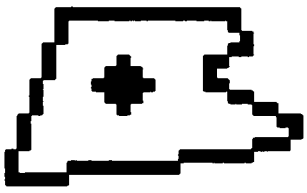

Figure 4.4 The Image After The Second Layer Is Filled

Step 4. Change the all pixels outside the contour to be 'black'

The noise outside the closed contour often consists of some white spots and lines. If all the pixels outside the contour are converted to be 'black', all the noise will be removed.

As long as Step 1 to Step 4 are finished, an ideal binary image is obtained. All the noise in the image will be removed.

4.5 Contour Tracing

The contour tracing algorithm can be described in terms of 'an observer', who 'walks' along the object boundary starting from any given point on the closed contour. The search proceeds from top to bottom and from left to right in the image space to find the first point on the boundary. If the point is found, then it will be taken as the starting point for the tracing process. From this initial point, searching starts in the directions 0, 1, 2, 3, 4, 5, 6, 7 respectively (Figure 4.5). If a point on the boundary is found, record its chain code, which is its direction, and take this point as a new starting point. The process is repeated until the contour is closed.

The following four steps are conducted to obtain the chain codes of the closed contour of a shape.

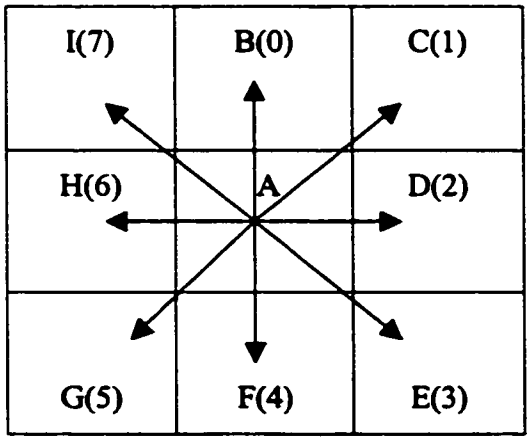

Figure 4.5 Numbering Scheme for 8-Connectivity Chain Code

Step 1: Choose the top-left pixel on the contour as the starting point

Suppose the chosen top-left pixel is point $A1$ as shown in Figure 4.6. The chain code of point $A1$ can be found by checking the intensity values of the pixels in directions 2, 3, 4 and 5 (Figure 4.5). If the intensity value of pixel D in direction 2 is 1, then the chain code of point $A1$ is 2. If not, continue to check the intensity value of pixel E in direction 3, and so on, until the chain code for Pixel $A1$ is found. Then a new point $A2$ is chosen as the new starting point as shown in Figure 4.7.

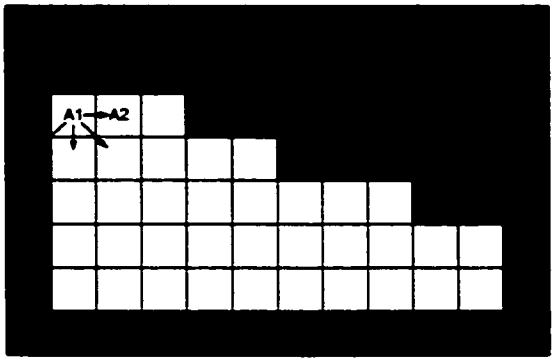

Figure 4.6 Diagram of Contour Tracing on Point $A1$

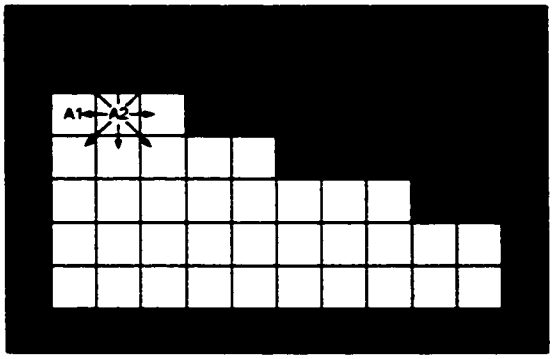

Figure 4.7 Diagram of Contour Tracing on Point $A2$

Step 2: Create an array *Binary*[8]

Now the new starting point is treated as point *A* in Figure 4.5. Checking the intensity value of pixel *B* first, the value *Binary*(0) can be set according to its intensity value. If it is 1, then set the value *Binary*(0) as 1; otherwise, set it as 0.

Following the same procedure: checking the intensity values of the pixels in the rest 7 directions, the values of *Binary*(2), *Binary*(3),..., and *Binary*(7) can be determined by the intensity values of pixels *C*, *D*, *E*, *F*, *G*, *H*, and *I*. For the image shown in Figure 4.7, the contents of the array *Binary*[8] for pixel *A*2 are:

Binary(0)=0,

Binary(1)=0 ,

Binary(2)=1 , ⟵ Transition

Binary(3)=1 ,

Binary(4)=1 ,

Binary(5)=1 ,

Binary(6)=1 , ⟵ Transition

Binary(7)=0 .

Step 3: Generate the chain code from the array *Binary*[8]

Now two positions in the array need to be found where the value transits. The chain code of pixel *A*2 to the next pixel is either 6 or 2 (the two positions in the array where the value transits. They represent the edge of the image). How to decide which is going to be the desired chain code? Since the contour is being traced clockwise, the rules as described follows can be used.

Subtracting the previous chain code from these two values and taking the absolute values of them, the following equations are obtained.

$$Difference1 = |6 - previous\_chain\_code| = |6 - 2| = 4,$$

$$Difference2 = |2 - previous\_chain\_code| = |2 - 2| = 0.$$

Because *Difference1* equals to 4 and *Difference2* does not equal to 4, the desired chain code is 2 instead of 6. The reason is that the absolute value of the difference between two opposite direction codes is 4. In this case, direction 6 is corresponding to pixel $A1$, which has already been processed in step 1 and has been recorded on the contour. Therefore, 2 is the desired chain code.

If the elements of the array *Binary*[8] have the following values:

Binary(1)=1,

Binary(2)=1 , ⟵ Transition

Binary(3)=0 ,

Binary(4)=0 ,

Binary(5)=0 ,

Binary(6)=1 , ⟵ Transition

Binary(7)=1 ,

Binary(8)=1 .

the two positions need to be found as the arrows pointing to. Then the rules described above can be used to determine which one is the chain code.

Step 4: Repeat the step 2 and step 3 until the tracing process reaches the original starting point

If the tracing process reaches the original starting point, terminate the tracing process; otherwise, go to step 2.

4.6 Pattern Recognition

Before discussing the pattern recognition, a simplified coding scheme as shown in Table 4.1 is introduced in this section. It is used to code the angles of the polygon, which represents the contour of a shape approximately.

Table 4.1 Angle Coding Scheme

Angle	Angle Code	Angle	Angle Code
0~22	0	180~206	8
22~44	1	206~228	9
44~66	2	228~250	A
66~88	3	250~272	B
88~110	4	272~294	C
110~132	5	294~316	D
132~154	6	316~338	E
154~180	7	338~360	F

After coding the angles, the corresponding angle string and the shape number can be obtained. The shape number of the template image was generated and saved before the input image is processed. And the shape number of the input image can also be found. Comparing it with that of the template, if they are not equal, they do not belong to the same pattern; otherwise, compare the side lengths of the corresponding polygons to see if the two patterns match or not. When comparing the side lengths of the polygons, the reference point should be the vertex, which has the minimum angle.

For the polygon, all the angles of which are the same, a new reference will be used, which is the side of the polygon with the minimum side length.

As we know that the resolution of an image is limited, therefore, when an image is rotated over a certain degree, the rotated image may not be exactly the same as the original one. And the δ function may change too. If the δ value of a non-critical point, which has the local maximum value, becomes a little bit greater than the specified threshold after the image rotation, this point may become a critical point after the image is rotated. While in some other cases, the δ value of a critical point may become a little

bit less than the threshold and can not be treated as a critical point anymore after the image is rotated. Surely, it will generate different shape numbers after image rotation and result in pattern recognition failure. In order to make the algorithm more robust, the following scheme is figured out to solve the problem.

A region is defined for the δ function, whose maximum and minimum δ values are 120% and 80% of the original threshold respectively. If the δ value of a critical point falls into this region, this point may be treated as either a critical point or not. For example, there are two such kind of critical points as shown in Example 4 in Chapter 4.10.4, they are treated as special critical points. For these two special critical points, both critical points can be kept, or one critical point is removed, or the other is removed, or both are removed. Therefore, there are 4 different shape numbers for the same input image. As long as one of the shape numbers is the same as that of the template and the side lengths are also approximately the same, they belong to the same pattern. By adding this new procedure, the method becomes more accurate and robust to cover more cases.

If the input image and the template belong to the same pattern, then complete the following steps to calculate the deviations or the rotation angle.

4.7 Deviation Calculation

Since a polygon can be used to approximate the contour of a certain shape, the average of critical point coordinates can also be used to represent the position of the contour. The deviation along X direction will be the difference between the averages of the coordinates of the critical points of the input image and the template along X-direction. And a similar formula can be used for the calculation of the deviation along Y direction.

Deviation-X=Average of Critical Point Coordinates along X-direction (input image)

- Average of Critical Point Coordinates along X-direction (Template) (4.5)

Deviation-Y=Average of Critical Point Coordinates along Y-direction (input image)

- Average of Critical Point Coordinates along Y-direction (Template) (4.6)

4.8 Rotation Angle Calculation

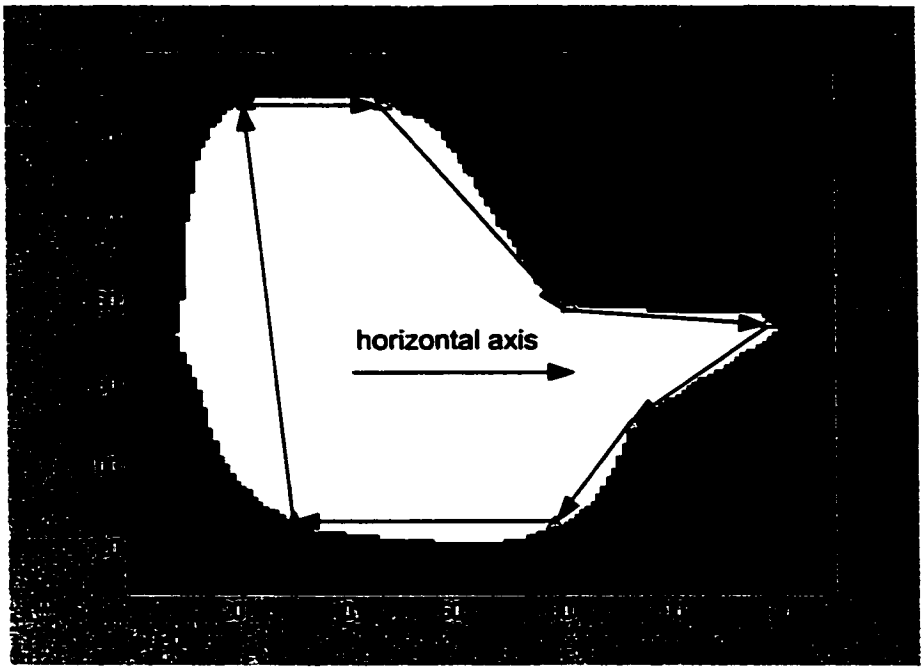

Figure 4.8 Rotation Angle Calculation

The vectors are defined clockwise as the connections of every two neighboring critical points. The angles between the vectors and the horizontal axis can be found. The angle will range from 0° to 360°. By calculating the differences between every 2 corresponding angles of the input image and the template, and taking the average of them, the rotation angle can be obtained. When calculating the rotation angle, the reference point is the same as that in Chapter 4.6.

4.9 Algorithm Design Based on The Directional Flow-Change Method

8 steps are conducted to implement the algorithm, the flow chart of which is shown in Figure 4.9. In this Figure, '*Flag1*' is used to indicate if the deviation calculation or the rotation angle calculation is needed.

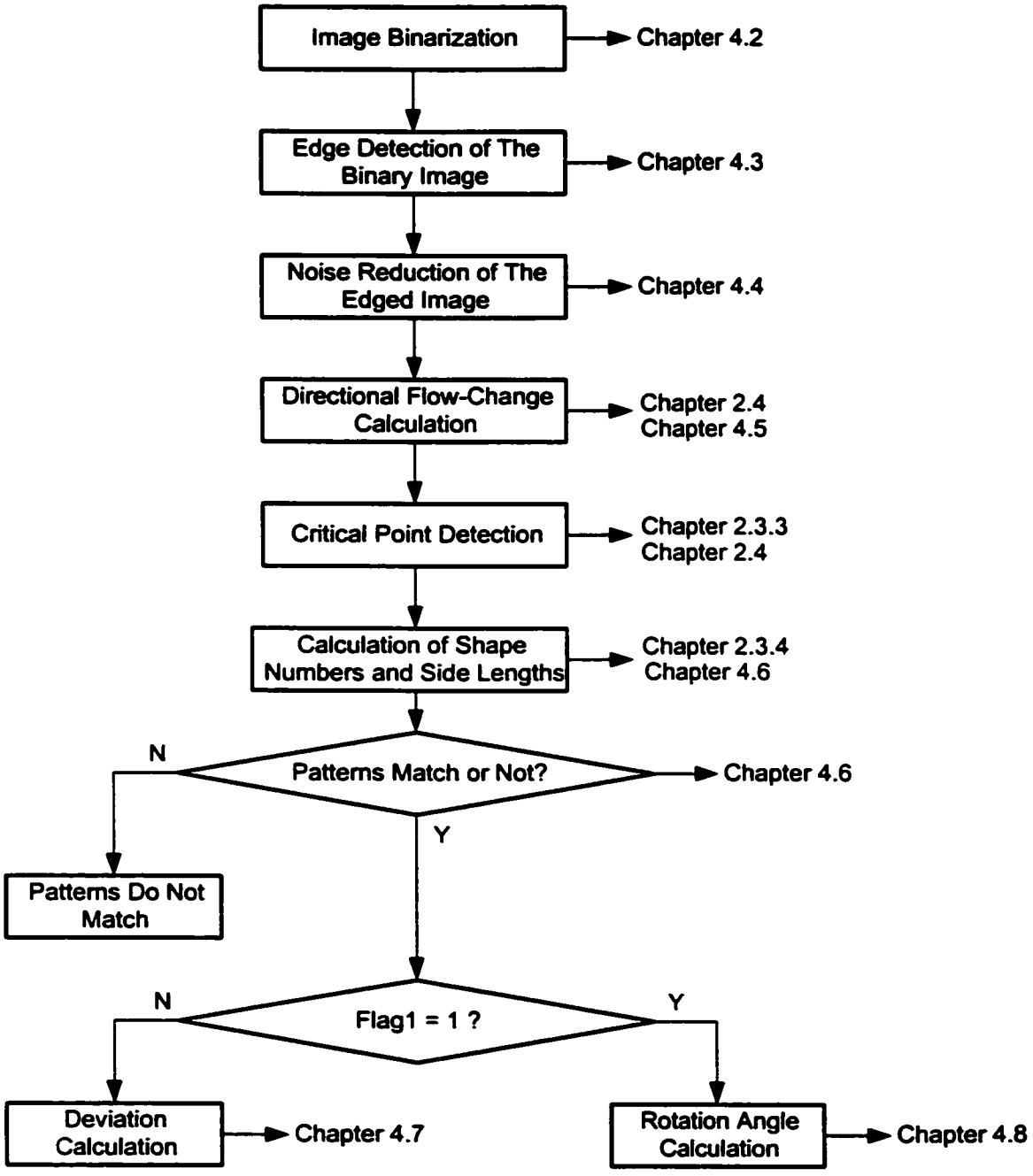

Figure 4.9 Flow Chart of The Algorithm Based on The Directional Flow-Change Method

Step 1: Get the binary image of the input image.

Step 2: Get the edge of the binary image.

Step 3: Reduce the noise of the edged image.

Step 4: Calculate the directional flow-changes of the all pixels on the contour.

Step 5: Detect the critical points of the closed contour.

Step 6: Calculate the angles of the polygon formed by connecting every two neighboring critical points. Convert the sequence of angles into a sequence of angle codes, and then the shape numbers can be found.

Step 7: Compare the shape numbers and the side lengths of the polygons to see if the two patterns do match or not. If yes, go to step 8; otherwise, terminate the algorithm.

Step 8: Calculate the position deviations of the input image along X and Y directions compared with the template or calculate the rotation angle of the input image compared with the template.

4.10 Examples

In order to test and illustrate the performance of the algorithm, some examples are given.

4.10.1 Example 1

Typical figures are presented as well as the corresponding steps.

Step 1: Get the binary image of the input image

Figure 4.10 is the input image with low contrast.

Figure 4.10 Input Image

Figure 4.11 is the histogram $H(z)$ of the input image.

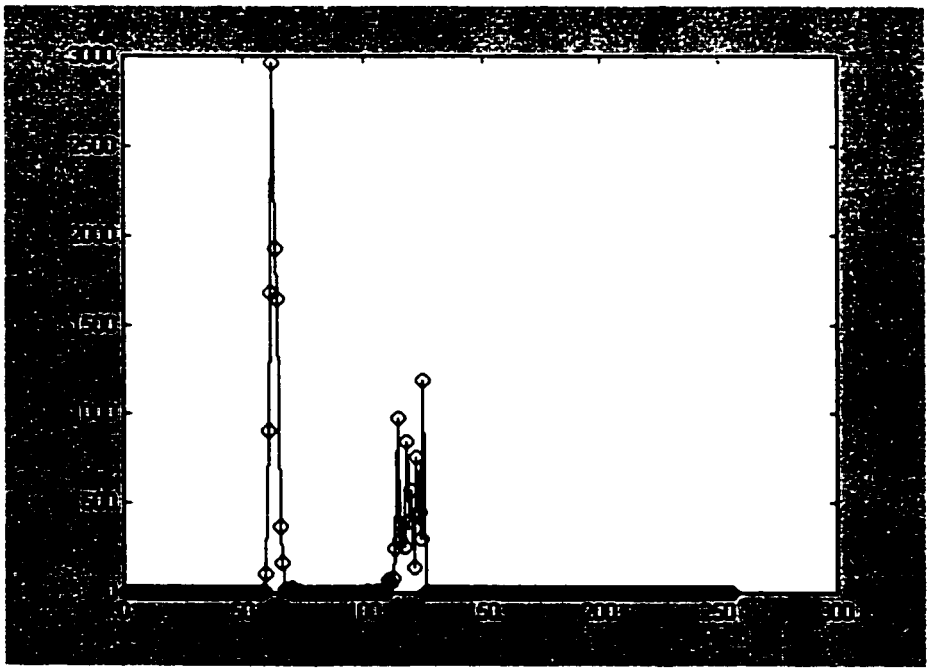

Figure 4.11 Histogram of The Input Image

In order to convert a gray level image into a binary image, the threshold value needs to be found first. A simple method is presented as follows for this purpose.

The two points with the local maximum values, should be found first. Then the midpoint of the two points can be obtained easily. The corresponding gray level of the midpoint will be treated as the threshold value, which is 96 in this example.

By applying the threshold value 96, a binary image can be obtained as shown in Figure 4.12.

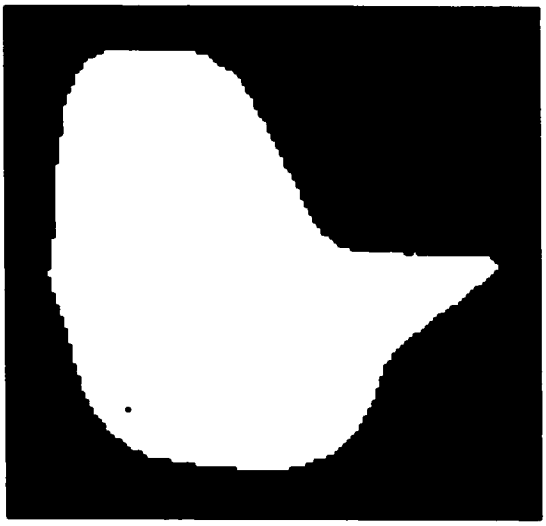

Figure 4.12 Binary Image

In order to test the noise reduction algorithm, a binary image with noise (Figure 4.13) is generated by adding some noise in the previous binary image (Figure 4.12).

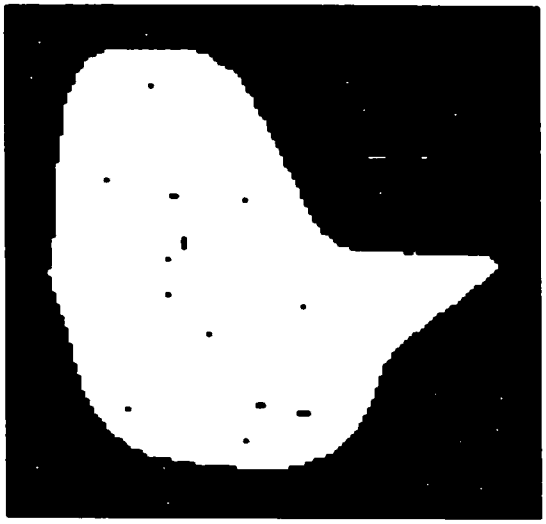

Figure 4.13 Binary Image with Some Noise

Step 2: Get the edge of the binary image

The edged image of Figure 4.13 is shown in Figure 4.14.

Figure 4.14 The Edged Image

Step 3: Reduce the noise of the edged image

By applying the noise reduction algorithm, Figure 4.15 can be obtained.

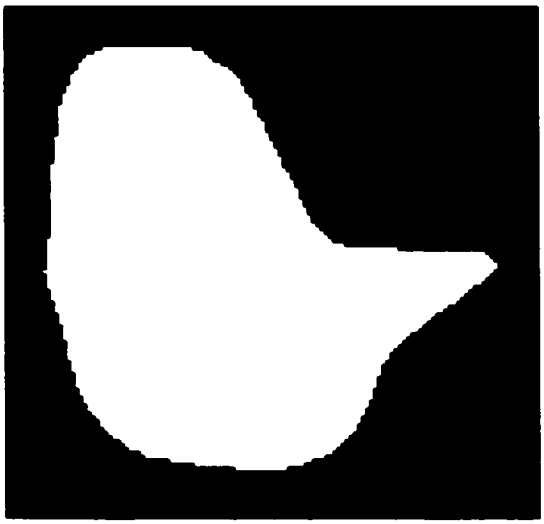

Figure 4.15 The Image after Noise Reduction

It can be seen that all the noise presented in the edged image is removed.

Step 4: Calculate the directional folw-changes

The following parameters are chosen first:

$$\gamma = 0.3, \quad \theta = 0.9, \quad t = 0.8.$$

The threshold value for detecting the critical points can be obtained, which is 8 in this example. The δ Function of the closed contour is shown in Figure 4.16.

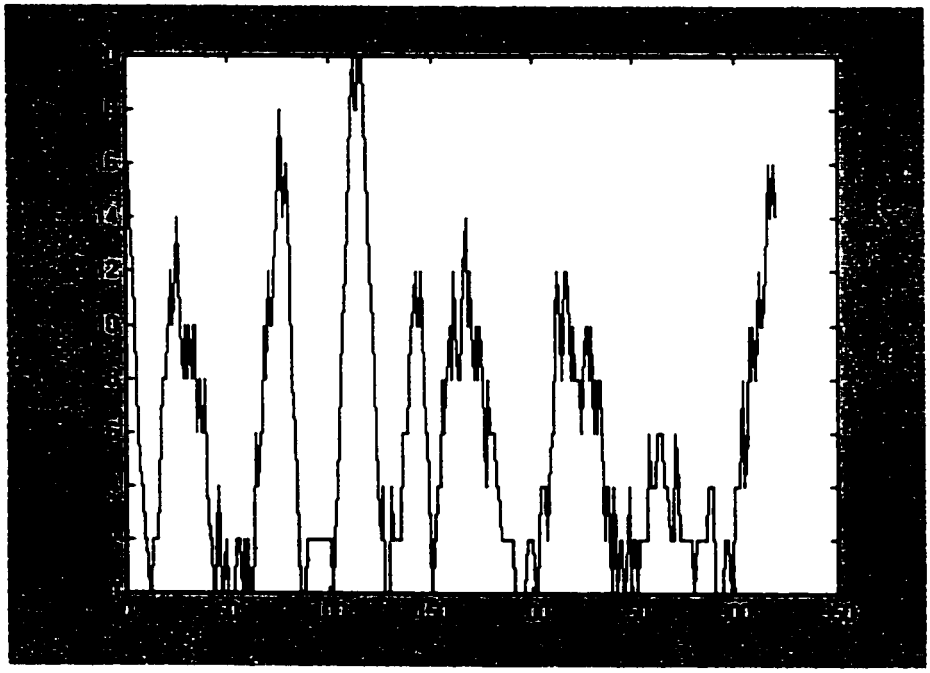

Figure 4.16 δ Function of The Closed Contour

When the δ function is processed through a low pass filter, a smoother δ function is obtained as shown in Figure 4.17.

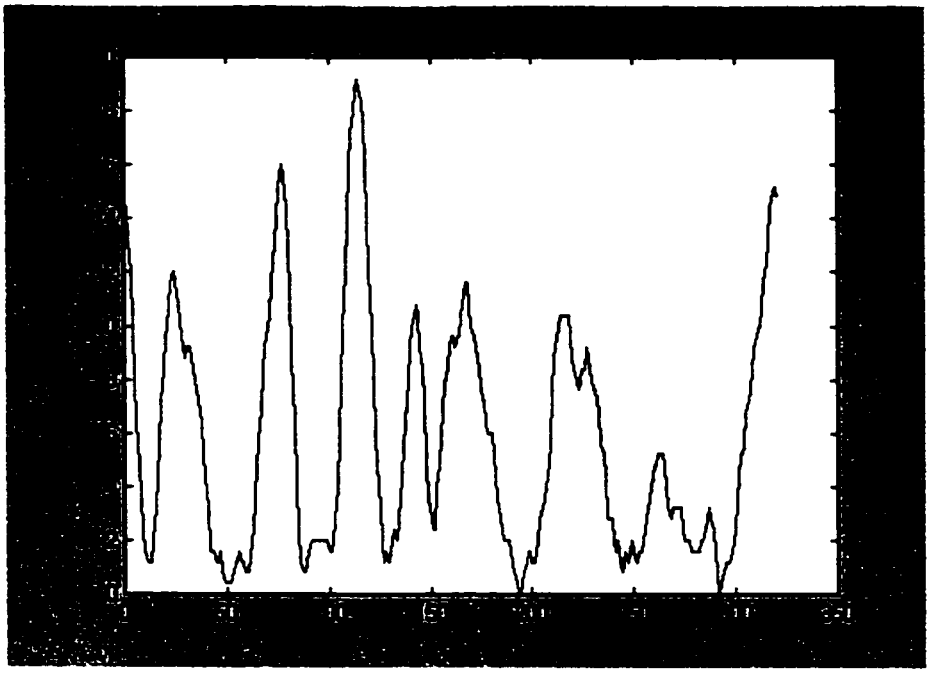

Figure 4.17 The δ Function after A Filter

Step 5: Detect the critical Points

The critical points found by the algorithm are:

24, 77, 114, 143, 168, 216, 319,

which are marked by ' Δ ' in Figure 4.18.

Step 6: Find the Angle String and the Shape Number

A polygon is formed by connecting every two neighboring critical points (Figure 4.18). Starting with angle $\alpha 1$, a sequence of the angles of the polygon are obtained as:

124.8192 229.0075 51.1961 192.9099 120.8732 95.7709 85.5145.

The related angle string is:

5 A 2 8 5 4 3.

And the Shape Number is 285435A.

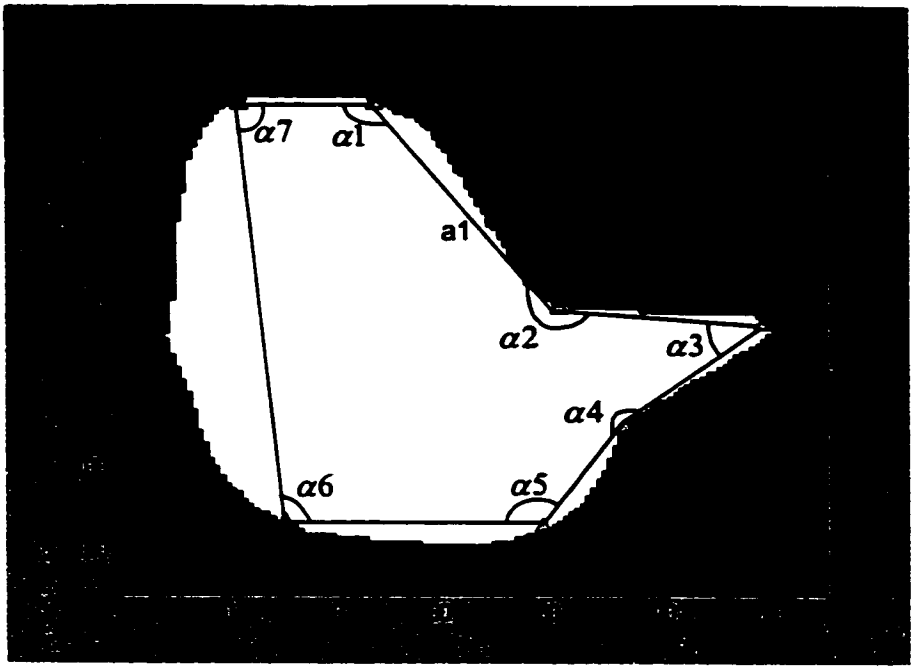

Figure 4.18 Critical Points Found by The Algorithm

Step 7: Pattern Recognition

The shape number of the input image needs to be compared with that of the template first. If they are equal, then compare the side lengths of the polygons to see if the two patterns do match or not. Starting with the side $a1$ clockwise, the side lengths of the polygon are:

59.6406 37.2156 35.3553 28.3019 48.0104 101.3163 25.0000.

There is a parameter '*Times*' used to indicate which vertex has the minimum angle. For example, when *Times*=3, it means that the 3rd angle $\alpha 3$ is the minimum angle, which will be the reference point for side length comparison and rotation angle calculation.

If the corresponding side lengths are also approximately the same, beside the same shape number, the input image and the template belong to the same pattern.

Step 8: Deviation Calculation or Rotation Angle Calculation

1. Deviation Calculation

The averages of critical point coordinates along both *X*-direction and *Y*-direction can be found, which are 68 (*pixels*) and 66.7143 (*pixels*) respectively. If there is a template, it is very easy to find the deviations.

2. Rotation Angle Calculation

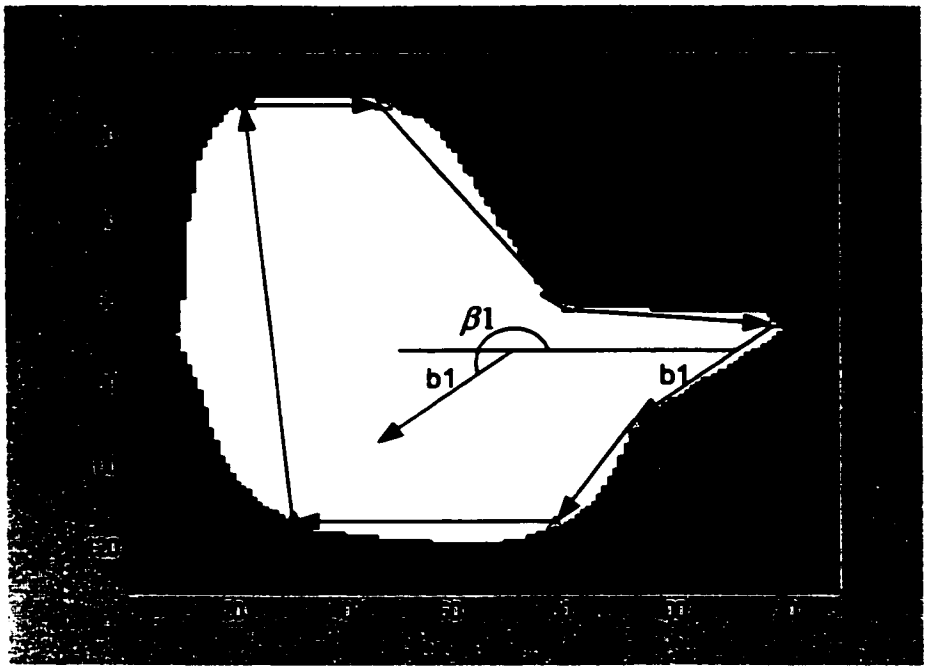

Figure 4.19 Rotation Angle Calculation

Starting with angle $\beta 1$, which is the angle between the vector $b1$ and the horizontal axis, the angles found clockwise between the vectors and the horizontal axis are:

225.0003 237.9949 178.8061 94.5286 0 304.7560 353.8298.

If there is a template, it is very easy to find the rotation angle.

4.10.2 Example 2

Figure 4.20 shows another example image, which has a different shape. The critical points are marked by ' Δ ' on this figure.

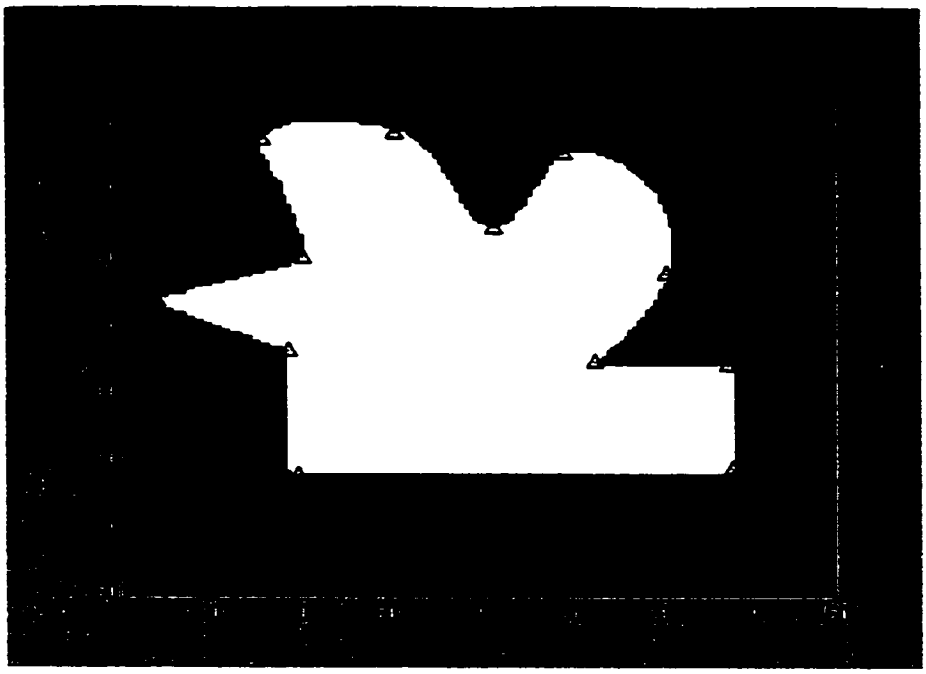

Figure 4.20 Template Image

Figure 4.21 is an input image with deviation. Figure 4.22 is an image with counterclockwise rotation and Figure 4.23 with clockwise rotation.

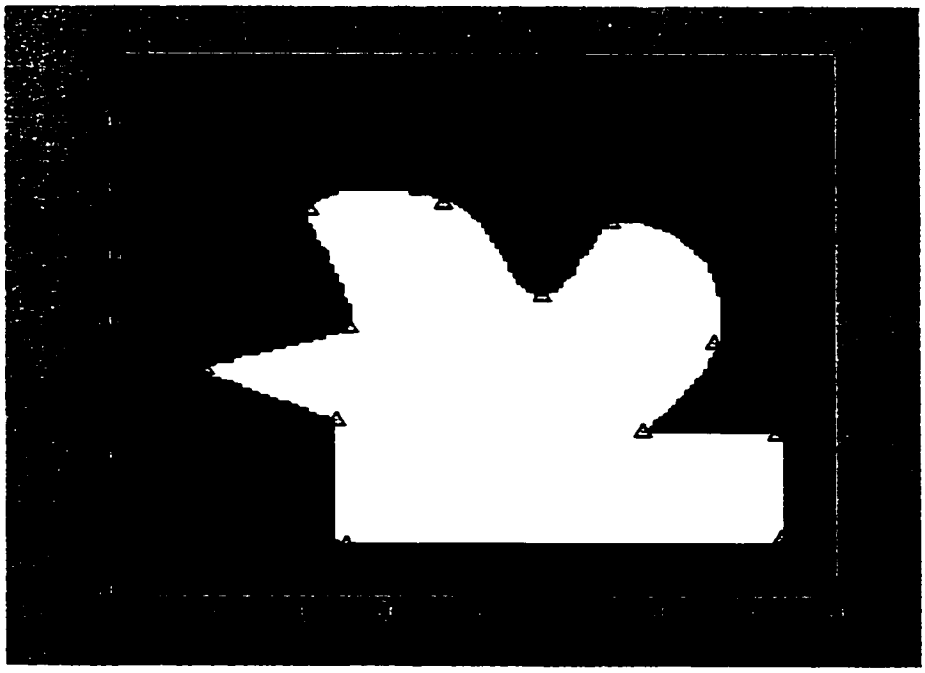

Figure 4.21 Input Image with Deviation

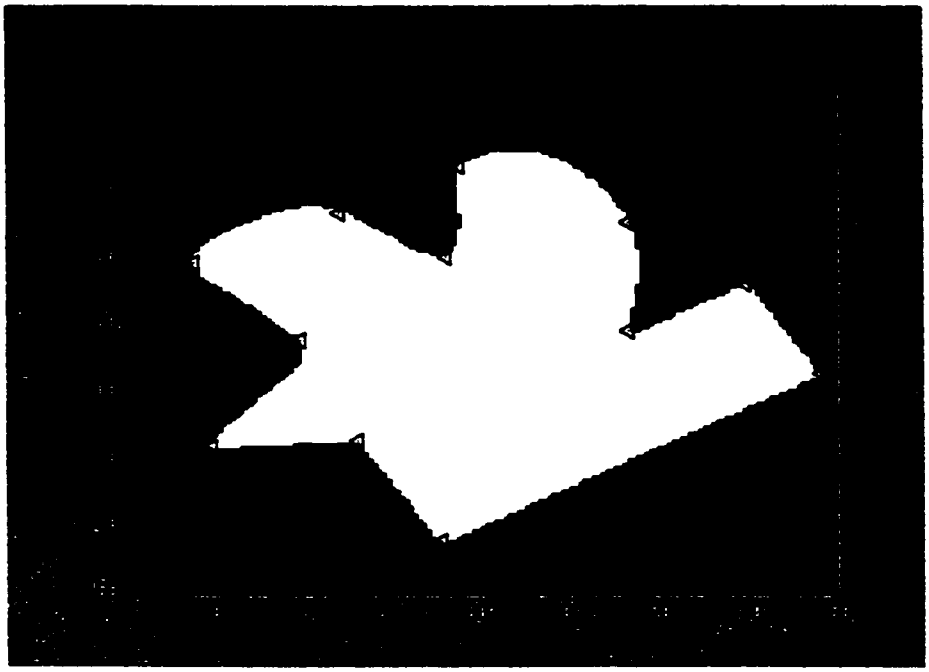

Figure 4.22 Input Image with Counterclockwise Rotation

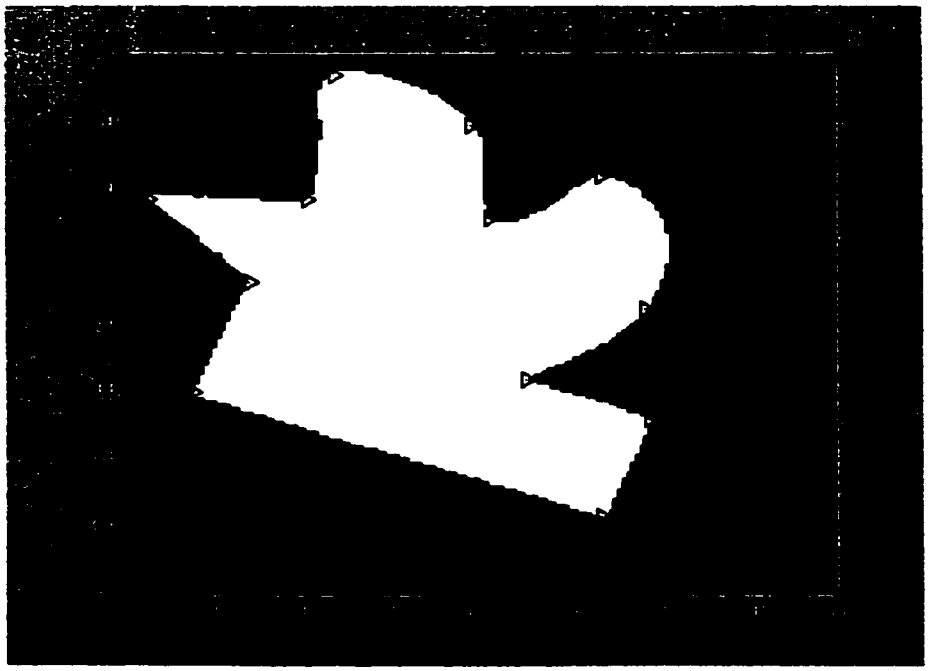

Figure 4.23 Input Image with Clockwise Rotation

Table 4.2 shows the result by listing the angle strings, shape numbers, numbers of critical points, and so on. The deviations and rotation angle found by the algorithm are also listed in this table.

Table 4.2 Result of Example 2

	Template	**Input Image with Deviations**	**Input Image with Counterclockwise Rotation**	**Input Image with Clockwise Rotation**
Angle String	5 C 3 5 D 4 4 4 A 2 B 3	5 C 3 5 D 4 4 4 A 2 B 3	5 D 4 4 4 A 2 B 3 5 C 3	5 C 3 5 D 4 4 4 A 2 B 3
Shape Number	2B35C35D444A	2B35C35D444A	2B35C35D444A	2B35C35D444A
Number of Critical Points	12	12	12	12
Side Lengths	35 36 30 35 27 42 30 30 31 98 37 33	35 27 42 30 30 31 98 37 33 35 36 30	32 29 31 98 35 33 37 33 36 27 27 41	28 28 41 34 30 31 99 35 33 36 37 34
***Angles**	21 104 3 308 53 302 238 358 271 181 93 153	21 104 3 308 53 302 238 358 271 181 93 153	57 136 24 331 83 335 270 28 302 210 122 183	0 80 334 278 27 284 217 336 249 158 68 132
***Times**	10	10	7	10
Average-*X* (*pixels*)	75.8333	86.8333		
Average-*Y* (*pixels*)	70.4167	91.4167		
Deviation-*X* (*pixels*)		11		
Deviation-*Y* (*pixels*)		21		
Real Deviation-*X* (*pixels*)		11		
Real Deviation-*Y* (*pixels*)		21		
Rotation Angle (°)			29.6734	-23.5627
Real Rotation Angle (°)			30	-22

*Angles: Indicate the angles between the vectors and the horizontal axis.

*Times: Indicates which vertex has the minimum angle.

From Table 4.2, it can be seen that all the shape numbers are the same and the

corresponding side lengths are also approximately the same. Hence all of the 4 shapes belong to the same pattern.

4.10.3 Example 3 (Special Case 1)

Figure 4.24 and Figure 4.25 have a different story. All of the angles of the two polygons are the same.

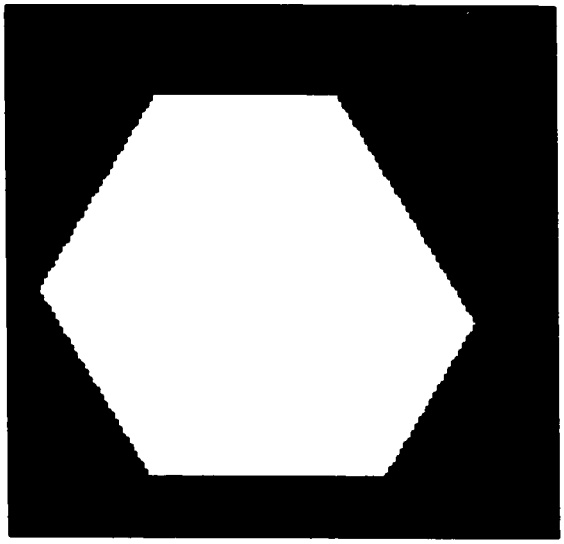

Figure 4.24 A Special Shape (Template)

All the angles of the both polygons are 120°. The corresponding angle string is:

5 5 5 5 5 5.

Therefore the shape number is 555555.

A different reference has to be used for side length comparison and rotation angle calculation. The side of the polygon, which has the minimum length, is treated as the reference side. This is a different case compared with example 1 and 2.

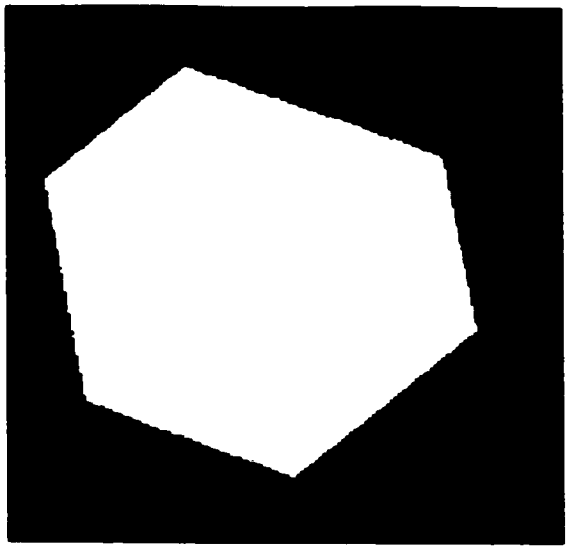

Figure 4.25 A Special Shape (Input Image)

4.10.4 Example 4 (Special Case 2)

According to Chapter 4.6, some special critical points may fall into the defined region. An example regarding this problem is given as follows.

Figure 4.26 is a template image and Figure 4.27 is its δ function.

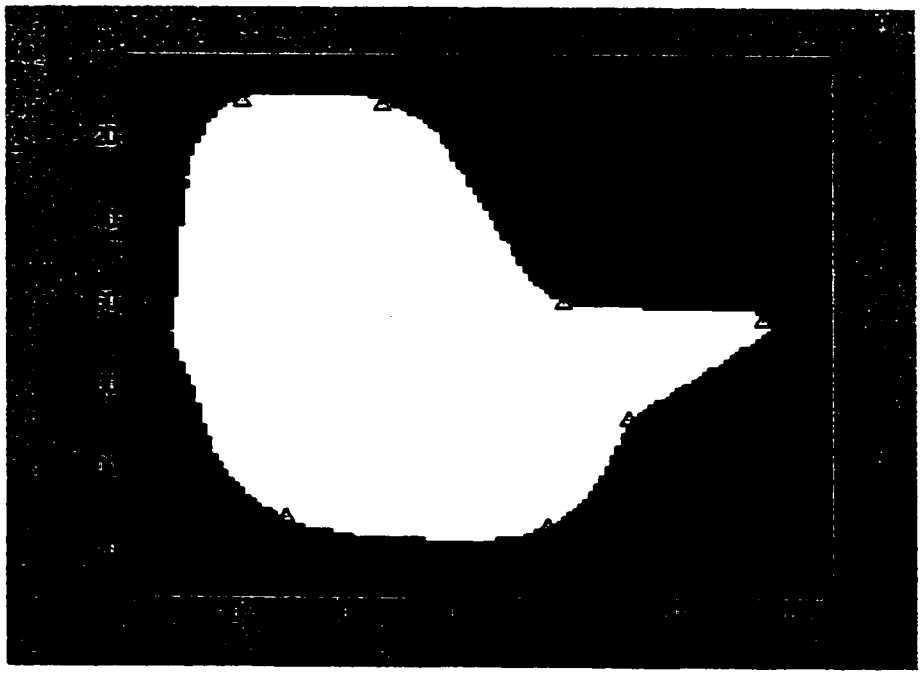

Figure 4.26 A Template

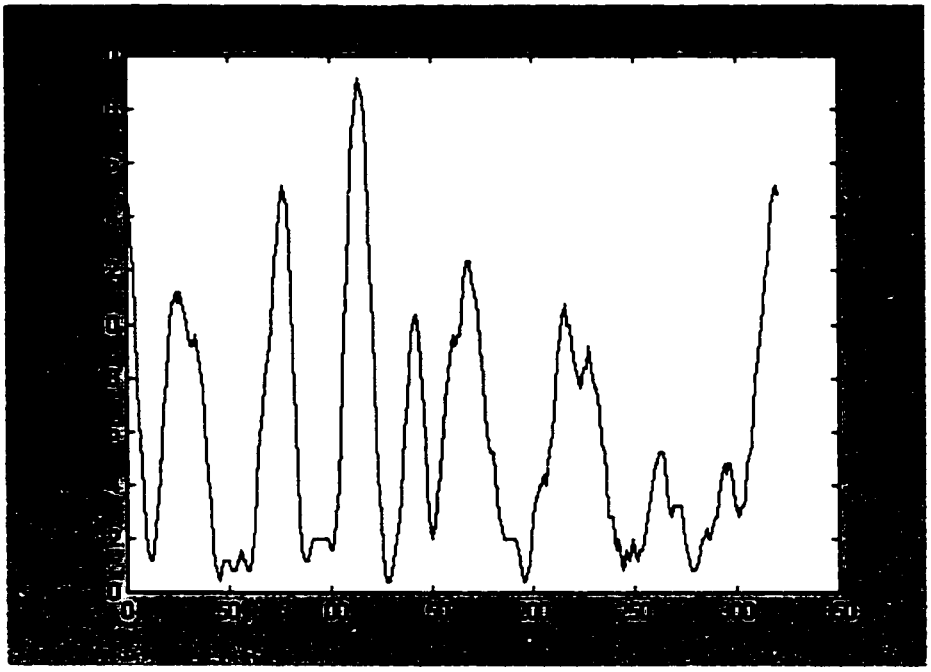

Figure 4.27 δ Function of The Template

Figure 4.28 is the rotated image and Figure 4.29 is its δ function.

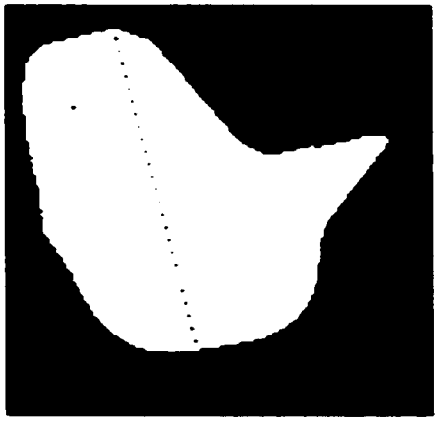

Figure 4.28 Input Image with Rotation

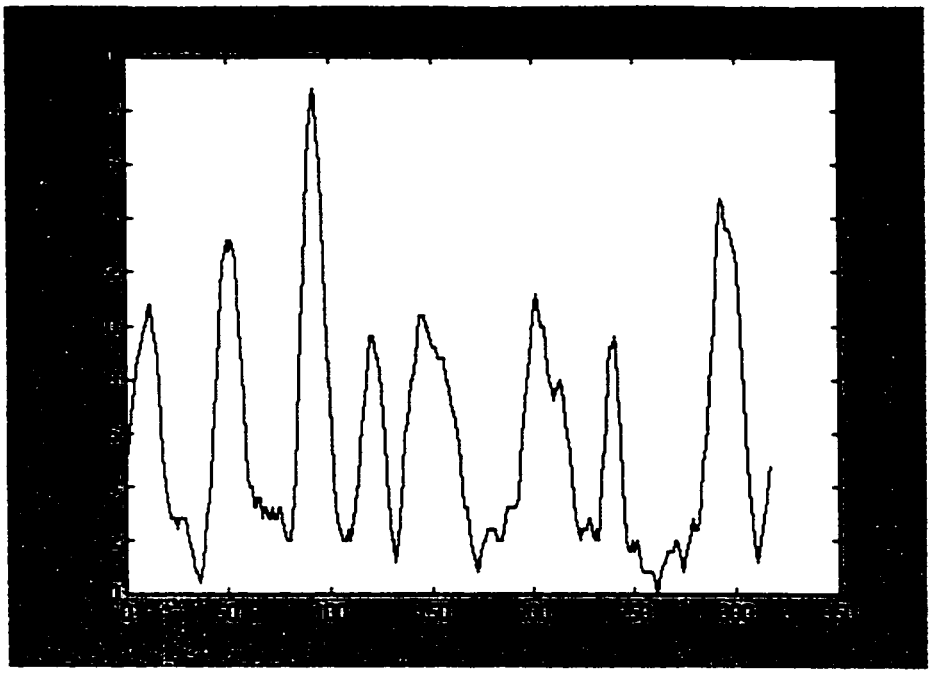

Figure 4.29 δ Function of The Rotated Image

Putting the 2 δ functions together as shown in Figure 4.30, some difference between them can be observed. No special Critical point falls into the specified region for the template. But in the rotated image, two special critical points fall into the defined region. In such case, the both critical points can be kept (case 1 as shown in Figure 4.31), or the first one is removed (case 2 as shown in Figure 4.32), or the second one is removed (case 3 as shown in Figure 4.33), or the both are removed (case 4 as shown in Figure 4.34). So 4 different shape numbers will be generated. But only in case 3, the shape number is the same with that of the template, and the corresponding side lengths of the polygon are also approximately the same with those of the template. Therefore, the input image and the template belong to the same pattern. Table 4.3 lists the result.

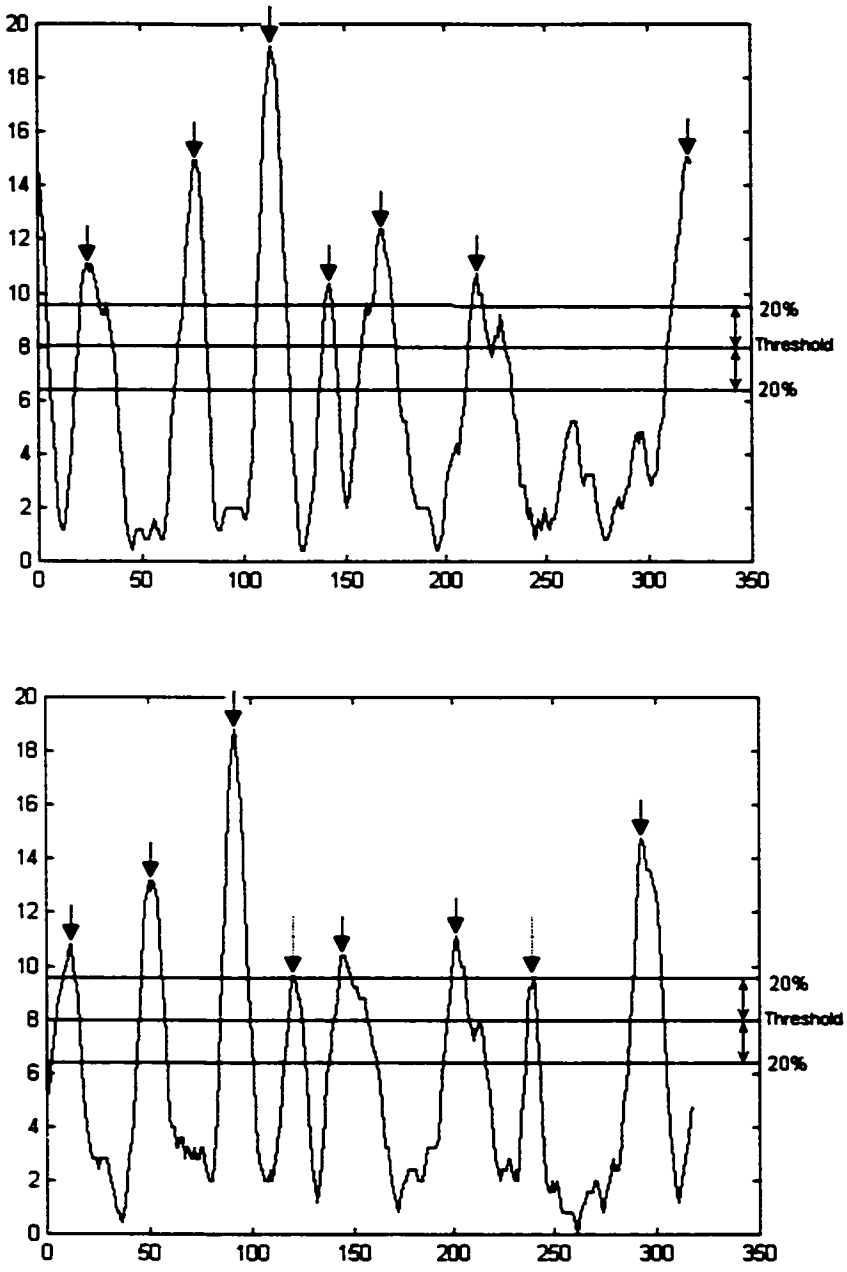

Figure 4.30 δ Functions of The Unrotated Image (Top) and The Rotated Image (Bottom)

Table 4.3 Result of Example 4

	Template	Input Image (Rotated Image)			
		Case 1	Case 2	Case 3	Case 4
*Flag	0000000	00100100			
Number of Critical Points	7	8	7	7	6
Shape Number	285435A	2855645A	255645A	285435A	25435A
Critical Points	25 77 114 142 168 216 319	51 92 121 145 201 240 293 12	51 92 145 201 240 293 12	51 92 121 145 201 293 12	51 92 145 201 293 12
Rotation Angle (°)				13.5690	
Real Rotation Angle (°)				14	

*Flag: Each bit in the *Flag* indicates if the corresponding critical point falls into the specified region or not (1 means 'yes' and 0 means 'no'.

Figure 4.31, 4.32, 4.33 and 4.34 are the corresponding diagrams for the 4 cases.

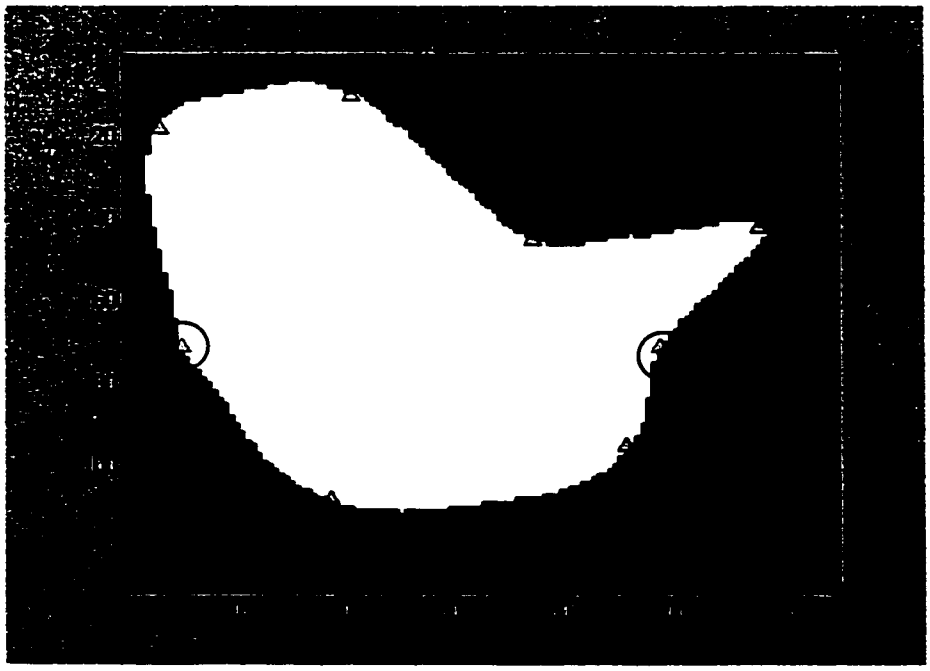

Figure 4.31 Rotated Image with The Both Special Critical Points

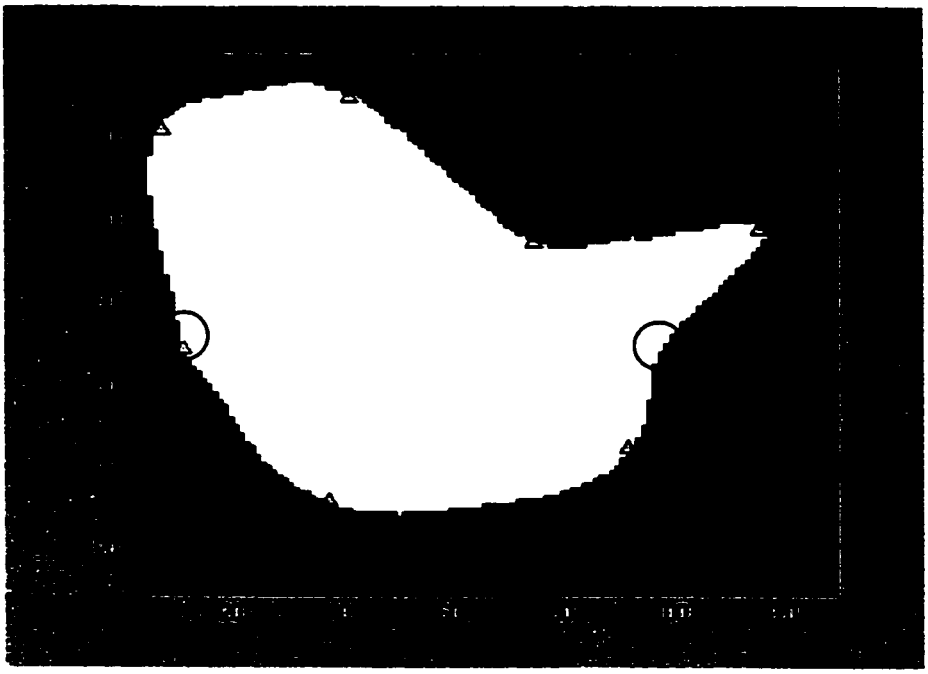

Figure 4.32 Rotated Image with The 2nd Special Critical Point

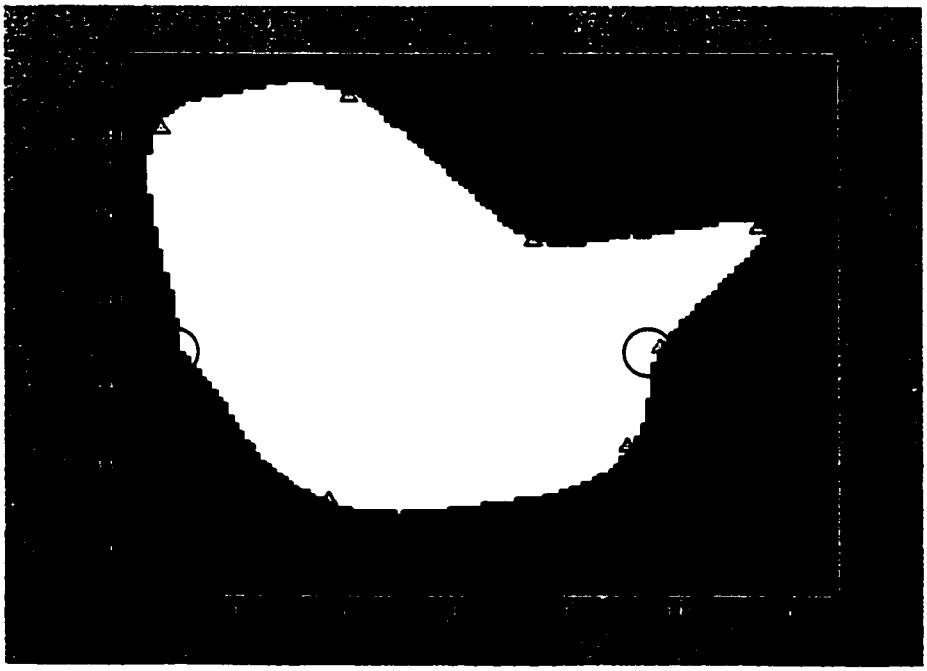

Figure 4.33 Rotated Image with The 1st Special Critical Point

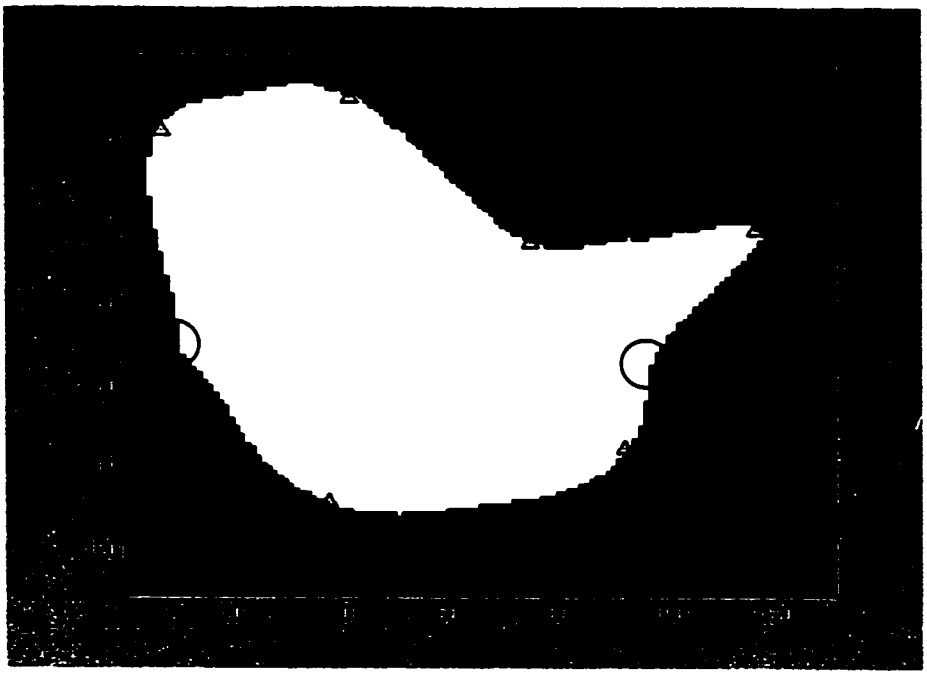

Figure 4.34 Rotated Image with The Both Special Critical Points Removed

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The research work in this thesis is aimed at developing the algorithms, which can be used to find the pattern of the shape of a target, as well as the position deviation and the rotation angle of the target in the input image compared with the reference position of the template. The ultimate goal of this research is to develop smart vision sensors for the existing problems in manufacturing industry, such as the problem existing in the assembly line for automobile door handle escutcheons.

The main contributions of this thesis are summarized as follows:

1. The algorithm based on the correlation analysis method is developed and has been applied to the design of a smart vision sensor for finding the 2-*D* deviation of the target in the input image.

2. The algorithm based on the Directional Flow-Change Method is developed and is being applied to the design of a smart vision sensor for identifying the pattern of the shape of a target, as well as the position deviation and the rotation angle.

3. The Directional Flow-Change method is modified with two revised definitions and 3 supplemental procedures, which make the algorithm for pattern recognition more complete and more accurate.

The pros and cons of these two algorithms are summarized as follows:

1. The Algorithm Based on The Correlation Method – Method 1

It is an efficient and easy method to apply with high noise rejection and reliability. The

notable drawback is that it takes much computing time to obtain the results because of large amount of correlation calculation involved. Besides, it is also not powerful enough to identify the pattern of a planar shape and to calculate the rotation angle between the input image and the template using this method.

2. The Algorithm Based on The Directional Flow-Change Method – Method 2

The algorithm based on the directional flow-change method has good performance in efficiency, accuracy and effectiveness. Not only can it identify the pattern of a target quickly, but also it can calculate the position deviation and the rotation angle accurately.

But this algorithm is much more complex than Method 1 and it carries lower noise rejection. The input image has to be processed to generate an ideal one with greatly reduced noise.

5.2 Comments on Potential Applications

Method 1 is better than Method 2 for single-pattern applications, because the pattern recognition is unnecessary. Besides, Method 1 is also simpler to implement. Some optimal computing methods can be conducted to reduce the computing time required to run the program. For example, the *FFT* algorithm together with the correlation theorem can be used to optimize the computing process. Another way is to change the gray level image to binary image before the correlation operation is performed. Because the data in a binary image is either 0 or 1, the correlation result will be the summation of 1's based on the basic and timesaving operations: $0 \times 1 = 0$, $1 \times 1 = 1$. Therefore, the computing time required can be reduced.

Method 2 is much better for multi-pattern applications.

A smart vision sensor can be designed based on either Method 1 or Method 2 depending on the individual engineering problem. Because it is the software solution that is mainly adopted for the design, it is very flexible to update the sensor functions and to augment the sensor into a closed-loop system design in practice.

5.3 Recommendations for Future Work

Further studies and work are recommended as follows:

1. Calculate The Deviation and The Rotation Angle Simultaneously

In this thesis, the algorithm based on the directional flow-change method is able to calculate both the deviation and the rotation angle, but not at the same time. The next step will be calculating them simultaneously.

2. Robust Algorithm Development

For future projects and research, the coding scheme described in Chapter 2.3 is recommended, which will be more robust and adaptable.

3. Processing of Color Images

In this thesis, all the research work targets on identifying the pattern and the position of a planar shape, which is usually captured in a binary or a gray level image. It is recommended that the research on pattern recognition and position measurement for color images be continued as well.

REFERENCES

[1] G. Appenzeller, P. Weckesser, R. Dillmann, "Active Parameter Control for the Low Level Vision System of A Mobile Robot", Proc. IROS, pp. 1256-1263, 1996.

[2] Carlo Arcelli, Giuliana. Ramella, "Finding Contour-Based Abstractions of Planar Patterns", Pattern Recognition, Vol. 26, No. 10, pp1563-1577, 1993.

[3] J. Austin, A. Turner, M. Turner, K. Lees, "Chemical Structure Matching Using Correlation Matrix Memories", IEE Conference Publication, Vol. 2, No. 470, pp. 619-624, Sep. 7-Sep. 10, 1999.

[4] Antonio Bandera, Cristina Urdiales, Fabian Arrebola, Francisco Sandoval, "On-Line Unsupervised Planar Shape Recognition Based on Curvature Functions", IECON Proceedings (Industrial Electronics Conference), Vol. 3, IEEE Comput. Soc. pp. 1268-1272, Aug. 31-Sep. 4, 1998.

[5] Orit Baruch, Murray H. Loew, "Segmentation of Two-Dimensional Boundaries Using Chain-code", Pattern Recognition, Vol. 21, No. 6, pp. 581-589, 1988.

[6] Rikard Berthilsson, Anders Heyden, "Recognition of Planar Objects Using the Density of Affine Shape", Computer Vision and Image Understanding, Vol. 76, No. 2, Acad. Press Inc., pp. 135-145, 1999.

[7] Jennifer Blais, Verlyn Fischer, Moalem Yoel, Matthew Saunders, "Correlation of Digital Image Metrics to Production ADC Matching Performance", IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop, pp. 86-92, Sep. 23-25, 1998.

[8] Giorgio Bonmassar, Eric L. Schwartz, "Improved cross-correlation for template matching on the Laplacian pyramid", Pattern Recognition Letters, Vol. 19, No. 8, Elsevier Science B. V., pp. 765-770, Jun. 1998.

[9] Sing-Tze Bow, "Pattern Recognition and Image Preprocessing", New York, M. Dekker, c1992.

[10] Gail A. Carpenter, Stephen Grossberg, "Pattern Recognition by Self-Organizing Neural Networks", Cambridge, Mass., MIT Press, c1991.

[11] Larry S. Davis, "Understanding Shape: Angles and Sides", IEEE Trans. Comput., Vol. 26, No. 3, pp. 236-242, 1977.

[12] Wai-Chi Fang, "A System-On-A-Chip Design of A Low-Power Smart Vision System", Proc. 1998 IEEE Workshop on Signal Processing Systems: Design and Implementation, pp. 63-72, 1998.

[13] Martin A. Fischler, Robert C. Bolles, "Perceptual Organization and Curve Partitioning", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 1, pp. 100-105, 1986.

[14] H. Freeman, "Computer Processing of Line Drawing Images", Computing Surveys, Vol. 6, No. 1, pp. 57-98, 1974.

[15] Herbert Freeman, Larry S. Davis, "A Corner-Finding Algorithm for Chain-Coded Curves", IEEE Transactions on Computers, C-26: pp. 297-303, Mar. 1977.

[16] Alan M. N. Fu, Hong Yan, "Effective Classification of Planar Shapes Based on Curve Segment Properties", Pattern Recognition Letters, Vol. 18, pp. 55-61, 1997.

[17] Alan M. N. Fu, Hong Yan, Kai Huang, "A Curve Bend Function Based Method to Characterize Contour Shapes", Pattern Recognition, Vol. 30, No. 10, pp. 1661-1671, 1997.

[18] Hongmei Gao, Xiang Chen, "Theoretic Design of A Smart Vision Sensor", 2001 IEEE Canadian Conference on Electrical and Computer Engineering, Toronto, May 13-16, 2001.

[19] Hongmei Gao, Xiang Chen, "Application of Directional Flow-Change Method to Design of A Smart Vision Sensor", was accepted to present in the IASTED International Conference on Robotics and Applications (RA2001), Tampa, USA, Nov. 19-22, 2001.

[20] M. Gokstorp, "Smart Vision System for Applied Image Processing", SPIE, Vol.

3101, pp. 276-282, 1997.

[21] R. C. Gonzalez, R. E. Woods, "Digital Image Processing", Addison-Wesley, Reading, MA, 1992.

[22] P. W. Huang, S. K. Dai, P. L. Lin, "Planar Shape Recognition by Directional Flow-Change Method", Pattern Recognition Letters, Vol. 20, No. 2, pp. 163-170, 1999.

[23] G. K. Knopf, S. Zhu, "Qualitative Detection of Object Movement by Mobile Camera Systems", Proc. ISIAC Second International Symposium on Intelligent Automation and Control, pp. 108.1-108.6, 1998.

[24] C. Koch, "Implementing Early Vision Algorithms in Analog Hardware-An Overview", SPIE, Vol. 1473, pp. 2-16, 1991.

[25] K. W. Kwok, K. C. Lo, "Recognition of Curved Shapes Using Geometric Invariants", International Conference on Signal Processing Proceedings, ICSP Vol. 2, pp. 1096-1099, Oct. 12-Oct. 16, 1998.

[26] Farzin Mokhtarian, Alan K. Mackworth, "A Theory of Multiscale, Curvature-Based Shape Representation for Planar Curves", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 8, pp. 789-805, 1992.

[27] Mikhail Mozerov, Vitaly Kober, Tae S. Choi, "Color Motion Stereo Based on Adaptive Correlation Matching", Proceedings of SPIE - The International Society for Optical Engineering 3808, pp. 693-701, Jul. 20-Jul. 23, 1999.

[28] Hideo Ogawa, "Corner Detection on Digital Curves Based on Local Symmetry of the Shape", Pattern Recognition, Vol. 22, No. 4, pp. 351-357, 1989.

[29] Dietrich W. R. Paulus, Joachim Hornegger, "Applied Pattern Recognition: A Practical Introduction to Image and Speech Processing in C++", Wiesbaden, Verlag Vieweg, 1998.

[30] H. Penz, I. Bajla, K. Mayer, W. Krattenthaler, "High-Speed Template Matching with Point Correlation in Image", Proceedings of SPIE - The International Society for Optical Engineering 3827, pp. 85-94, Jun. 14-Jun. 15, 1999.

[31] Anothai Rattarangsi, Roland T. Chin, "Scale-Based Detection of Corners of Planar

Curves", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 4, pp. 430-449, 1992.

[32] Azriel Rosenfeld, Emily Johnston, "Angle Detection on Digital Curves", IEEE Trans. Comput., C-22, No. 9, pp. 875-878, 1973.

[33] Azriel Rosenfeld, Joan S. Weszka, "An Improved Method of Angle Detection on Digital Curves", IEEE Trans. Comput., C-24, No. 9, pp. 940-941, 1975.

[34] Francisco J. Sanchez-Marin, "Automatic Recognition of Biological Shapes with and without Representations of Shape", Artificial Intelligence in Medicine, Vol. 18, Elsevier Science B. V., pp. 173-186, 2000.

[35] Robert A. Schowengerdt, "Remote Sensing-Models and Methods for Image Processing" (Second Edition), Academic Press, 1997.

[36] M. M. Selim, "Recognition of 2-D Shapes Using Complex Neural Networks: A Novel Approach", Journal of Engineering and Applied Science, Vol. 45, No. 5, pp. 811-821, Oct. 1998.

[37] Z. Shao, J. Kittler, "Shape Representation and Recognition Based on Invariant Unary and Binary Relations", Image and Vision Computing, Vol. 17, No. 5, Elsevier Science Ltd., pp. 429-444, 1999.

[38] Tzung-Sz Shen, Jianbing Huang, Chia-Hsiang Menq, "Multiple-Sensor Integration for Rapid and High-Precision Coordinate Metrology", Proc. the 1999 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 908-915, 1999.

[39] Oliver Sidla, Ernst Wilding, "Efficient Shape Recognition for the Detection of Reusable Material in A Waste Processing Plant", Proceedings of SPIE - The International Society for Optical Engineering 3827, pp. 52-58, Jun. 14-Jun. 15, 1999.

[40] Cho-huak Teh, Roland T. Chin, "On the Detection of Dominant Points on Digital Curves", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, No. 8, pp. 859-872, 1989.

[41] Naonori Ueda, Satoshi Suzuki, "Learning Visual Models From Shape Contours Using Multiscale Convex/Concave Structure Matching", IEEE Transactions on Pattern

Analysis and Machine Intelligence, Vol. 15, No. 4, pp. 337-352, 1993.

[42] C. C. Yang, M. M. Marefat, R. L. Kashyap, "Active Visual Inspection Based on CAD Models", Proc. IEEE, pp. 1120-1125, 1994.

[43] Tzay Y. Young, King-Sun Fu, "Handbook of Pattern Recognition and Image Processing", Orlando, Academic Press, c1986.

[44] P. Zhu, P. M. Chirlian, "On Critical Point Detection of Digital Shapes", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17, No. 8, pp.737-748, 1995.

[45] "Image Pattern Recognition: Algorithm Implementations, Techniques, and Technology", 1987, Los Angeles, California, Computer Society of the IEEE, Pattern Recognition Society, January 12-13.

[46] "Pattern Recognition and Image Processing in Physics: Proceedings of the Thirty-Seventh Scottish Universities' Summer School in Physics", Dundee, July 29-August 18, 1990.

Appendix A

Correlation Theorem

If the Fourier transform of $f[i,j]$ is $F[u,v]$ and that of $g[i,j]$ is $G[u,v]$, the Fourier transform of the correlation of two functions $f[i,j]$ and $g[i,j]$ is the product of their Fourier transforms with one of them complex conjugated. Thus,

$$f[i,j] \circ g[i,j] \Leftrightarrow F[u,v] \times G^*[u,v],$$

which indicates that the inverse transform of the right hand side gives the correlation of the two functions in the $[i,j]$ domain. An analogous result is formally stated as:

$$f[i,j] \times g^*[i,j] \Leftrightarrow F[u,v] \circ G\ [u,v].$$

These two results together constitute the correlation theorem.

Appendix B

Matlab Code for Algorithm Based on The Correlation Method

```
%*************************************************************************************************
%File Name: MatCorrelation.m
%This program is used to find the 2-D position deviation of the input image compared
%with the template image along both X and Y directions.
%*************************************************************************************************

%Close all figures and clear all variables
close all;
clear all;
cd d:\matlab53\ghm1;
zoom on;

%Initialize some variables
Thresh=0.2;
TestNumber1=4;
TestNumber2=3;
coe1=0.2;
coe2=0.8;

%Read the template image data from a file and store them in array InData[ ]
%M: The number of total rows of the template image
%N: The number of total columns of the template image
[Temp,map]=imread('model1','bmp');
InData=double(Temp);
[M,N]=size(InData);

%Get the edge of the template image
InDataEdge=double(edge(InData));

%Read the input image data from a file and store them in array InData1[ ]
%M1: The number of total rows of the input image
%N1: The number of total columns of the input image
InData1=double(imread('figure92','bmp'));
[M1,N1]=size(InData1);

%*************************************************************
%Step1: Enhance the contrast of the input image
%*************************************************************
%Initialize the counters
```

```
for i=1:1:256
  Counter(i)=0;
end

%Get the Histogram of the input image
for i=1:1:M1
  for j=1:1:N1
    k=InData1(i,j);
    Counter(k+1)=Counter(k+1)+1;
  end
end

figure
stem(Counter,'k')
xlabel('Gray Level z');
ylabel('H(z)');
title('Histogram Operation');

%Get the Cumulative Probability Density Function of the input image
c(1)=Counter(1);
for i=2:1:256
  c(i)=c(i-1)+Counter(i);
end

figure
plot(c/(M1*N1),'k')
xlabel('Gray Level');
ylabel('Fraction of Total Pixels');
title('Cumulative Probability Density Function');

%Find the corresponding gray-level value where the Cumulative Probability Density is approximately equal
%to coe1(20%)
for i=1:1:256
  if (c(i)/(M1*N1))>=coe1
    break;
  end
end
first=i-1;

%Find the corresponding gray-level value where the Cumulative Probability Density is approximately equal
%to coe2(80%)
for i=1:1:256
  if (c(i)/(M1*N1))>=coe2
    break;
  end
end
last=i-1;

%Process the image
%Set the intensity of the pixels with gray-level lower than 'first' to 0
%Set the intensity of the pixels with gray-level higher than 'last' to 255
%Do linear stretching for the remaining pixels
ratio=255.0/(last-first);
for i=1:1:M1
```

```
  for j=1:1:N1
    if InData1(i,j)<=first
      ContrastImage(i,j)=0;
    else if InData1(i,j)>=last
        ContrastImage(i,j)=255;
      else ContrastImage(i,j)=round((InData1(i,j)-first)*ratio);
      end
    end
  end
end

%Check the processing result
%Get the histogram and the Cumulative Probability Density Function of the processed image
for i=1:1:256
  d(i)=0;
end

for i=1:1:M1
  for j=1:1:N1
    k=ContrastImage(i,j);
    d(k+1)=d(k+1)+1;
   end
end
figure
stem(d,'k')
xlabel('Gray Level z');
ylabel('H(z)');
title('Histogram Operation');

e(1)=d(1);
for i=2:1:256
  e(i)=e(i-1)+d(i);
end
figure
plot(e,'k')
xlabel('Gray Level');
ylabel('Fraction of Total Pixels');
title('Cumulative Probability Density Function');

%Calculate the summation of the intensity values of each row and store it in array OutMax[ ]
Count=1;
Sum=0;
for i=1:1:M1
          for j=1:1:N1
    Sum=Sum+ContrastImage(i,j);
  end
  OutMax(i)=Sum;
  Sum=0;
  Count=Count+1;
end
MAX=max(OutMax);
%*************************************************************************************
%Step2: Find the approximate position of the complete foam barrier in the input image
%*************************************************************************************
```

```
%[2.1] Find the first 'Black Band', which consists of at least 'TestNumber2+1' 'Black Lines' out of
%'TestNumber1+1' consecutive lines
Flag=0;
LineNumber=1;
while(LineNumber<=M1)
  if (OutMax(LineNumber)<=(Thresh*MAX))
    BlackLine=0;
    for i=1:1:TestNumber1
      LineNumber=LineNumber+1;
      if (LineNumber>=M1)
        Flag=1;
        break;
      else
        if (OutMax(LineNumber)<=(Thresh*MAX))
          BlackLine=BlackLine+1;
        end
      end
    end
    if(Flag==1)
      break;
    else
      if(BlackLine>=TestNumber2)
        break;
      else
        LineNumber=LineNumber+1;
      end
    end
  else
    LineNumber=LineNumber+1;
  end
end

%[2.2] Find the first 'White Band', which consists of at least 'TestNumber2+1' 'White lines' out of
%'TestNumber1+1' consecutive lines
Flag1=0;
while(LineNumber<=M1)
  if (OutMax(LineNumber)>=(Thresh*MAX))
    WhiteLine=0;
    for i=1:1:TestNumber1
      LineNumber=LineNumber+1;
      if (LineNumber>=M1)
        Flag1=1;
        break;
      else
        if (OutMax(LineNumber)>=(Thresh*MAX))
          WhiteLine=WhiteLine+1;
        end
      end
    end
    if(Flag1==1)
      break;
    else
      if(WhiteLine>=TestNumber2)
        break;
```

```
      else
        LineNumber=LineNumber+1;
      end
    end
  else
    LineNumber=LineNumber+1;
  end
end
Band1=LineNumber;

%[2.3] Find the 2nd 'Black Band', which consists of at least 'TestNumber2+1' 'Black Lines' out of
%'TestNumber1+1' consecutive lines
Flag2=0;
while(LineNumber<=M1)
  if (OutMax(LineNumber)<=(Thresh*MAX))
    BlackLine=0;
    for i=1:1:TestNumber1
      LineNumber=LineNumber+1;
      if (LineNumber>=M1)
        Flag2=1;
        break;
      else
        if (OutMax(LineNumber)<=(Thresh*MAX))
          BlackLine=BlackLine+1;
        end
      end
    end
    if(Flag2==1)
      break;
    else
      if(BlackLine>=TestNumber2)
        break;
      else
        LineNumber=LineNumber+1;
      end
    end
  else
    LineNumber=LineNumber+1;
  end
end
Band2=LineNumber;

%[2.4] Find the approximate position of the 'Central Line' of the possible foam barrier
Central=round((Band1+Band2)/2);

%**********************************************************************************************
%Step3:Crop the input image to the same size as that of the template image
%**********************************************************************************************
if(Central<=M/2)
  InData2=InData1(1:M,:);
else if((M1-Central)<=M/2)
  InData2=InData1((M1-M+1):M1,:);
else if(mod(M,2)==0)
    InData2=InData1((Central-M/2):(Central+(M/2-1)),:);
```

```
  else
    InData2=InData1((Central-(M-1)/2):(Central+(M-1)/2),:);
  end
end
end

%Get the edge of the cropped image
InDataEdge2=double(edge(InData2));

%*************************************************************************************************************
%Step4: Find the edges of the template image and the cropped image
%*************************************************************************************************************
%Write the 2 edged images into 2 files
imwrite(uint8(round(InDataEdge*255)),map,'modeledge.bmp');
imwrite(uint8(round(InDataEdge2*255)),map,'online.bmp');

%*************************************************************************************************************
%Step5: Calculate the correlation between the edged template image data and cropped image data
%*************************************************************************************************************
OutDataEdge2 =xcorr2(InDataEdge2,InDataEdge);

for i=1:1:(2*M-1)
  CoordinateX(i)=i-M;
  CoordinateY(i)=i-M;
end
figure;
mesh(CoordinateX,CoordinateY,OutDataEdge2);

%*************************************************************************************************************
%Step6:Calculate the deviations in both X-direction and Y-direction
%*************************************************************************************************************
Flag=0;
Smax=max(max(OutDataEdge2));
for i=1:1:(2*M-1)
  for j=1:1:(2*N-1)
    if OutDataEdge2(i,j)>=Smax
      Position1=i;
      Position2=j;
      Flag=1;
      break;
    end
  end
  if (Flag==1)
    break;
  end
 end

%Calculate the deviations in both x-direction and y-direction
if(Central<=M/2)
  Position1=Position1-M;
else if((M1-Central)<=M/2)
  Position1=Position1-M+(M1-M+1);
else if(mod(M,2)==0)
    Position1=Position1-M+(Central-M/2);
```

```
  else
    Position1=Position1-M+(Central-(M-1)/2);
  end
end
end
Position1=Position1-(M1-M)/2;
Position2=Position2-N;
%**********************************************END
```

Appendix C

Matlab Code for Algorithm Based on The Directional Flow-Change Method

Appendix C.1: *Image Binarization*

```
%*************************************************************************
%File Name: Binarization.m
%This program is used to get the binary image of an input image.
%*************************************************************************

%Close all figures and clear all variables
close all;
clear all;
cd d:\matlab53\ghm2;
zoom on;

%Read the input image data from a file and store them in array InData1[ ]
%M1: The number of the total rows of the input image
%N1: The number of the total columns of the input image
[Temp,map]=imread('Example -2','bmp');
InData1=double(Temp);
[M1,N1]=size(InData1);

%Initialize the counters
for i=1:1:256
  Counter(i)=0;
end

%Get the Histogram of the input image
for i=1:1:M1
  for j=1:1:N1
    k=InData1(i,j);
    Counter(k+1)=Counter(k+1)+1;
  end
end

figure
stem(Counter,'k')
xlabel('Gray Level z');
ylabel('H(z)');
title('Histogram Operation');
```

```
for i=1:1:32
  LocalMax(i)=max(Counter(((i-1)*8+1):(i*8)));
end

%Find the fist point, which has the local maximum value
Temparary=LocalMax(1);
Counter3=0;
for i=2:1:32
  if Temparary>LocalMax(i)
    FirstMax=i-1;
    break;
  else
    Temparary=LocalMax(i);
  end
end

%Find the last point, which has the local maximum value
LastMax=0;
Temparary=LocalMax(32);
for j=31:-1:i
  if Temparary>LocalMax(j)
    LastMax=j+1;
    break;
  else
    Temparary=LocalMax(j);
  end
end

if LastMax==0
  LastMax=32;
end

%Find the threshold value for converting gray level image to binary image
MiddleMax=(FirstMax+LastMax)*4;

%Process the image
%Set the intensity of the pixels with gray-level lower than 'MiddleMax' to 0
%Set the intensity of the pixels with gray-level higher than 'MiddleMax' to 1
for i=1:1:M1
  for j=1:1:N1
    if InData1(i,j)<=MiddleMax
      ContrastImage(i,j)=0;
    else
      ContrastImage(i,j)=1;
    end
  end
end

cd d:\matlab53\ghm2;
imwrite(uint8(round(ContrastImage*255)),map,'binary.bmp');
```

Appendix C.2: *Edge Detection and Noise Reduction*

```
%*************************************************************************
%File Name: EdgeofImage.m
%This program is used to detect the edge of a planar shape and remove the noise
%presented in the corresponding image of the shape.
%*************************************************************************

clear all;
close all;
cd d:\matlab53\ghm2

%*********************************************************
%Part1: Edge Detection Algorithm
%*********************************************************
%Load data from an image file
[Temp,map]=imread('binary','bmp');
InData=double(Temp);
[Rows,Coloums]=size(InData);

%Process the data using the threshold value '1'
for i=1:1:(Rows-1)
  for j=1:1:(Coloums-1)
    if InData(i,j)~=InData(i+1,j) | InData(i,j)~=InData(i,j+1)
      InputData3(i,j)=1;
    else
      InputData3(i,j)=0;
    end
  end
end

%Write the edged image data into a file
imwrite(uint8(round(InputData3*255)),map,'Newedge2.bmp');

%*************************************************************************
%Part2: Noise Reduction Algorithm
%*************************************************************************
BrightValue3=1;

%Create a pure black image
for i=1:1:Rows
  for j=1:1:Coloums
    InputData2(i,j)=0;
  end
end
```

```
%*************************************************************************************
%Step1: Randomly choose a point inside the contour as the starting point
%*************************************************************************************
PointX=round(Rows/2);
PointY=round(Coloums/2);
Flag=0;

%Check if the chosen point and its surrounding 8 pixels are white
% If yes, go to Step 2
% If no, continue to search the image until the required point is found
for i=PointX:2:Rows
  for j=PointY:2:Coloums
    if InData(i,j)==1 & InData(i,j-1)==1 & InData(i,j+1)==1
      if InData(i-1,j)==1 & InData(i-1,j-1)==1 & InData(i-1,j+1)==1
        if InData(i+1,j)==1 & InData(i+1,j-1)==1 & InData(i+1,j+1)==1
          SearchX(1,1)=i;
          SearchY(1,1)=j;
          Flag=1;
        end
      end
    end
    if Flag==1 break;
    end
  end
  if Flag==1 break;
  end
end

%If the required point is not found, continue to search the image
if Flag==0
for i=PointX:2:Rows
  for j=PointY:-2:1
    if InData(i,j)==1 & InData(i,j-1)==1 & InData(i,j+1)==1
      if InData(i-1,j)==1 & InData(i-1,j-1)==1 & InData(i-1,j+1)==1
        if InData(i+1,j)==1 & InData(i+1,j-1)==1 & InData(i+1,j+1)==1
          SearchX(1,1)=i;
          SearchY(1,1)=j;
          Flag=1;
        end
      end
     end
    if Flag==1 break;
    end
  end
  if Flag==1 break;
  end
end
end

%If the required point is not found, continue to search the image
if Flag==0
for i=PointX:-2:1
  for j=PointY:-2:1
    if InData(i,j)==1 & InData(i,j-1)==1 & InData(i,j+1)==1
```

```
      if InData(i-1,j)==1 & InData(i-1,j-1)==1 & InData(i-1,j+1)==1
        if InData(i+1,j)==1 & InData(i+1,j-1)==1 & InData(i+1,j+1)==1
          SearchX(1,1)=i;
          SearchY(1,1)=j;
          Flag=1;
        end
       end
     end
    if Flag==1 break;
    end
  end
  if Flag==1 break;
  end
end
end

%If the required point is not found, continue to search the image
if Flag==0
for i=PointX:-2:1
  for j=PointY:2:Coloums
    if InData(i,j)==1 & InData(i,j-1)==1 & InData(i,j+1)==1
      if InData(i-1,j)==1 & InData(i-1,j-1)==1 & InData(i-1,j+1)==1
        if InData(i+1,j)==1 & InData(i+1,j-1)==1 & InData(i+1,j+1)==1
          SearchX(1,1)=i;
          SearchY(1,1)=j;
          Flag=1;
         end
       end
     end
    if Flag==1 break;
    end
  end
  if Flag==1 break;
  end
end
end

%**************************************************************************************************
%Step 2: Process the adjacent 4 pixels of the starting point
%Step 3: Repeat the procedure until all the pixels inside the closed contour become 'white'
%Step 4: Change the all pixels outside the contour to be 'black'
%**************************************************************************************************
CounterLine=1;
InputData2(SearchX(1,1),SearchY(1,1))=BrightValue3;
k=-1;
i=2;
while(k~=1)
  j=1;
  k=1;
  while(j<=CounterLine)
    if InputData3(SearchX(i-1,j),SearchY(i-1,j)+1)==0
      InputData3(SearchX(i-1,j),SearchY(i-1,j)+1)=BrightValue3;
      InputData2(SearchX(i-1,j),SearchY(i-1,j)+1)=BrightValue3;
      SearchX(i,k)=SearchX(i-1,j);
```

```
      SearchY(i,k)=SearchY(i-1,j)+1;
      k=k+1;
    else
      InputData2(SearchX(i-1,j),SearchY(i-1,j)+1)=BrightValue3;
    end
    if InputData3(SearchX(i-1,j)+1,SearchY(i-1,j))==0
      InputData3(SearchX(i-1,j)+1,SearchY(i-1,j))=BrightValue3;
      InputData2(SearchX(i-1,j)+1,SearchY(i-1,j))=BrightValue3;
      SearchX(i,k)=SearchX(i-1,j)+1;
      SearchY(i,k)=SearchY(i-1,j);
      k=k+1;
    else
      InputData2(SearchX(i-1,j)+1,SearchY(i-1,j))=BrightValue3;
    end
    if InputData3(SearchX(i-1,j),SearchY(i-1,j)-1)==0
      InputData3(SearchX(i-1,j),SearchY(i-1,j)-1)=BrightValue3;
      InputData2(SearchX(i-1,j),SearchY(i-1,j)-1)=BrightValue3;
      SearchX(i,k)=SearchX(i-1,j);
      SearchY(i,k)=SearchY(i-1,j)-1;
      k=k+1;
    else
      InputData2(SearchX(i-1,j),SearchY(i-1,j)-1)=BrightValue3;
    end
    if InputData3(SearchX(i-1,j)-1,SearchY(i-1,j))==0
      InputData3(SearchX(i-1,j)-1,SearchY(i-1,j))=BrightValue3;
      InputData2(SearchX(i-1,j)-1,SearchY(i-1,j))=BrightValue3;
      SearchX(i,k)=SearchX(i-1,j)-1;
      SearchY(i,k)=SearchY(i-1,j);
      k=k+1;
     else
      InputData2(SearchX(i-1,j)-1,SearchY(i-1,j))=BrightValue3;
     end
    j=j+1;
  end
  CounterLine=k-1;
  i=i+1;
end

%Display the image after noise reduction
colormap(map);
image(InputData2*255);

%Write the image data after noise reduction into a file
imwrite(uint8(round(InputData2*255)),map,'try.bmp');
```

Appendix C.3: *Algorithm Based on The Directional Flow-Change Method*

```
%*************************************************************************
%File Name: DirectionalFlowChangeFinal.m
%This program is used to identify the pattern of a target and find the deviations along
%both X and Y directions or the rotation angle of the input image of the target compared
%with the reference position of the template image.
%*************************************************************************

clear all;
close all;

%Load the template image data from a file
[InputData2,map]=imread('polygontwice+90.bmp');

%Load the input image data from a file
[InputData,map]=imread('try.bmp');
[Rows,Coloums]=size(InputData);

%Initialize some variables
%Define the 'Supported Rate' 'Gamma'
BrightValue=255;
BrightValue1=255;
Flag=0;
Gamma=0.03;
coef1=0.8;
coef2=1.2;
ThresholdLength=0.25

%*************************************************************************************
%Step1: Get the chain codes of the closed contour and store them in array d[ ]
%*************************************************************************************
%[1.1] Choose the top-left pixel on the contour as the starting point and store its coordinates in
%(StartPoint1,StartPoint2)
for i=1:1:Rows
  for j=1:1:Coloums
    if InputData(i,j)==BrightValue
      StartPoint1=i;
      StartPoint2=j;
      Flag=1;
      break;
    end
  end
  if Flag==1
    break;
```

```
 end
end

Count=1;
Deadloop=0;

%Find the chain code of the chosen point
% Just check it in 4 directions
if InputData(i,j+1)==BrightValue
  d(Count)=2;
  j=j+1;
  TempRow(Count)=i;
  TempCol(Count)=j;
  else if InputData(i+1,j+1)==BrightValue
      d(Count)=3;
      i=i+1;
      j=j+1;
            TempRow(Count)=i;
      TempCol(Count)=j;
    else if InputData(i+1,j)==BrightValue
        d(Count)=4;
        i=i+1;
                TempRow(Count)=i;
        TempCol(Count)=j;
      else if InputData(i+1,j-1)==BrightValue
          d(Count)=5;
          i=i+1;
          j=j-1;
                  TempRow(Count)=i;
          TempCol(Count)=j;
        end
      end
    end
  end

%[1.2] Create an array Binary[8]
%Check the gray level of the pixel, which is in the direction 0 of a certain point
%If it is 1, set the value Binary(0) as 1
%If it is 0, set the value Binary(0) as 0
%Follow the same procedure to check the gray levels of the pixels in the rest 7 directions, and set the
%corresponding values of Binary(1),Binary(2),.....,Binary(7) either as 1 or 0
Count=Count+1;
while(Deadloop==0 )
  if InputData(i-1,j)==BrightValue
    Binary(1)=1;
     cc(1)=0;
     TempRoww(1)=i-1;
    TempColl(1)=j;
   else
    Binary(1)=0;
   end
   if InputData(i,j+1)==BrightValue
    Binary(3)=1;
    cc(3)=2;
```

```
  TempRoww(3)=i;
  TempColl(3)=j+1;
else
  Binary(3)=0;
end
if InputData(i+1,j)==BrightValue
  Binary(5)=1;
  cc(5)=4;
  TempRoww(5)=i+1;
  TempColl(5)=j;
else
  Binary(5)=0;
end
if InputData(i,j-1)==BrightValue
  Binary(7)=1;
  cc(7)=6;
  TempRoww(7)=i;
  TempColl(7)=j-1;
else
  Binary(7)=0;
end
if InputData(i-1,j+1)==BrightValue
  Binary(2)=1;
  cc(2)=1;
      TempRoww(2)=i-1;
  TempColl(2)=j+1;
else
  Binary(2)=0;
end
if InputData(i+1,j+1)==BrightValue
  Binary(4)=1;
  cc(4)=3;
  TempRoww(4)=i+1;
  TempColl(4)=j+1;
else
  Binary(4)=0;
end
if InputData(i+1,j-1)==BrightValue
  Binary(6)=1;
  cc(6)=5;
  TempRoww(6)=i+1;
  TempColl(6)=j-1;
else
  Binary(6)=0;
end
if InputData(i-1,j-1)==BrightValue
  Binary(8)=1;
  cc(8)=7;
  TempRoww(8)=i-1;
  TempColl(8)=j-1;
else
  Binary(8)=0;
end
```

```
%[1.3]Generate the chain code from the array Binary[8]
%Extend the size of Binary[ ] from 8 to 16
Binary(9)=Binary(1);
Binary(10)=Binary(2);
Binary(11)=Binary(3);
Binary(12)=Binary(4);
Binary(13)=Binary(5);
Binary(14)=Binary(6);
Binary(15)=Binary(7);
Binary(16)=Binary(8);

%Look for the first '0' in the array Binary[ ]
for i=1:1:16
  if Binary(i)==0;
    break;
  end
end

%Look for the first '1' in the array Binary[ ] and save its position in 'Chain1Start'
for j=i:1:16
  if Binary(j)==1
    break;
  end
end
Chain1Start=j;

%Look for the second '0' in the array Binary[ ] and save its previous position in 'Chain1End'
for i=j:1:16
  if Binary(i)==0;
    break;
  end
end
Chain1End=i-1;
if Chain1End>8
  Chain1End=Chain1End-8;
end
if Chain1Start>8
  Chain1Start=Chain1Start-8;
end

%Determine if  'Chain1Start' or  'Chain1End' is the next chain code
if abs(d(Count-1)-cc(Chain1Start))==4
  d(Count)=cc(Chain1End);
  TempRow(Count)=TempRoww(Chain1End);
  TempCol(Count)=TempColl(Chain1End);
Else
  d(Count)=cc(Chain1Start);
  TempRow(Count)=TempRoww(Chain1Start);
  TempCol(Count)=TempColl(Chain1Start);
end
i=TempRow(Count);
   j=TempCol(Count);
```

```
%[1.4] Repeat the procedure until the tracing process reaches the original starting point
%Check if the tracing process reaches the starting point
% If yes, terminate the finding algorithm
if i==StartPoint1 & j==StartPoint2
   break;
 end
 Count=Count+1;
end

%'dCount' is the total number of the pixels of the closed contour
dCount=Count;
%**********************************************************************************************************
%Step 2:Calculate the directional flow-change at each point on the contour with contour segment
%of length 'J' on both sides of the point
%**********************************************************************************************************
%Define the 'Supported Length' 'J'
J=floor(Gamma*dCount);
if abs(J-Gamma*dCount)>0.5
  J=J+1;
end

%Extend the array d[ ] both at the beginning point and the ending point with 'J' elements
%Save the extended array in e[ ] and the size of it becomes 'dCount+J*2'
for i=1:1:J
  e(i)=d(dCount-J+i);
end
for i=(J+1):1:(J+dCount)
  e(i)=d(i-J);
end
for i=(J+dCount+1):1:(dCount+2*J)
  e(i)=d(i-J-dCount);
end

%Initialize some variables related to directional flow-change calculation
for i=1:1:dCount
  G0In(i)=0;
  G1In(i)=0;
  G2In(i)=0;
  G3In(i)=0;
  G4In(i)=0;
  G5In(i)=0;
  G6In(i)=0;
  G7In(i)=0;
  G0Out(i)=0;
  G1Out(i)=0;
  G2Out(i)=0;
  G3Out(i)=0;
  G4Out(i)=0;
  G5Out(i)=0;
  G6Out(i)=0;
  G7Out(i)=0;
end
```

```
%[2.1]Calculate the input flows in all 8 directions at each point on the contour with contour
%segment of length 'J' and store them in  G0In[ ]~G7In[ ]
for i=(J+1):1:(dCount+J)
  for j=1:1:J
          if e(i-j)==0 G0In(i-J)=G0In(i-J)+1;
      else if e(i-j)==1 G1In(i-J)=G1In(i-J)+1;
      else if e(i-j)==2 G2In(i-J)=G2In(i-J)+1;
           else if e(i-j)==3 G3In(i-J)=G3In(i-J)+1;
                else if e(i-j)==4 G4In(i-J)=G4In(i-J)+1;
                     else if e(i-j)==5 G5In(i-J)=G5In(i-J)+1;
                          else if e(i-j)==6 G6In(i-J)=G6In(i-J)+1;
                     else G7In(i-J)=G7In(i-J)+1;
                     end
                end
               end
              end
             end
            end
          end
        end
      end
    end
  end

%[2.2]Calculate the output flows in all 8 directions at each point on the contour with contour
%segment with length 'J'  and store them in  G0Out[ ]~G7Out[ ]
for i=(J+1):1:(dCount+J)
  for j=1:1:J
          if e(i+j-1)==0 G0Out(i-J)=G0Out(i-J)+1;
      else if e(i+j-1)==1 G1Out(i-J)=G1Out(i-J)+1;
      else if e(i+j-1)==2 G2Out(i-J)=G2Out(i-J)+1;
           else if e(i+j-1)==3 G3Out(i-J)=G3Out(i-J)+1;
                else if e(i+j-1)==4 G4Out(i-J)=G4Out(i-J)+1;
                     else if e(i+j-1)==5 G5Out(i-J)=G5Out(i-J)+1;
                          else if e(i+j-1)==6 G6Out(i-J)=G6Out(i-J)+1;
                     else G7Out(i-J)=G7Out(i-J)+1;
                     end
                end
               end
              end
             end
            end
          end
        end
      end
    end
  end

%[2.3]Calculate the flow changes in all 8 directions at each point on the contour with contour segment
%of length 'J' on both sides of a point and store them in  G0Change[ ]~G7Change[ ]
for i=1:1:dCount
  G0Change(i)=abs(G0Out(i)-G0In(i));
  G1Change(i)=abs(G1Out(i)-G1In(i));
  G2Change(i)=abs(G2Out(i)-G2In(i));
  G3Change(i)=abs(G3Out(i)-G3In(i));
  G4Change(i)=abs(G4Out(i)-G4In(i));
  G5Change(i)=abs(G5Out(i)-G5In(i));
  G6Change(i)=abs(G6Out(i)-G6In(i));
```

```
  G7Change(i)=abs(G7Out(i)-G7In(i));
end

%[2.4]Calculate the directional flow change at each point on the contour with contour segment
%of length 'J' on both sides of a point and store it in Delta[ ]
for i=1:1:dCount

Delta(i)=abs(G0Change(i)+G1Change(i)+G2Change(i)+G3Change(i)+G4Change(i)+G5Change(i)+G6Chang
e(i)+G7Change(i));
end

%**************************************************************************
%Step 3:Detect the critical points of the closed contour
%**************************************************************************

Delta1=Delta;
AccumDelta1=0;
for i=1:1:dCount
  AccumDelta1=AccumDelta1+Delta1(i);
end

%Define three variables 't', 'L' and 'theta'
t=0.8;
ThresholdT=t*J;
L=floor(0.8*J);
if abs(L-0.7*J)>=0.5
  L=L+1;
end
figure(4)
plot(Delta1,'k')
Theta=0.9;

%Extend the array Delta[ ] both at the beginning point and the ending point with 'L' elements and store the
%new data in the array Delta1[ ]
%The length of the array Delta1[ ] becomes 'dCount+L*2'
w=ceil(L/2);
for i=1:1:w
  Delta1(i)=Delta(dCount-w+i);
end
for i=(w+1):1:(w+dCount)
  Delta1(i)=Delta(i-w);
end
for i=(w+dCount+1):1:(dCount+2*w)
  Delta1(i)=Delta(i-w-dCount);
end
for i=(w+1):1:(dCount+w)
    accum=0;
  for j=-floor((w-1)/2):1:ceil((w-1)/2)
    accum=accum+Delta1(i+j);
  end
  Delta(i-w)=accum/w;
end
Delta2=Delta;
figure(2)
```

```
plot(Delta,'k')

%Detect the critical points of the contour (see chapter 2.3.3 for details)
CriticalNumber=0;
FlagCount=0;
for i=1:1:dCount
  if Delta2(i)>=ThresholdT*coef1
    break;
  end
end

for j=(i+1):1:dCount
  if Delta2(j)>Delta2(i)
    i=j;
  else
    if Delta2(j)<=Theta*Delta2(i)
      break;
    end
  end
end
Mini=j;
HighCount=0;
Minimum=Delta2(j);
for k=j:1:(dCount-1)
  if Delta2(k+1)>Delta2(k)
    HighCount=HighCount+1;
  else
    Minimum=Delta2(k+1);
    Mini=k+1;
  end
  if HighCount>ceil(L/2)
    break;
  end
end
pp=Mini;

d=Mini;

for m=(dCount+1):1:(dCount+Mini)
  Delta2(m)=Delta(m-dCount);
end

while(d<(dCount+pp))
for i=(d+1):1:(dCount+pp)
  if Delta2(i)>=ThresholdT*coef1
    break;
  end
end

for j=(i+1):1:(dCount+pp)
  if Delta2(j)>Delta2(i)
    i=j;
  else
    if Delta2(j)<=Theta*Delta2(i)
```

```
      break;
    end
  end
end

Mini=j;
HighCount=0;
Minimum=Delta2(j);
for k=j:1:(dCount+pp-1)
  if Delta2(k+1)>Delta2(k)
    HighCount=HighCount+1;
  else
    Minimum=Delta2(k+1);
    Mini=k+1;
  end
  if HighCount>ceil(L/2)
    break;
  end
end

dd=Mini;
for m=d:1:dd
  if Delta2(m)==Delta2(i)
    break;
  end
end
ave1=m;
for m=dd:-1:d
  if Delta2(m)==Delta2(i)
    break;
  end
end
ave2=m;
ave=round((ave1+ave2)/2);
Candidate=ave;
if CriticalNumber==0
  CriticalNumber=CriticalNumber+1;
  if Delta2(ave)<=ThresholdT*coef2
    Flag(CriticalNumber)=1;
    FlagCount=FlagCount+1;
  else
    Flag(CriticalNumber)=0;
  end

  CriticalPoints(CriticalNumber)=ave;
  Previous=Candidate;
  d=dd;
else if (Candidate-Previous)>=L
    CriticalNumber=CriticalNumber+1;
    if Delta2(ave)<=ThresholdT*coef2
      Flag(CriticalNumber)=1;
      FlagCount=FlagCount+1;
    else
      Flag(CriticalNumber)=0;
```

```
    end

    CriticalPoints(CriticalNumber)=ave;
    Previous=Candidate;
  else if Delta2(Previous)<Delta2(Candidate)
      Previous=Candidate;
    end
  end
end
d=dd;
end

%Check if there are any special critical points
%If yes, process them
%If there is only one special critical point, run the following fragment to remove it, or skip the program to
keep it
q=2^FlagCount;
if FlagCount==1
  for i=1:1:CriticalNumber
    if Flag(i)==1
      break;
    end
  end
  for j=i:1:(CriticalNumber-1)
    CriticalPoints(j)=CriticalPoints(j+1);
  end
  CriticalNumber= CriticalNumber-1;
 end

%If there are two special critical points, run the following program
%Run the following fragment to remove the first special critical point or skip the fragment to keep it
if FlagCount==2
  for i=1:1:CriticalNumber
    if Flag(i)==1
      break;
    end
  end
  for j=i:1:(CriticalNumber-1)
    CriticalPoints(j)=CriticalPoints(j+1);
  end
  CriticalNumber= CriticalNumber-1;
end

%Run the following fragment to remove the second special critical point or skip the fragment to keep it
if FlagCount==2
  for i=CriticalNumber:-1:1
    if Flag(i)==1
      break;
    end
  end
  for j=i:1:(CriticalNumber-1)
    CriticalPoints(j)=CriticalPoints(j+1);
  end
  CriticalNumber= CriticalNumber-1;
```

```
end

%Run the following fragment to remove both special critical points or skip the fragment to keep them
if FlagCount==2
  for i=1:1:CriticalNumber
    if Flag(i)==1
      m=i;
     break;
    end
  end
  for i=CriticalNumber:-1:1
    if Flag(i)==1
      n=i;
     break;
    end
  end
  for i=m:1:n-2
    CriticalPoints(i)=CriticalPoints(i+1);
  end
  for i=(n-1):1:CriticalNumber-2
    CriticalPoints(i)=CriticalPoints(i+2);
  end
  CriticalNumber= CriticalNumber-2;
end

%Delete the last redundant critical point if the condition is satisfied
if dCount-CriticalPoints(CriticalNumber)+CriticalPoints(1)<=L
  CriticalPoints(1)=CriticalPoints(CriticalNumber);
  if CriticalPoints(1)>dCount
    CriticalPoints(1)=CriticalPoints(1)-dCount;
  end
  CriticalNumber= CriticalNumber-1;
  else if CriticalPoints(CriticalNumber)>dCount
    CriticalPoints(CriticalNumber)=CriticalPoints(CriticalNumber)-dCount;
  end
end

%Find the x and y coordinates of the pixels on the closed contour and store them in arrays TempRow1[ ]
%and TempCol[ ]
TempRow1(1)=TempRow(dCount);
TempCol1(1)=TempCol(dCount);
for i=2:1:dCount
  TempRow1(i)=TempRow(i-1);
  TempCol1(i)=TempCol(i-1);
end


%Display the original image and the critical points simultaneously
%Find the x and y coordinates of the critical points and store them in
%CriticalCoordinateX[ ] and CriticalCoordinateY[ ]
figure(3)
colormap(map);
image(InputData);
for i=1:1:CriticalNumber
```

```
    hold on;
   CriticalCoordinateX(i)=TempCol1(CriticalPoints(i));
    CriticalCoordinateY(i)=TempRow1(CriticalPoints(i));
  plot(CriticalCoordinateX(i),CriticalCoordinateY(i),'k^-');
end

%*****************************************************************************************************
%Step 4:Calculate the angels of the polygon formed by connecting every two neighboring critical points
%*****************************************************************************************************

%Calculate the side lengths of the polygon and store them in array SideLength[ ]
%(x_1,y_1):  The previous point
%(x, y): The current point
%(x1,y1): The next point
x=CriticalCoordinateX(1);
y=CriticalCoordinateY(1);
x_1=CriticalCoordinateX(CriticalNumber);
y_1=CriticalCoordinateY(CriticalNumber);
for i=2:1:CriticalNumber
   x1=CriticalCoordinateX(i);
   y1=CriticalCoordinateY(i);
   SideLength(i-1)=sqrt((x1-x)^2+(y1-y)^2);
   ThirdLength(i-1)=sqrt((x1-x_1)^2+(y1-y_1)^2);
   MiddlePointX(i-1)=round((x1+x_1)/2);
   MiddlePointY(i-1)=round((y1+y_1)/2);
   x_1=x;
   y_1=y;
   x=x1;
   y=y1;
end
SideLength(CriticalNumber)=sqrt((x1-CriticalCoordinateX(1))^2+(y1-CriticalCoordinateY(1))^2);
ThirdLength(CriticalNumber)=sqrt((CriticalCoordinateX(1)-x_1)^2+(CriticalCoordinateY(1)-y_1)^2);
MiddlePointX(CriticalNumber)=round((CriticalCoordinateX(1)+x_1)/2);
MiddlePointY(CriticalNumber)=round((CriticalCoordinateY(1)+y_1)/2);

%Calculate the angles of the polygon formed by connecting every two neighboring critical points
%Check if the angle is an inner or outer angle and store them in array CriticalAngle[ ]
CriticalAngle(1)=acos(((SideLength(1))^2+(SideLength(CriticalNumber))^2-
(ThirdLength(1))^2)/(2*SideLength(1)*SideLength(CriticalNumber)));
if InputData(MiddlePointY(1),MiddlePointX(1))==BrightValue1  CriticalAngle(1)=CriticalAngle(1)*180/3.14;
else CriticalAngle(1)=360-CriticalAngle(1)*180/3.14;
end

for i=2:1:CriticalNumber
  CriticalAngle(i)=acos(((SideLength(i))^2+(SideLength(i-1))^2-
                    (ThirdLength(i))^2)/(2*SideLength(i)*SideLength(i-1)));
  if InputData(MiddlePointY(i),MiddlePointX(i))==BrightValue1  CriticalAngle(i)=CriticalAngle(i)*180/3.14;
  else CriticalAngle(i)=360-CriticalAngle(i)*180/3.14;
  end
end
%*****************************************************************************************************
%Step 5:Convert the sequence of angles into a sequence of angle codes called Angle String
%Find the Shape Number of the Angle String
%*****************************************************************************************************
```

```
%Convert the sequence of angles into a sequence of angle codes called Angle String
%Store the Angle String in 'StringCode'
StringCode=0;
for i=1:1:CriticalNumber
  if CriticalAngle(i)>=0 & CriticalAngle(i)<22
    AngleString(i)=0;
  else if CriticalAngle(i)>=22 & CriticalAngle(i)<44
      AngleString(i)=1;
    else if CriticalAngle(i)>=44 & CriticalAngle(i)<66
        AngleString(i)=2;
     else if CriticalAngle(i)>=66 & CriticalAngle(i)<88
         AngleString(i)=3;
         else if CriticalAngle(i)>=88 & CriticalAngle(i)<110
             AngleString(i)=4;
             else if CriticalAngle(i)>=110 & CriticalAngle(i)<132
                 AngleString(i)=5;
                 else if CriticalAngle(i)>=132 & CriticalAngle(i)<154
                     AngleString(i)=6;
                     else if CriticalAngle(i)>=154 & CriticalAngle(i)<18
                         AngleString(i)=7;
                         else if CriticalAngle(i)>=180 & CriticalAngle(i)<206
                           AngleString(i)=8;
                           else if CriticalAngle(i)>=206 & CriticalAngle(i)<228
                               AngleString(i)=9;
                               else if CriticalAngle(i)>=228 & CriticalAngle(i)<250
                                   AngleString(i)=10;
                                   else if CriticalAngle(i)>=250 & CriticalAngle(i)<272
                                       AngleString(i)=11;
                                       else if CriticalAngle(i)>=272 & CriticalAngle(i)<294
                                           AngleString(i)=12;
                                           else if CriticalAngle(i)>=294 & CriticalAngle(i)<316
                                               AngleString(i)=13;
                                             else if CriticalAngle(i)>=316 & CriticalAngle(i)<338
                                                 AngleString(i)=14;
                                                   else  AngleString(i)=15;
                                                end
                                               end
                                           end
                                       end
                                   end
                               end
                           end
                     end
                 end
             end
         end
     end
  end
  end
  StringCode=bitshift(StringCode,4)+AngleString(i);
  if i==2 | i==1
    FinalNumber1=15;
    FinalNumber2=15;
```

```
  else
    FinalNumber1=bitshift(FinalNumber1,4)+15;
    FinalNumber2=bitshift(FinalNumber2,4);
  end
end
FinalNumber2=bitshift(FinalNumber2,4);

%Define a shape number as a permutation of an angle string such that this
%permutation forms an integer of minimum magnitude
%Store it in 'ShapeNumber'
StringCode1=StringCode;
ShapeNumber=StringCode;
Times=1;
Max=StringCode;
for i=2:1:CriticalNumber
  Temp1=bitand(StringCode, FinalNumber1);
  Temp1=bitshift(Temp1,4);
  Temp2=bitand(StringCode,FinalNumber2)/2^((CriticalNumber-1)*4);
  Temp3=bitor(Temp1,Temp2);
  if Temp3<ShapeNumber
    ShapeNumber=Temp3;
     Times=i;
  end
  if Temp3>Max
    Max=Temp3;
  end
  StringCode=Temp3;
end

%*****************************************************************************************************
%Step 6:Compare the shape numbers and the side lengths of the polygons, which are used to
%approximate the planar shapes, between the input image and the template image to see if the
%two patterns do match or not.
%*****************************************************************************************************
%Define the template shape number, side lengths and angles between the vectors and horizontal axis
%Define the average of critical point coordinates of the template image along X and Y directions
TemplateNumber= 42287962;
TemplateLength=[ 34.6554  30.0167  48.0416  101.3163  26.0192 58.2495  37.2156 ];
TemplateAngle=[223.8312  240.0186 177.6136  94.5286  357.7974  304.5087  353.8298];
TemplatePerimeter=0;
TemplateX=68;
TemplateY=66.71
TemplateLengthMin=min(TemplateLength);

%Compare the shape numbers of the polygons between the input image and the template image
%If they are equal, then compare the corresponding side lengths of the two polygons
%If they are also equal , then they belong to the same pattern
Flag5=0;
MiniCount=0;
MiniCount1=0;
FlagMatch=0;
Flag2=0;
Flag3=0;
if ShapeNumber~=TemplateNumber
```

```
  disp('These two shapes do not match');
else  if Max~=ShapeNumber
     for i=Times:1:CriticalNumber
     if abs((SideLength(i)-TemplateLength(i-Times+1))/TemplateLength(i-Times+1))>ThresholdLength
          Flag5=1;
               disp('2These two shapes do not match');
          break;
          end
       end
     if Flag5==0
     for i=1:1:(Times-1)
          if abs((SideLength(i)-TemplateLength(CriticalNumber-Times+i+1))
                        /TemplateLength(CriticalNumber-Times+i+1))>ThresholdLength
          Flag5=1;
                    disp('3These two shapes do not match');
           break;
       end
     end
      disp('These two shapes match');
              FlagMatch=1;
    end
   else
     for i=1:1:CriticalNumber
       if abs((SideLength(i)-TemplateLengthMin)/TemplateLengthMin)<=ThresholdLength
         MiniCount=MiniCount+1;
         Mini(MiniCount)=i;
     end
     end
     for j=1:1:MiniCount
     for i=Mini(j):1:CriticalNumber
          if abs((SideLength(i)-TemplateLength(i-Mini(j)+1))/TemplateLength(i-Mini(j)+1))>ThresholdLength
          Flag2=1;
          end
     end
     if Flag2==0
          for i=1:1:(Mini(j)-1)
               if abs((SideLength(i)-TemplateLength(CriticalNumber-
                     Mini(j)+i+1))/TemplateLength(CriticalNumber-Mini(j)+i+1))>ThresholdLength
               Flag3=1;
          end
          end
     end
     if Flag2==0 & Flag3==0
           disp('These two shapes match');
                        FlagMatch=1;
          break;
       end
       Flag2=0;
       Flag3=0;
     end
     Times=Mini(j);
     end
end
```

```
%If the two patterns do match, complete step 7 or step 8
%*************************************************************************************************************
%Step 7:Calculate the deviations of the input image along the X and Y directions compared with the
%template
%*************************************************************************************************************
%Calculate the averages of the critical point coordinates along X and Y directions
%Calculate the X and Y deviations of the input image compared with the template
if FlagMatch==1
AverageX=0;
AverageY=0;
for i=1:1:CriticalNumber
  AverageX=AverageX+CriticalCoordinateX(i);
  AverageY=AverageY+CriticalCoordinateY(i);
end
AverageX=AverageX/CriticalNumber;
AverageY=AverageY/CriticalNumber;
DeviationX=AverageX-TemplateX;
DeviationY=AverageY-TemplateY;

%*************************************************************************************************************
%Step 8:Calculate the rotation angle of the input image compared with the template image
%*************************************************************************************************************
%Calculate the rotation angle of the input image compared with the template image
ccount=1;
for i=Times:1:CriticalNumber-1
  x_1=CriticalCoordinateX(i);
  y_1=CriticalCoordinateY(i);
     x=CriticalCoordinateX(i+1);
     y=CriticalCoordinateY(i+1);
     DifferenceX1=x-x_1;
     DifferenceY1=(y_1-y);
  Angle1(ccount)=atan2(DifferenceY1,DifferenceX1)*180/3.1416;
  ccount=ccount+1;
end
  x_1=CriticalCoordinateX(CriticalNumber);
  y_1=CriticalCoordinateY(CriticalNumber);
     x=CriticalCoordinateX(1);
     y=CriticalCoordinateY(1);
     DifferenceX1=x-x_1;
     DifferenceY1=(y_1-y);
  Angle1(ccount)=atan2(DifferenceY1,DifferenceX1)*180/3.1416;
  ccount=ccount+1;
for i=1:1:Times-1
  x_1=CriticalCoordinateX(i);
  y_1=CriticalCoordinateY(i);
     x=CriticalCoordinateX(i+1);
     y=CriticalCoordinateY(i+1);
     DifferenceX1=x-x_1;
     DifferenceY1=(y_1-y);
  Angle1(ccount)=atan2(DifferenceY1,DifferenceX1)*180/3.1416;
  ccount=ccount+1;
end

for i=1:1:ccount-1
```

```
  if Angle1(i)<0
    Angle1(i)=360+Angle1(i);
  end
  if TemplateAngle(i)<0
    TemplateAngle(i)=360+TemplateAngle(i);
  end
end

TotalAngle=0;
for i=1:1:ccount-1
  AngleDifference(i)=Angle1(i)-TemplateAngle(i);
  if AngleDifference(i)<0
    AngleDifference(i)=AngleDifference(i)+360;
  end
 if AngleDifference(i)>180
   AngleDifference(i)=AngleDifference(i)-360;
 end
  TotalAngle=TotalAngle+AngleDifference(i);
end

RotationAngle=TotalAngle/(ccount-1)
if RotationAngle>180
  RotationAngle=RotationAngle-180;
end

if abs(abs(RotationAngle)-180)<90
  RotationAngle=RotationAngle-180
end
end
%******************************END
```

VITA AUCTORIS

Hongmei Gao was born in October 26th, 1968 in P.R. China. She received her Bachelor's Degree in Engineering and Master's Degree in Engineering from the Department of Automatic Control in Beijing University of Aeronautics and Astronautics, respectively in 1990 and 1993. She is currently a candidate for the Master of Applied Science Degree in the Department of Electrical and Computer Engineering at the University of Windsor and hopes to graduate in summer 2001.