A Shape Recognition Scheme Based on Real-Valued Boundary Points

D. J. Buehrer, C. S. Lu and C. C. Chang

Department of Computer Science and Information Engineering, National Chung Cheng University, Chaiyi, Taiwan 621, R. O. C.
E-mail: ccc@cs.ccu.edu.tw

Abstract

An efficient shape recognition scheme based on three new ideas is presented. First, a new method is presented for estimating real-valued boundary points from the integer-valued coordinates and the grey-level values of binary images. Secondly, we can quickly determine all of the feature points with this representation by using an angle calculation formula and the separating technique which are also proposed in this paper. Then, each shape is represented by an array of pairs, where each pair contains the coordinate of a feature point and its distance from the centroid. Thirdly, in the matching process, we also propose a new split-merge technique to assist in the shape matching. The effectiveness of the shape recognition scheme is clearly proven by the good recognition rates of our experiments.

Key Words: Subpixel accuracy, real-valued coordinate representation of boundaries, feature point, shape representation, shape matching, split-merge, scaling factor, skewing

1. Introduction

Shape recognition systems will be extensively applied in several fields to give machines the capability of knowing what they see. One of the applications is a quality control task which involves determining whether the products are the same as a model or not. The recognition work becomes more complicated when the products presented in the transport line are in different directions and positions. The purpose of this paper is to develop an efficient method for shape recognition.

A picture (an object and it's background) is composed of a 2-dimensional array of pixels, each of which has a grey level value. The larger the grey level is, the brighter the pixel is, and vice versa. The common nature of existing methods is to segment the grey level image (i.e., extract objects from the background) by means of binary thresholding. The grey level image then becomes a binary image after thresholding. In the binary image, every pixel is either bright (whose grey level is higher than the threshold) or dark (whose grey level is lower than the threshold). The extracted object (binary image) approximates its original real object coarsely, but may lose local important information or gain extra local noise. Information is also lost by reason of the integer-valued coordinates of the pixels.

Therefore, some smoothing methods are needed to improve the border approximation. Our real-valued coordinate contour representation, which directly approximates the original object pixel by pixel, is such an improvement. We call this improvement "subpixel accuracy".

The motivation for the real-valued coordinate representation of boundaries is described next. A digital image is composed of many grids, with grey levels (where each pixel is either bright or dark) located at integer-valued coordinates, as shown in Fig.1.1(b). It is distorted and divided into some jagged segments relative to its original image. But, boundary points with real-valued coordinates will look like Fig. 1.1(c), which has smaller jags. In other words, the thresholded image with little distortion will have less influence on the recognition work.
The "subpixel accuracy" concept has been widely studied in registration [7, 12, 17] and edge detection [2, 3, 9, 16] problems. All of these above published papers want to achieve the goal of subpixel accuracy. In this paper, real-valued coordinates are used to represent a contour that approximates the original shape to subpixel accuracy. These real-valued coordinates are generated by making use of the grey-level information contained in the boundary points and their neighboring points. Therefore, our real-valued coordinate representation plays a key role in shape recognition.

There are two other major parts to shape recognition besides the reprocessing. The first is shape representation. Its goal is to categorize the identifiable classes of input data via extraction of the image captured from the cameras or scanners. We have proposed a "separating" technique for this aspect of shape recognition. The second part is shape matching. This is the process of checking whether these recognized objects are similar to the pattern shape or not. The matching methods that are proposed are based upon the related shape representations. Here, we also propose a "split-merge" technique to assist in the matching work.

The remainder of this paper is organized as follows. Real-valued coordinate representations of boundaries will be discussed in Section 2. Section 3 gives the definition of feature points and describes how to detect them. Here, we also present a separating technique to find all of the feature points. In addition, we also introduce the shape representation method. Shape matching and an associated split-merge technique are discussed in Section 4. Experiments and results are given in Section 5. Finally, conclusions are given in Section 6.

2. Real-Valued Coordinate Representation of Boundaries

The preprocess of the shape recognition scheme is boundary extraction. We use the raster-scan mechanism to extract the first boundary point from the bottom-left of the image and then apply the eight-connected boundary following algorithm [21] to obtain a series of connected boundary points in a counterclockwise direction. In the process of boundary point extraction, there is some information generated which we will use to calculate real-valued coordinates, as shown in Fig. 2.1. The arrow indicates the direction from next adjacent background point to next boundary point. The grids labeled 0, 1, 2, 3, 4, 5, 6, 7 are eight neighbors of $\downarrow$. Position 6 is the adjacent background point relative to $\downarrow$. We will explain in the next paragraph.

Let $\downarrow$ denote the current boundary point. The point whose arrow comes into the current boundary point is called an adjacent background point (ABGP) relative to the current boundary point. The inner point is in the interior part of the object, defined as the point that $\downarrow$ wants to move toward. For convenience, the digits 0, 1, 2, 3, 4, 5, 6, 7 represent the possible boundary point positions. There is one of the four directions $\rightarrow$, $\uparrow$, $\leftarrow$, $\downarrow$ reaching to the next boundary point. Characters E(East), W(West), S(South), N(North) are given to represent the four directions from the next adjacent background point to the next boundary point. For example, the arrow $\rightarrow$ which is located between positions 7 & 0, indicated in Fig. 2.1, shows that if position 0 is the next boundary point relative to the current boundary point $\downarrow$ then position 7 is the adjacent background point relative to position 0.

Therefore, the possible spatial relations of the three points $\square$ (ABGP), $\downarrow$ (CBP), $\bullet$ (IP) are shown in Fig. 2.2. Note that the current boundary point $\downarrow$ is located at the center of the 3*3 window.
In our method, evaluations of a current boundary point's real-valued coordinate correlate with the direction from the adjacent background point to the current and next boundary points, respectively. These two directions are indicated by the above E, W, S, N. There are also some occasions when the real values depend upon the position of the current boundary point ▲ relative to the previous boundary point ■, and the position of the next boundary point ▼ relative to the current boundary point ■.

Now, an analysis of our method is described below. Here are some symbols and variables that we use:

(a). ▲ denotes the previous boundary point relative to ■.
(b). ▼ denotes the next boundary point relative to ■.
(c). d1 denotes the direction coming into to the current boundary point from its adjacent background point.
(d). d2 denotes the direction coming into to the next boundary point from its adjacent background point.
(e). d=d1 ‧ d2, where ‧ is the concatenation operation.
(f). p1 denotes the position of ■ relative to ▲.
(g). p2 denotes the position of ▼ relative to ■.
(h). p=p1 ‧ p2, where ‧ is the concatenation operation.
(i). (Cx, Cy) denotes the coordinate of ■.
(j). Cgl denotes the grey level of ■. Ngl denotes the grey level of ●.
(k). T is the threshold separating the object and its background.
(l). Density = (Cgl-T)/(Ngl-T).

The variable "Density" is used to express the spatial relationship between CBP and IP. The bigger the Density is, the farther apart CBP and IP are, and vice versa. This is the spirit of the propose real-valued coordinate representation of boundaries. In order to accurately approximate the original shape (without resorting to higher resolution), real-valued coordinates are necessary. As shown in Fig. 2.3, let ◇ and ◆ be the boundary points located at integer-valued coordinates as the dashed arrows indicate. They want to move toward their inner point ■. Symbols □ and □ denote the locations of these boundary points by means of the real-valued coordinate representation. In Fig. 2.3(a), the proportion variable Density determined by the grey-level value is small, so the boundary point approximates the inner point. However, in Fig. 2.3(b), the boundary point has a grey-level value which is almost as dark as that of the inner point, so the variable Density is large, and the boundary point is thus far from the inner point. The dash lines denote the locations that these boundary points should be located in the conventional integer-valued coordinate representation. From Fig. 2.3, we know that the real-valued coordinate representation of the boundaries is a more accurate representation than the integer-valued coordinate representation. Therefore, we lay stress on the importance of this preprocessing work.
Fig. 2. 4. (1). d=EE, RCx=Cx, RCy=Cy+1-Density

Fig. 2. 4. (2). d=NN, RCx=Cx-1+Density, RCy=Cy

Fig. 2. 4. (2). d=NN, RCx=Cx-1+Density, RCy=Cy
Fig. 2. 4. (3) d=WW, RCx=Cx, RCy=Cy-1+Density

Fig. 2. 4. (4) d=SS, RCx=Cx+1-Density, RCy=Cy
Fig. 2. 4.(5). (a) \( p=02 \), \( \text{RC}_x=C_x-1+\text{Density} \), \( \text{RC}_y=C_y+1-\text{Density} \); (b), (c), (d) Any \( p \) except 02 \( \text{RC}_x=C_x-1+\text{Density} \), \( \text{RC}_y=C_y \)

Fig. 2. 4.(6). D=NW, (a) \( p=24 \), \( \text{RC}_x=C_x-1+\text{Density} \), \( \text{RC}_y=C_y+1-\text{Density} \); (b), (c), (d) Any \( p \) except 24 \( \text{RC}_x=C_x \), \( \text{RC}_y=C_y+1-\text{Density} \)
Fig. 2. 4. (7). $d = WS$, (a) $p = 46$, $RC_x = C_x + 1 - \text{Density}$, $RC_y = C_y - 1 + \text{Density}$; (b), (c), (d) Any $p$ except 46 $RC_x = C_x + 1 - \text{Density}$, $RC_y = C_y$

Fig. 2. 4. (8). $D = SE$, (a) $p = 60$, $RC_x = C_x + 1 - \text{Density}$, $RC_y = C_y + 1 - \text{Density}$; (b), (c), (d) Any $p$ except 60, $RC_x = C_x$, $RC_y = C_y + 1 - \text{Density}$
Fig. 2. 4. (9). d=NE, RCx=Cx-1+Density, RCy=Cy

Fig. 2. 4. (10). d=WN, RCx=Cx, RCy=Cy-1+Density

Fig. 2. 4. (11). d=SW, RCx=Cx+1-Density, RCy=Cy

Fig. 2. 4. (12). d=ES, RCx=Cx, RCy=Cy+1-Density

Fig. 2. 4. (1)~(12) are heuristic formulas for the real-valued coordinate representation of boundaries.
the spatial relationships between the previous boundary point, the current boundary point and the next boundary point. All the possible tendencies are fully detailed in Fig. 2.4(1) ~ (12). Fig. 2.4(1) ~ Fig. 2.4(4) and Fig. 2.4(9) - Fig. 2.4(12) have similar conditions. (Cx, Cy) is the integer-valued coordinate of the current boundary point ■, and (RCx, RCy) is its real-valued coordinate approximating the inner point ●, achieving subpixel accuracy. The four figures 2.4(5) ~ 2.4(8) all have a similar condition attached to them. They need also the information p in order to determine the real-valued coordinates, besides the information d. For these four figures, each subfigure (a) is different from subfigures (b), (c), (d). The formulas show the heuristics that we use.

Each time we arrive at a boundary point in the process of boundary following, the real-valued coordinate of the previous boundary point could be evaluated by the above heuristic formulas. We will go back to the first boundary point at the end of the boundary following. At this moment, the location of the final boundary point is evaluated and all the boundary points are represented with real-valued coordinates. The preprocess of shape recognition is thus completed.

3. Feature Point Extraction and Shape Representation

3.1 Feature point extraction

We will introduce our feature point extraction method in this subsection. Based on the real-valued boundary point representation, the algorithm is very simple and straightforward, using only an angle calculation formula.

Now, we define a so-called "feature point". The feature points have two forms in this paper. The first form is defined as any point with a sharp vertex angle < 150° (found experimentally to be a good value) called a corner point [4, 5, 6, 10, 11, 13, 14, 15, 18, 19], found by the angle calculation formula shown in Fig. 3.1. The second kind of feature points are those points whose distances to curve L are local maxima or local minima, where L is the curve with the two adjacent sharp vertex angle points (corner points) as end points.

First of all, our algorithm takes the boundary points generated in Section 2 as input. Suppose these points are pi = (xi, yi), 1 ≤ i ≤ n. For any pi we choose p_{i-2}, pi and p_{i+2} to compute the angle between the two segments (p_{i-2}, pi) and (pi, p_{i+2}). If the angle is less than 150°, the point pi is a possible corner point. We consider the neighborhood of pi which has a series of possible corner points, and choose the point with the smallest angle as the corner point (the first form). This point is absolutely a true feature point because of its small angle. We illustrate this in Fig. 3.1. The boundary point pi with angle \( \theta_1 \) is selected as the feature point because it has the smallest angle among its neighborhood \{p1, p2, p3\}.

Fig. 3.1 An example of selecting the feature point of the first form (corner point).

Next, we have developed a "separating" technique to find the feature points of the second form. The whole algorithm is presented below.

"Separating technique" Algorithm:

**Input:** The corner points generated from the angle calculation, stored in an array.

**Output:** All the possible feature points.

**step(1):** Set k = 1. Declare an (n+2) * 2 array Fp.

**step(2):** While (k ≤ n) do

\begin{algorithmic}
\State Store the k-th feature point into the new array Fp;
\State if the curve (pk, pk+1) needs to be separated then
\State \quad if the point p'k in the curve (pk, pk+1) is a local maximum or local minimum then
\State \quad \quad if the length between p'k and the previous feature point > Tol then
\State \quad \quad \quad \quad (p'k is a feature point & add p'k to the new array Fp)
\State \quad \end{algorithmic}

**end**

**step(3):** For each feature point \( p_i \) do

\begin{algorithmic}
\State Calculate the distance di between the centroid and \( p_i \).
\State Store di and the sequence number s_i of \( p_i \) (i.e., two-tuple (di, s_i)) into another new array.
\State
\end{algorithmic}

We will illustrate the algorithm in detail (see Fig. 3.2). First of all, when should our so-called curve should be separated? The two end points of a curve are both corner points. We connect these two points with a straight line and calculate the
maximum distance from the curve to the line. If the length of the curve is larger than a given tolerance \( Tol_1 \) and the maximum distance over the length is also larger than another given tolerance \( Tol_2 \) then the curve needs to be separated. Secondly, our separating technique is a so-called "incremental and decremental distance evaluation". If the curve is to be separated, we then apply the technique. The incremental and decremental distance evaluation technique calculates the vertical distances from all the points in the curve to the line. We sequentially compare the differences of adjacent pairs of these distances and record the number of increases and decreases. A boundary point \( p \) is said to be at a "local maximum or minimum" if there is a sufficient number of increases (decreases) before \( p \) and enough decreases (increases) after \( p \). These local minima and maxima are the second kind of feature points. After performing the separating process, all possible feature points will be found. In Fig. 3.2, point \( a \) and point \( b \) are the starting point and ending point of the curve \((a, b)\) respectively. We will find the possible feature points \( c \) and \( d \) sequentially by using the separating technique.

The above formula has been simplified and it is very easy, so we won't discuss it here.

### 3.2 Shape representation

From the discussion in Subsection 3.1, we obtain the feature points stored sequentially in an array \( X_c = \frac{1}{n} \sum_{i=1}^{n} x_i \) and \( Y_c = \frac{1}{n} \sum_{i=1}^{n} y_i \) of \((x_i, y_i)\) of an object, crucial in our shape recognition, is defined as:

where \( n \) is the total number of boundary points in the image, and \((x_i, y_i)\) is the coordinate of boundary point \( p_i \).

First, we calculate the distance between each feature point and the centroid. Additionally, in order to correctly describe the position of the feature points and use the split-merge technique discussed in the next section, we assign a sequence number to each boundary point. The sequence number of the \( k\)-th boundary point in a shape which has \( n \) boundary points labeled 1, 2, ..., \( n \) is \( k \), where \( 1 \leq k \leq n \). A more detailed description is given below.

For example, as shown in Fig. 3.3, our pattern shape \( P_3 \) has twelve feature points. The bold lines connect every pair of feature points. The symbol "*" denotes the centroid of \( P_3 \). In Fig. 3.3(a), the dashed line which connects the feature point \( p_i \) and the centroid denotes the distance \( d_i \) and in Fig. 3.3(b), the sequence number of \( p_i \) in the boundary points is \( s_i \) where \( 1 \leq i \leq 12 \). From the sequence number, we can know the relative position of every feature point. So, in this paper, the shape representation is the set of two-tuple elements \((d_i, s_i)\), \( 1 \leq i \leq m \), where \( m \) is the total number of feature points in a shape. For example, the shape representation in Fig. 3.3 is \{\( (d_1, s_1) \), \( (d_2, s_2) \), ..., \( (d_{12}, s_{12}) \)\}.

Suppose \((x_1, y_1)\) and \((X_B, Y_B)\) are the coordinates of \( a \) and \( b \) respectively. The slope of the line is \((y_B-y_1)/(x_B-x_1)\). Now we want to show how to calculate the vertical distance from the point in the curve to the line. Let \((x', y')\) be the coordinate of the point on the curve and \((x, y)\) be the point on the straight line such that the distance between \((x', y')\) and \((x, y)\) is the minimal distance from \((x', y')\) to the line. Coordinate \((x, y)\) and the vertical distance \(vd\) from \((x', y')\) to \((x, y)\) are computed as follows:

\[
\begin{align*}
vd &= \sqrt{\left(x - x'\right)^2 + \left(y - y'\right)^2} \\
x &= x' + y' \times \text{slope} - y \times \text{slope} \\
x_{t} &= x' + y' \times \text{slope} - y \times \text{slope} \\
Y_c &= \frac{1}{n} \sum_{i=1}^{n} y_i
\end{align*}
\]

The above formula has been simplified and it is very easy, so we won't discuss it here.

**Fig. 3.3 Shape representation with two criteria : distance(a) and position(b).**
Where
d_2: the distance between p_i and centroid,
s_i: the sequence number of p_i in the sequenced boundary points,
distance: the shortest length between two feature points,
length: the total number of boundary points between two feature points.

4. Shape Matching

Pattern matching involves matching two shapes. Pattern shapes are called the standard shapes, and the input shapes are obtained from pattern shapes by means of from one to three mixed actions. These three types of actions are rotation, scaling and translation.

In our shape matching, there is also one thing that we must pay special attention to. The size of the input shape relative to the pattern shape will be changed during these actions. The size of an object is defined as the number of boundary points. Knowing the relationship between the size of a pattern shape and the size of the input shape is necessary for the matching work. We define this relationship, called the Scaling Factor (SF), as follows:

\[ SF = \frac{\text{the number of boundary points of the input shape}}{\text{the number of boundary points of the pattern shape}} \]

i.e.,

\[ SF = \frac{\text{the contour length of the input shape}}{\text{the contour length of the pattern shape}} \]

In most shape recognition experiments, "rotation" and "scaling" mean rotation and scaling of an object about its centroid. Then the recognition work will proceed in an ideal way. In fact, real applications and our experiments are not so ideal. If the input shapes are obtained from their pattern shapes by means of three mixed actions (rotation, scaling and translation), then the input shapes are rotated, translated and the parts of them are not uniformly scaled. This situation will be called the skewing problem. The non-uniform scaling is due to the three-dimensional nature of the problem when the centroid moves.

The matching method proposed in this section is independent of translation, rotation and scaling when involving the scaling factor. First, we must record the number of elements in the shape representation arrays of the pattern and input shapes respectively. In the matching process, we must correctly select the starting elements for the shape representation of the pattern and the input shapes for matching. Owing to distortion or noise, or even if these shapes are symmetric, it is difficult to determine the feature point to be selected as the starting point. So, we store these two arrays twice separately, i.e., the new arrays have twice the length of the original ones, and the matching process proceeds circularly.

For the convenience of discussion, let I_{di} denote the first attribute value of the element in the input shape representation array, and let M_{di} denote the first attribute value of the element in the pattern shape representation array. Let |I_{di} - M_{di}| denote the absolute value of the difference between I_{di} & M_{di}. If the input shape and the pattern shape are similar, then all the relative differences |I_{di} / SF - M_{di}| should be within a tolerance. The expression |I_{di} / SF - M_{di}| involves the scaling factor to make sure that our matching algorithm is independent of translation, rotation and scaling. But, when we consider the skewing problem, the change of distances between every feature point and the centroid are not always consistent with the scaling factor. So, it is hard for us to solve this problem. Therefore, to solve the skewing problem we use a larger tolerance for |I_{di}/SF - M_{di}| to tolerate the non-uniform scaling caused by skewing. Beginning at the first elements of the shape representation arrays of the pattern and input shapes, if |I_{di} / SF - M_{di}| is less than a tolerance Tol_3, then we find the two starting matched points and we compare the next elements repeatedly. On failure, we go on to the next element of the input shape representation array and we compare the next elements repeatedly. On failure, we go on to the next element of the input shape representation array. If there is not any element in the input shape representation array that can be matched with the first element in pattern shape representation array, we go on to the next element in pattern shape representation array and restart from the first element in input shape representation array. Therefore, the matching work goes on until the end of the pattern shape representation array. When these two arrays are matched completely, the two shapes are similar.

When we find a pair of starting matched elements, mentioned as in the above paragraph, we compare the next elements of the pattern and the input shape representation arrays. Now, there are two conditions that must be satisfied. The first condition is that the relative distances between the centroid and each feature point in the input and pattern shapes must be within a given tolerance Tol_1. Before discussing the second condition, we define some symbols. Let I_1 denote the length (the total number of boundary points) from the starting matched point to the input shape point.
which will now be matched. Let $M_{l2}$ denote the length (the total number of boundary points) from the starting matched point to the pattern shape point which will also be matched now. The "length" is determined from the second attribute value of elements in the shape representation arrays. We define the "length ratio" as $I_1/I_2$. The ratio must also be less than a given tolerance $Tol_4$ and larger than another given tolerance $Tol_5$ to make sure that all of the feature points are located at the proper positions in a shape. This is the second condition for matching. The pattern and input shapes are then similar if all the matches satisfy these two conditions. When one of the conditions is not satisfied, the split-merge technique is applied to assist our matching work. Now we state the split-merge technique as follows:

If $I_1/I_2$ is less than a given tolerance $Tol_5$, then we apply the merge technique. We illustrate as follows.

In Fig. 4.1 and Fig. 4.2, $s$ and $s'$ are starting points, and apparently we know that $a$, $a'$, $b$, $b'$, $c$ & $c'$ are matched pairs. But in our program, when we try to match $b$ and $b''$, the value of $I_1/I_2$ is less than a given tolerance $Tol_5$. Therefore, we need to perform a merge operation. The original two segments $(a', b'')$ and $(b'', b')$ are merged into one curve $(a', b')$. That is, the element corresponding to the point $b''$ is deleted from the input shape representation array. A part of the input shape can now match this part of the pattern shape. The causes of the extra generated feature point $b$ may be due to distortion or noise, or even to the non-perfectness of the separating technique in the feature point extraction process. But we delete these undesirable situations by means of our merge operation.

If $I_1/I_2$ is larger than a given tolerance $Tol_4$, we apply the split technique discussed below. For the sake of clarity, we illustrate by Fig. 4.3(a), (b), (c). Let $*$ denote the correct position of the feature point following $a'$, i.e., the position at which the point $b'$ should be located.

In the above possible parts of an input shape, we want to know how far apart $b'$ and $c$ are to decide whether to directly split curve $(a', b')$ or to first delete $b'$ (using a merge operation) and then split the curve $(a', c')$ in order to get the correct position of $b$. In Fig. 4.3(a), the distance between $*$ and $b'$ is closer, so it may be hard to find the correct position of $*$ because of the short distance between $*$ and $b'$. So, we first delete $b'$ and then split the curve $(a', c')$. If we have the case in Fig. 4.3(b) or Fig. 4.3(c), we don't know whether $b'$ is close to $c'$ or not, and the distance between $*$ and $b'$ is large enough, so we directly split curve $(a', b')$ to get the correct feature point.

If we can't find a correct feature point to match $b$ in the pattern shape, we judge that the two shapes are not matched from the starting matched point $s$. 

![Fig. 4.1](image1.png) A part of the sample shape with some feature points.

![Fig. 4.2](image2.png) The situation where an input shape should be merged.

![Fig. 4.3(a)](image3.png) Some feature points in an input shape.

![Fig. 4.3(b)](image4.png) Some feature points in an input shape.

![Fig. 4.3(c)](image5.png) Some feature points in an input shape.
and s'. Therefore, we circularly select the next point in the input shape as the starting point for matching. If the last input feature point has already been selected, then we select the next possible starting point in the pattern shape and select the first point in input shape to restart the matching work.

Matching algorithm with "Split-Merge technique":

**Step(1)**: /* npattern and ninput are the length of pattern and input shape representation array before duplication */
Store the input & pattern shape representation arrays twice.

**Step(2)**: /* pattern_starting_pt and input_starting_pt denote the index of the first matched elements in the pattern and input shape representation arrays, respectively */
Initialize pattern_starting_pt and input_starting_pt

**Step(3)**: While (pattern_starting_pt ≤ npattern)
begin
/* increment starting points */
find next matched(input_starting_pt, pattern_starting_pt);
/* input_index and pattern index point to the currently matched elements of the input and pattern shape representation arrays, respectively */
make a copy of input shape representation array, set input_index = 1;
flag = TRUE;
while(flag & pattern_index < npattern + pattern_starting_pt)
begin
  input_index = input_index + 1;
  pattern_index = pattern_index + 1;
  flag == match(input_index, pattern_index);
if (not flag) then
  begin
    Call split-merge;
    flag = match(input_index, pattern_index);
  end
end
if (flag = TRUE) then
  return (TRUE); /* the input shape is similar to its pattern shape */
  pattern_starting_pt = pattern_starting_pt + 1;
  input_starting_pt = 1;
end
**Step(4)**: return(FALSE).
/* the input shape isn't similar to its pattern shape */

In the shape matching process, there are some tolerances defined by us which are influential in our recognition rate. These tolerances listed below are obtained by experimentation:

\[
\begin{align*}
\text{Tol}_1 &= (5 \times SF) \text{ pixels;} \\
\text{Tol}_2 &= 0.04; \\
\text{Tol}_3 &= 11 \text{ pixels;} \\
\text{Tol}_4 &= SF + 0.065; \\
\text{Tol}_5 &= SF - 0.065;
\end{align*}
\]

The scaling factor (SF) is involved in these tolerances, so our matching algorithm is independent of rotation, scaling and translation. \(\text{Tol}_1\) forces the length between every pair of feature points to be bigger than a certain value. Whether we need to use the separating technique for feature point extraction or not depends on the curvature of the curve. The degree of curvature is defined as

\[
\frac{\text{maximum vertical distance}}{\text{length of curve}}
\]

If the degree of curvature is larger than \(\text{Tol}_2\), then we apply the separating technique. The first condition for matching is that corresponding distances to the centroid must be within \(\text{Tol}_3\). We also need the last two tolerances to make sure that the matched feature points in the input shape are located in the proper positions relative to the positions of the feature points in the pattern shape. The first two tolerances are the satisfaction conditions used by the separating technique, and the others are used in the pattern matching algorithm.

5. Experiments and Results

In order to show the feasibility of our shape recognition scheme, we take three kinds of hammers and four kinds of pliers as our pattern shapes. These seven pattern shapes, which come from [8, 20], are shown in Fig. 5.1. To be less affected by distortion or noise, we take pictures in a well-controlled environment. The resolution of our grey-level images taken by a SONY CCD video camera is 200 * 256. From the experimental results, we prove that our method is still good in such a low resolution environment. The proposed scheme is implemented on a 32-bit 80386 PC with Turbo C 2.0.

**Experiment I**

In the first experiment, these input shapes are obtained from their pattern shapes by scaling. But slight translations occur because of non-uniform scaling.

**Experiment II**

In the second experiment, the input shapes are obtained from their pattern shapes by using the three mixed actions. The centroid of the input shapes are different from that of their pattern shape due to translation. The rotated angles of the input
Fig. 5. Pattern shapes for shape recognition
Fig. 5. 2. ■ denotes the pattern shape, ○ and ● denote the input shapes in Experiment Ι and Experiment Π respectively: (a)–(g) show the locations of the input shapes relative to their pattern shapes, H1–P4.

Table 5. 1. Contour length of each pattern shape

<table>
<thead>
<tr>
<th>Type</th>
<th>H1</th>
<th>H2</th>
<th>H3</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Length of Each Pattern Shape (Unit : pixel)</td>
<td>405</td>
<td>390</td>
<td>386</td>
<td>491</td>
<td>476</td>
<td>371</td>
<td>487</td>
</tr>
</tbody>
</table>
shapes range from 0° to 360°. Owing to translation and rotation, the parts of the input shapes are not uniformly scaled. This is equivalent to seeing objects from different directions and locations. The locations of the centroids and the scaling factors of the input shapes relative to those of their pattern shapes are shown in Fig. 5.2(a) ~ Fig. 5.2(g) and Fig. 5.3(a) ~ Fig. 5.3(g) respectively. But the input shapes' rotational angles, which are uniformly distributed between 0° and 360°, are not shown here.

Our shape recognition scheme first uses real-valued coordinates to represent the boundary points generated by the contour following algorithm. Then, we calculate the angles by means of the inverse arctan function to find the points with sharp angles. These points are certainly feature points. We also use the separating technique discussed in Section 3 to find all other feature points. Finally, the split-merge technique discussed in Section 4 is used to assist in the matching work. The real-valued coordinate representation, with the separating technique and the split-merge technique play essential roles in our shape recognition scheme.

For the convenience of the experiments, we fix the camera and rotate or scale the object. This gives the same result as moving the camera to see the object from different directions or distances. The contour length of each pattern shape is given in Table 5.1. Recall that the resolution in our experiments is 200 * 256.

The recognition rate is defined as

\[
\text{recognition rate} = \frac{\text{number of the recognized shapes}}{\text{number of input shapes}}
\]

For all 226 input shapes, there are seven input shapes that don't match their original pattern shapes.

The average recognition rate thus is about 97%. The amount of skewing which occurs in the first experiment is not serious. However, we find that all of these seven un-recognized shapes, which come from the second experiment, are highly skewed. If we would resolve the skewing problem then we could get a higher average recognition rates.

6. Conclusion

The good experimental results acquired have shown the efficiency of our recognition method. The grey level of each pixel in the object's contour is important for our real-valued boundary point representation and will be affected by illumination or reflection in our experimental environment. That is, bad illumination and reflection affect our real-valued coordinate calculations. In order to get a better boundary, we use a fairly standard trick to solve the problem. We get the boundary of an object first, and for each boundary point p we choose its right and left neighboring boundary points to calculate their (including p) average coordinate to replace the coordinate of p. From experimental results, we know that this trick is reasonable and acceptable.

Some good two-dimensional shape recognition methods [1, 20] have been presented by other authors. But their scaling factors are fixed to certain values, which are larger than or equal to 1. Our experimental conditions are very similar to real world conditions. Experiment I had translation and scaling only, whereas Experiment II also included rotation. All the input shapes are translated and randomly enlarged or reduced to reasonable degrees, and our rotational angles range from 0° to 360°. The recognition method we have proposed in this paper can effectively determine whether a pattern shape and an input shape are similar or not.

7. Summary

An efficient and simple shape recognition method has been developed in this paper. First, we describe how to represent the boundary of an object by means of real-valued coordinates. With this representation, we can easily determine the corner points using an angle calculation formula. These corner points are truly feature points. Next, an "incremental and decremental distance evaluation" separating technique is presented to help us find all of the other feature points. The split-merge technique is developed to assist in the matching work. The real-valued boundary point representation, the separating technique and the split-merge technique play major roles in the shape recognition process. Algorithms for these three techniques are also proposed. The experimental results show that this shape recognition method is effective and feasible for recognizing two dimensional shapes taken in a low-noise environment.
Fig. 5.3 ■ denotes the pattern shape, ○ and ● denote the input shapes in Experiment I and Experiment II respectively: (a)–(g) show the scaling factors of the input shapes relative to their pattern shapes, H1–P4.
Reference


Manuscript Received: May. 26, 2000
Revision Received: Jun. 19, 2000
And Accepted: Jun. 19, 2000