

To the Graduate Council:

I am submitting herewith a thesis written by Fernando Rodrigo Rannou entitled "Equilateral Polygon Approximation of Closed Contours." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor
Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

Michael G. Shomerson

Bradley Vander Zanden

Accepted for the Council:

Cowminkel
Associate Vice Chancellor

and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature



Date

11/17/93

EQUILATERAL POLYGONAL APPROXIMATION
OF CLOSED CONTOURS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Fernando Rodrigo Rannou
December 1993

To Jenny

Acknowledgments

I would like to express my highest gratitude to my advisor Dr. Jens Gregor, who truly supported me in all aspects of this work. His guidance and assistance made this work possible. Appreciation is also extended to the other members of my committee, Dr. Michael G. Thomason and Dr. Bradley Vander Zanden.

Gratitude is also offered to the Departamento de Ingeniería en Informática at Universidad de Santiago de Chile for giving me the opportunity to continue my education.

I would like to thank the members of my family. Special thanks go to my father, who has been doing all the paperwork to make my life easier.

Finally, I want to thank my wife Jenny, whose love is invaluable support in my life.

Abstract

Polygon Approximation is the process of representing a general contour with a piecewise linear function. When the length of each linear segment is constant, the result is an Equilateral Polygonal Approximation.

This thesis proposes a first attempt to solve the Equilateral Polygonal Approximation Problem for closed contours. The solution is based on Nonlinear Programming and the approximation is modeled as an optimization process. The constant length restriction is simply handled as a set of nonlinear constraints. Four objective functions are presented: a least-squares function, two area-based functions, and an energy function.

The energy function is presented as the major contribution of this work. It considers each polygon vertex as being pulled by a set of linear springs connected to the contour and the optimal solution is achieved when this system reaches its equilibrium state.

Experimental results are reported for a set of fourteen tools using the energy method. Approximations for five different segment lengths are derived for each tool. Two sets of experiments are carried out over all the approximations obtained. The first experiment is intended to assess the quality of the approximations, that is, to measure how closely they resemble original contours. The results of this experiment indicates that the approximations are highly satisfactory for small segment lengths. The other experiment measures the classification capabilities of the approximations using string matching by dynamic programming. The main

conclusion that can be drawn from this experiment is that the segment length plays an important role when the method is used in pattern recognition.

Contents

1	Introduction	1
1.1	Objective of the study	3
1.2	Outline of the thesis	3
2	Polygonal Approximation	5
2.1	Introduction	5
2.2	Polygon approximation by splitting	7
2.3	Polygon approximation by linear scan	9
3	Equilateral Polygonal Approximation	13
3.1	Computing an initial guess	17
3.2	A least-squares function	19
3.3	The area as main feature	22
3.3.1	Global minimization	23
3.3.2	Minimizing local areas	24
3.4	An energy function and an optimal solution	28
3.4.1	Motivation	28
3.4.2	Formulation	29
3.4.3	Computation of the number of segments	32

3.4.4	Variations to the original model	35
3.5	Concluding remarks	37
4	Support Functions	39
4.1	Contour following	39
4.2	Contour coding	42
5	Experimental Results	46
5.1	Introduction	46
5.2	Data material	47
5.3	Results and discussion	50
5.4	Experiment I: Qualitative assessment	56
5.4.1	Results	58
5.4.2	Discussion	61
5.5	Experiment II: Classification performance	62
5.5.1	Method	62
5.5.2	String matching	64
5.5.3	Results and discussion	65
6	Conclusion	72
	Bibliography	76
	Appendices	80
A	Khoros	81
A.1	Introduction	81
A.2	Programming in Khoros	84
B	Nonlinear programming	90

List of Tables

5.1	Values for microclass P_1	59
5.2	Values for microclass P_3	59
5.3	Values for microclass S_3	59
5.4	Values for microclass W_1	59
5.5	Confusion table for PA's with length 5 and quantization level 1 .	66
5.6	Confusion table for PA's with length 7 and quantization level 1 .	66
5.7	Confusion table for PA's with length 7 and quantization level 1 .	66
5.8	Confusion table for PA's with length 11 and quantization level 1 .	67
5.9	Confusion table for PA's with length 13 and quantization level 1 .	67
5.10	Confusion table for PA's with length 13 and quantization level 1 .	69
5.11	Confusion table for Macroclass W	69
5.12	Confusion table for Macroclass W	69

List of Figures

2.1	Maximum-distance criterion to create new segments. Segment $\overline{s_i s_j}$ is split into $\overline{s_i s_l}$ and $\overline{s_l s_j}$ when the orthogonal distance from point s_l to it is greater than a constant value.	8
2.2	Tree representation of the splitting process. An initial segment AF in (a) is sucessively split until a distance criterion is satisfied. (b) shows the order with which the approximation is carried out. . . .	8
2.3	Algorithm for PA by splitting	10
2.4	Cone intersection method. (a) Data point s_j satisfies error criterion; (b) TOP_{min} and $BOTTOM_{max}$ cross each other and a new segment is created	11
2.5	Algorithm for PA by linear scan	12
3.1	A discrete grid not always work to assign polygon points. Neither s_4 , s_5 , and s_6 are at a distance 2 from point s_3	14
3.2	Example of how polygon vertices are interconnected because of the length constraint.	15
3.3	Finding the next polygon point by a simple walk along the contour. . .	17
3.4	Computation of the last polygon point for the initial estimate. . .	18
3.5	Curve points are orthogonally projected on their respective segments. . .	20

3.6	Computing if c_j is in R_i . In this case $c_j \in R_i$ because its projection s lies between the two end points of the segment.	21
3.7	Defining circular regions to constrain area minimization. A polygon vertex can only be located inside its circular region.	24
3.8	Curve points are considered as being polygon vertices. In (a) no intersections occur among segments. In (b) segments $\overline{bc}$ and $\overline{cd}$ intersect segment $\overline{s_i s_{i+1}}$	25
3.9	All contour points p inside the polygon are rotated out.	26
3.10	When computing the area enclosed between the contour and the segment, the points must be taken in the direction shown in the figure.	27
3.11	Resulting sequence of contour, intersection, and polygon points for previous example.	28
3.12	A vertex point is attached to only a subset of contour points . . .	30
3.13	General case for computing potential energy of a spring	31
3.14	For some cases, a cluster effect is obtained in the final PA when the initial polygon is computed using previous algorithm.	32
3.15	Example of how a segment cannot be appropriately fit to the contour. 35	
3.16	Two adjacent regions R_{i-1} and R_i are merged if ϕ is less than a given fixed value.	36
4.1	Contour following algorithm.	40
4.2	8-Neighborhood and associated directions.	41
4.3	Illustration of Contour Following. c_{i+1} is found by a clockwise search in the neighborhood of c_i	42
4.4	θ_i is the internal angle between two segments.	43
4.5	Computing angle θ_2	44

4.6	Contour coding algorithm.	45
5.1	Some of the objects used in the experiments: pliers 1 and 2 (P_1 , P_2), wrenches 4 and 5 (W_4 , W_5), and screwdrivers 1 and 3 (S_1 , S_3).	49
5.2	Pliers 1, 2, and 3 approximated with lengths 5 and 11	51
5.3	Screwdrivers 1, 2, and 3 approximated with lengths 5 and 11	52
5.4	Screwdrivers 4, 5, and 6 approximated with lengths 5 and 11	53
5.5	Wrenches 1, 2, and 3 approximated with lengths 5 and 11	54
5.6	Wrenches 4 and 5 approximated with lengths 5 and 11	55
5.7	Variation of μ_s and σ_s^2 with respect to the segment length of an approximation. Data are averages over all 7 microclasses.	60
5.8	Variation of μ_{xx} , μ_{yy} , and μ_{xy} with respect to the segment length of an approximation. Data are averages over all 7 microclasses.	60
5.9	Total classification rates for the three macroclasses when prototype and unknown strings have the same segment length.	68
5.10	Classification rates when the segment length of unknown string is maintained fixed. The values shown are totals over all possible segment lengths for prototypes.	70
A.1	Khoros architecture	82
A.2	The user interface window for vadd	86

Chapter 1

Introduction

One purpose of a pattern recognition system is to identify unknown objects on basis of a given set of pattern models or classes. There are many approaches to deal with this problem. In decision-theoretic methods a set of numerical features are extracted from the object, and the recognition is carried out using statistical or geometrical techniques. In structural approaches objects are modeled as an ordered set of structural elements, usually called primitives, and unknown samples are recognized by matching or parsing their structural elements against a relational model or grammar respectively.

For some applications, the recognition process is simplified by using information extracted only from the shape of the object. Gray levels and/or frequency properties are not taken into account. This is very common in automatic inspection of industrial parts where all the objects are manufactured using the same materials. Cases where the shape could be irrelevant are for instance texture analysis or where spectral analysis is required. For two-dimensional problems the object is conveniently represented only by its contour. Since a contour can consist of thousands of data points, piecewise linear approximations are applied in order

to reduce the amount of data. This is important for reason of computational efficiency not only in representation but also in recognition. The purpose of Polygonal Approximation (PA) is to produce such piecewise linear approximations.

After a polygonal approximation is done, we need to code in some way this representation in order to obtain a suitable input for the matching algorithm we have chosen. A common coding scheme is a sequence of pairs (l_i, θ_i) , where l_i and θ_i respectively denote length and angle. Specifically, l_i could be the length from the object centroid to vertex i and θ_i could be the angle formed by the two rays from the centroid to two consecutive vertices i and $i + 1$. This coding is very powerful in terms of descriptive capabilities because we can completely reconstruct the polygon from the coding. However, it is not suitable if we want to use classical string matching algorithms for the recognition phase. For this class of algorithms [17] both the prototype (model) and the pattern must be represented as a sequence of symbols and not as a sequence of pairs of symbols.

There are at least three solutions to this problem: (1) define a function f such that $a_i = f(l_i, \theta_i)$, (2) use an attribute string matching algorithm or (3) create a polygonal approximation algorithm such that all the polygon sides have the same length. The first solution would give us a fast way of mapping a polygon to a sequence of symbols. The real problem lies in that the function must be onto, that is, no two different points on the plane may map to the same symbol. The second solution has been previously investigated by some authors. In particular the reader is referred to [16], where polygons are dynamically modified to guide the matching process. The disadvantage in this case is that the string matching is highly dependent on the polygons and therefore not suitable for using faster string matching algorithms. The third solution has not been studied in any previous work. One of the principal advantages of method (3) is that we would be able to

apply the classical algorithms as well as faster and more advanced algorithms that exist [3].

1.1 Objective of the study

The main objective of the present work is to develop a new method to approximate any closed contour by a polygon such that all its segments have the same length. We will call this problem the *Equilateral Polygon Approximation Problem*.

The solution presented is based on Nonlinear Programming (NLP). The PAs are obtained by minimizing appropriate objective functions subject to constraints. It is also a very important part of the study assessment of the method. Two types of evaluations are made: a qualitative assessment and classification performance. We expect these experiments to give us a measure of merit of the solution and more importantly new insights about the problem itself to guide future research.

1.2 Outline of the thesis

The thesis is organized into 5 chapters. A brief review of the literature is presented in Chapter 2. In order to understand the problem more clearly, two specific algorithms are presented in detail. These algorithms address the PA in two different ways. A brief note about complexity is given.

Chapter 3 deals with the equilateral polygon approximation. A discussion is made to support the validity of using NLP. Concepts of NLP are also introduced. Four solutions are proposed and discussed. We consider the last solution as the main contribution of this work.

Some support routines had to be developed and implemented during the devel-

opment of the work. The two most important are described in Chapter 4. They are a contour following routine and a contour coding routine.

The final solution to the EPA problem is evaluated in Chapter 5 which describes two experiments. The first experiment assesses the "quality" of the PA's obtained for different objects. The second experiment measures the classification performance that can be obtained by using strings generated from equilateral polygons.

Conclusions and directions for future reasearch are made in Chapter 6.

Appendix A describes in general terms the characteristics and principal components of Khoros, the software for visualization into which many of the routines were integrated.

In order to carry out the optimization itself a software tool for numerical problems was used. Appendix B describes in mathematical terms specific details of the particular routine used.

Chapter 2

Polygonal Approximation

2.1 Introduction

The field of polygonal approximation has been studied during the past two decades. The first paper on piecewise approximation is due to Stone [15], who presented the least-squares method to approximate an open one-dimensional contour by a piecewise linear function. For image processing and shape analysis, algorithms have been developed to handle closed contours ever since. One way of classifying these algorithms is according to the method used to walk along the contour, namely PA by splitting and PA by linear scan. In PA by splitting a contour (or a section of it) is approximated by successively splitting existing segments. PA by linear scan decides the creation of vertices “as we go” walking on the contour.

There are many other algorithms that combine these two approaches. They differ mainly in the criterion used to drive the creation of new vertices. For instance, Montanari [9] and Sklansky et al. [13] find minimum-perimeter polygons. Montanari uses nonlinear programming to minimize the perimeter of the polygon such that the coordinate vertices of its segments are constrained to lie in circular

regions centered very close to the contour. Sklansky's method is based on the rubber band paradigm which states that the rubber band will take the shape with minimum perimeter when its elastic energy is minimized. Ramer [11] presents an iterative approach to split segments when a criterion of distance to the contour is satisfied. Pavlidis and Horowitz [10] proposes a hybrid approach of merging and splitting. A segment can be split or adjacent segments can be merged such that an error criterion is satisfied. Williams [19] finds the longest segment such that all subsegments with the same initial point are constrained to pass within specified distance from the data points. A similar strategy is followed by Sklansky and Gonzalez [14]. Kurozumi and Davis [8] presents a minimax method to find an approximating polygon with the minimum number of segments such that the distance from the data points to the sides is less than a specified constant. The authors claim that their algorithm is the optimal for polygonal approximation. Dunham [1] also finds a polygon with the minimum number of segments such that the sides are required to approximate the contour within a uniform error (L_∞ norm) with fixed initial and final points. A modified dynamic programming approach is used in this case. Recently, Ray and Ray [12] find the longest possible segment with the minimum possible absolute error (L_1 norm) deviation from the data points. An area criterion is used by Wall and Danielson [18]. They use a scan-along algorithm to output a new segment when the area deviation per length unit of the current segment exceeds a specified constant.

In the next two sections we will review in detail two algorithms for polygonal approximation that are good examples of the two approaches used to walk along the contour: PA by splitting and PA by scan-along.

2.2 Polygon approximation by splitting

Ramer [11] describes an algorithm to approximate a contour by a polygon using the splitting approach. The algorithm takes an ordered set of points representing a digitized contour, and by means of an iterative procedure it partitions this set into subsets according to a distance criterion. That is, if the maximum distance from any point in the set to a line segment joining the first and last points of the set is greater to a predefined threshold, the set is splitted into two subsets.

Starting with the entire contour in the set $S = \{s_0, \dots, s_{n-1}\}$, the algorithm chooses two points from it to create the first segment. If the contour is open s_0 and s_{n-1} are chosen, but if the contour is closed the top left-most point and the bottom right-most point are chosen for these extremal points. Thus the initial polygon consists of only two points and one segment.

Suppose that in the k -th iteration we have a subset $S_{ij}^k = \{s_i, s_{i+1}, \dots, s_j\}$ which is to be tested against the maximum-distance criterion (see Figure 2.1). Let also $L_{ij}^k = \overline{s_i s_j}$ be the line segment between s_i and s_j and $d(s, L_{ij}^k)$ the distance between any point $s \in S_{ij}^k$ and L_{ij}^k . Let s_l be the most distant point in S_{ij}^k to the segment, that is

$$d(s_l, L_{ij}^k) \geq d(s_r, L_{ij}^k) \quad s_l, s_r \in S_{ij}^k$$

Then, if this distance is greater than a constant threshold α , the segment is partitioned into the new sets $\{s_i, \dots, s_l\}$ and $\{s_l, \dots, s_j\}$. This partition also means that the polygonal segment L_{ij}^k has been replaced by two new segments. The process is repeated until all the sides satisfy the maximum-distance criterion.

The operation of this algorithm can be described in terms of a binary tree. The root represents the whole contour, each node is a partition (subset) and the

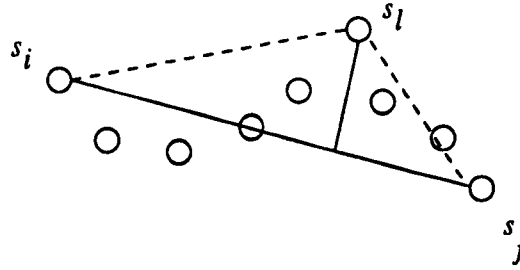


Figure 2.1: Maximum-distance criterion to create new segments. Segment $\overline{s_i s_j}$ is split into $\overline{s_i s_l}$ and $\overline{s_l s_j}$ when the orthogonal distance from point s_l to it is greater than a constant value.

leaves are the final partitions sought (segments). For example in Figure 2.2 (a) we show an open contour and in (b) its corresponding final tree. The initial partition is represented by the root AF , that is segment $\overline{AF}$. In a sequential scan from A to F , point C is found to be the farthest point to segment $\overline{AF}$ and that its distance to $\overline{AF}$ is greater than a specified value. The two new segments $\overline{AC}$ and $\overline{CF}$ are therefore created. In the tree this action is represented by the creation of the children AC and CF .

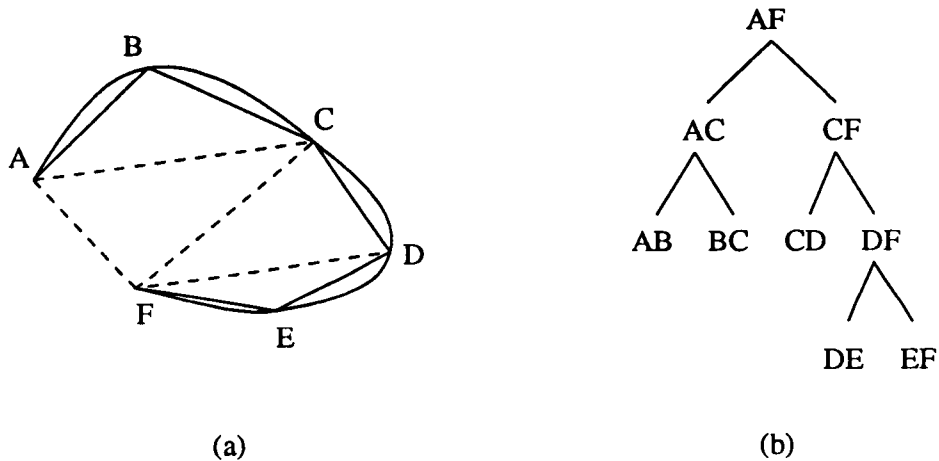


Figure 2.2: Tree representation of the splitting process. An initial segment AF in (a) is successively split until a distance criterion is satisfied. (b) shows the order with which the approximation is carried out.

The following are a few observations about Ramer's algorithm:

- The set of the polygonal vertices are a subset of the original contour points.
- The two initial points are chosen arbitrarily and the algorithm does not guarantee to produce the minimum number of line segments.
- The computational complexity is $O(m \log n)$, where m is the number of points in the initial contour, and n is the number of polygon line segments.
- The iterative approach can be implemented in parallel.

Figure 2.3 presents a stack implementation of the above tree description.

2.3 Polygon approximation by linear scan

Sklansky and Gonzalez [14] propose a “scan-along” algorithm to find a polygonal approximation A of a contour C such that the Hausdorff-Euclidean distance $H(A, C)$ ¹ between A and C is no bigger than a given error value ϵ . The algorithm is sometimes referred as the *cone intersection* algorithm because it uses intersecting cones to determine whether a contour point satisfies the error distance criterion.

The principal component of this algorithm is the Optimal Segment Finder (OSF) which will be explained with reference to Figure 2.4. Let s_{i-1} be the last vertex found by OSF. Starting from s_{i-1} and for each new point the algorithm constructs a cone with origin in s_{i-1} , one ray is the top tangent to the circle with radius ϵ and centered in s_j , and the other ray is the bottom tangent to the same circle. TOP_j and $BOTTOM_j$ are defined as the angles between the top ray T_j and

¹ $H(A, B) = \max(\max_{x_1 \in B} \min_{x_2 \in A} \|x_1 - x_2\|, \max_{x_1 \in A} \min_{x_2 \in B} \|x_1 - x_2\|)$

Algorithm: *PA by splitting*

Input: Set $S = \{s_0, s_1, \dots, s_{m-1}\}$ representing the curve. Threshold α for the maximum-distance criterion.

Output: Set $S' = \{s'_0, s'_1, \dots, s'_{n-1}\}$ representing the vertices of the polygon.

```

{
  if (chain is open) {
     $s'_x = s_0$ ;
     $s'_y = s_{n-1}$ ;
    push(CLOSED,  $s'_x$ );
    push(OPEN,  $s'_y$ );
  } else {
     $s'_x$  = top left-most point in  $S$ ;
     $s'_y$  = bottom right-most point in  $S$ ;
    push(OPEN,  $s'_y$ );
    push(CLOSED,  $s'_x$ );
    push(OPEN,  $s'_x$ );
  }
  while ( !empty(OPEN) ) {
     $s'_i = \text{top}(\text{OPEN})$ ;
     $s'_f = \text{top}(\text{CLOSED})$ ;
     $d_m = \max\{d_j \mid d_j = d(s_j, \overline{s'_i s'_f})\}$ ;
     $s'_m$  = point with maximum distance;
    if ( $d_m > \alpha$ )
      push(OPEN,  $s'_m$ );
    else
      push(CLOSED, pop(OPEN));
  }
  while (! empty(CLOSED)) {
     $s'_i = \text{pop}(\text{CLOSED})$ ;
     $i++$ ;
  }
}

```

Figure 2.3: Algorithm for PA by splitting

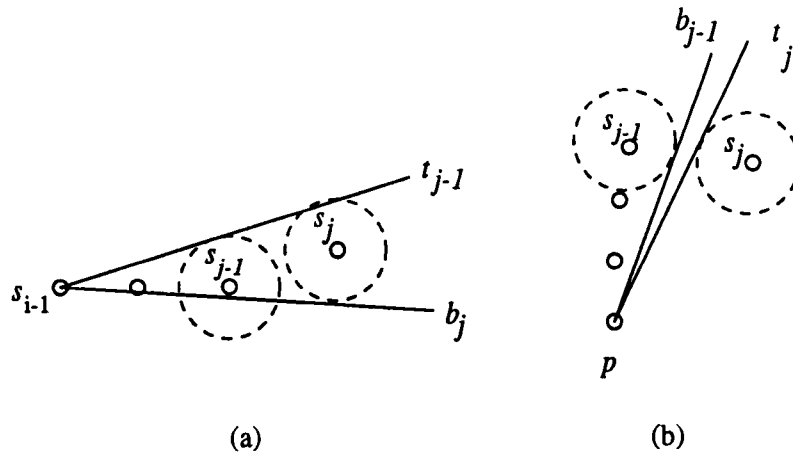


Figure 2.4: Cone intersection method. (a) Data point s_j satisfies error criterion; (b) TOP_{min} and $BOTTOM_{max}$ cross each other and a new segment is created

the bottom ray B_j and the positive direction of the X axis respectively. OSF will keep in TOP_{min} the smallest TOP_j and in $BOTTOM_{max}$ the largest $BOTTOM_j$ for the current segment. If TOP_{min} is smaller than $BOTTOM_{max}$ (crisscross), contour point s_{j-1} becomes the next polygon vertex s_i and the process is repeated with new source point s_j . Otherwise, OSF will continue walking along the contour looking for the next point that violates this criterion. In Figure 2.4 we show only the minimum top tangent and the maximum bottom tangent encountered so far. Thus, s_j satisfies the error criterion in (a) but not in (b).

Among the important features of Sklansky's algorithm are:

- The resulting polygonal vertices are a subset of the contour.
- The complexity of the algorithm is $O(m)$, where m is the number of contour points.
- Since the algorithm is a scan-along process, it cannot be parallelized.

Although the authors introduced two other rules to deal with special cases, we present here only a simplified version of their algorithm in Figure 2.5.

Algorithm: *PA by linear scan*

Input: Set $S = \{s_0, s_1, \dots, s_{m-1}\}$ representing the curve. Radius ϵ for circles.

Output: Set $S' = \{s'_0, s'_1, \dots, s'_{n-1}\}$ representing the vertices of the polygon.

```

{
  p = s0; /* source point */
  j = 1; /* index of current point */
  k = 0;
  s'k = s0;
  while (sj ≠ s0) {
    create circle with radius  $\epsilon$  centered in sj;
    Tj = top tangent from p to circle;
    Bj = bottom tangent from p to circle;
    TOPj =  $\angle(T_j, X)$ ;
    BOTTOMj =  $\angle(B_j, X)$ ;
    if (TOPj < TOPp) TOPmin = TOPj;
    if (BOTTOMj > BOTTOMp) BOTTOMmax = BOTTOMj;
    if (TOPmin < BOTTOMmax) {
      k = k + 1;
      s'k = sj-1; /* new polygon vertex */
      p = sj-1;
    } else
      j = j + 1;
  }
}

```

Figure 2.5: Algorithm for PA by linear scan

Chapter 3

Equilateral Polygonal Approximation

The polygonal approximation problem has been defined before as the process of approximating a given contour by a sequence of straight lines. Different algorithms have been proposed for this problem, but no algorithm exists to approximate a contour by a polygon for which all its segments have the same length.

To understand the very nature of the problem at hand we need to look at the role that the segment length has played in other approaches, some of which are “scan-along”, others are “split” techniques, and many are hybrids. Different error functions have been defined to drive the creation (and destruction) of segments. There are global as well as local criterion functions. But in none of them have the sides been constrained to be of the same length. We notice that although these algorithms might be modified such that the segment length is restricted to be in a certain interval, they were not designed with this in mind.

Let $C = \{c_0, \dots, c_{m-1}\}$ be a discrete contour of m points $c_i = (x_{ic}, y_{ic})$ to be approximated by a polygon. Let also l be the length of a segment. Then the PA

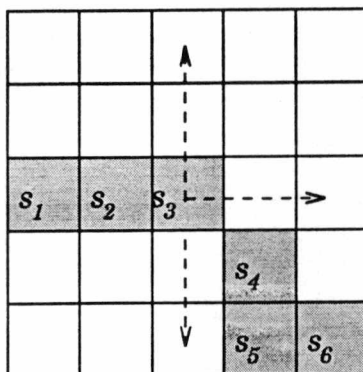


Figure 3.1: A discrete grid not always work to assign polygon points. Neither s_4 , s_5 , and s_6 are at a distance 2 from point s_3 .

is to produce a sequence of polygonal vertices $s_i = (x_i, y_i)$, $i = 0, \dots, n - 1$, such that for all i

$$(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 = l^2 \quad (3.1)$$

Since we are using Euclidean distance, we will allow the polygon coordinates to take on any value in the real plane. This means they are not necessarily restricted to be a subset of the original data set. The reason for this decision is illustrated with an example in Figure 3.1.

Suppose that $l = 2$ and that the last segment found was $\overline{s_1 s_3}$. From point s_3 there are only 3 positions at a distance 2 from it and none of them is a contour point. Candidate points for the next polygon vertex are s_4 , s_5 , and s_6 , but their distances to s_3 are $\sqrt{2}$, $\sqrt{5}$, and $\sqrt{8}$ respectively. Therefore in order to create the next segment we must either relax constraint 3.1 or extend the discrete grid to the real plane.

It has been found during the development of this work that constraint 3.1 introduces a high level of complexity in the problem, hard to handle with many of the classical and successful techniques for PA. Perhaps one of the most difficult

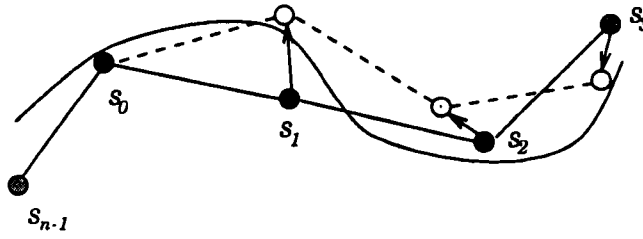


Figure 3.2: Example of how polygon vertices are interconnected because of the length constraint.

problems is that the polygon vertices seem to be highly connected. This connection produces a “ripple” effect when a point is moved. For instance, in Figure 3.2 we show one possible scenario when point s_1 is moved in order to achieve a better approximation. Since no segment can stretch, points s_2 and s_3 must also be moved. It is also possible that this ripple effect reaches point s_1 again.

For some cases, it is likely that the repositioning of a point will produce a worse approximation for other sections of the contour. Since this example shows only one of many possible configurations, a natural question arises regarding the selection of new point locations: Where should the points be relocated to in order to improve the polygonal approximation?

One way of facing this question is by using nonlinear programming (NLP). In the general case one wants to optimize (minimize or maximize) a nonlinear objective function $F(X)$ such that the optimal vector X satisfies a set of nonlinear constraints. In mathematical terms, the formulation becomes

$$\begin{aligned}
 &\text{Minimize} && F(X) \\
 &\text{subject to} && \\
 &&& X_l \leq X \leq X_u \\
 &&& c_i(X) = 0 \quad i = 1, \dots, m_e \\
 &&& c_i(X) \geq 0 \quad i = m_e + 1, \dots, m
 \end{aligned} \tag{3.2}$$

where X_l and X_u are lower and upper vector bounds for X , $c_i(X)$ is a set of nonlinear constraint functions, and m_e is the number of equality constraints.

In our case the task reduces to find an appropriate objective function, use the constraints 3.1 and find the point where it is optimized. It is clear that the function to be defined has to be such that the final polygonal approximation resembles as closely as possible the original contour. Although there are many ways to develop dissimilarity (or similarity) measures for PA, there is no single function that computes a unique "shape measure" that guides us in developing a perfect objective function.

For a valid NLP formulation, the set of vertices $s_i = (x_i, y_i)$ must be mapped into the column vector X in the following way:

$$X = \begin{bmatrix} x_0 \\ y_0 \\ x_1 \\ y_1 \\ \vdots \\ x_{n-1} \\ y_{n-1} \end{bmatrix}$$

For convenience and clarity, we express the objective functions in terms of contour points $c_j = (x_{jc}, y_{jc})$ and polygon points $s_i = (x_i, y_i)$, instead of in terms of elements from vector X . We will assume that an appropriate procedure exists to evaluate F according this mapping.

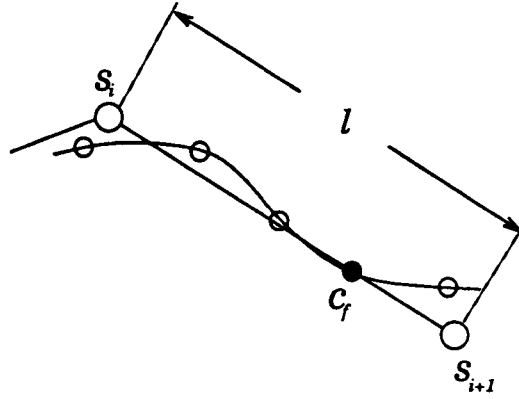


Figure 3.3: Finding the next polygon point by a simple walk along the contour.

3.1 Computing an initial guess

The NLP approach requires an initial estimate of the solution in order to start the computation. Depending of the NLP algorithm this estimate can be outside or inside the feasible region. In our case we need to provide a set $S = (s_0, \dots, s_{n-1})$ for which all the constraints are satisfied. Starting at point s_0 , the algorithm walks along the contour finding polygon vertices s_i separated at a distance l by a simple extrapolation procedure. With reference to Figure 3.3, let s_i be the last vertex found and let c_f be the farthest contour point from s_i such that $d(s_i, c_f) \leq l$. The coordinates of the new vertex s_{i+1} are found by projecting a line segment with slope $(y_{fc} - y_i)/(x_{fc} - x_i)$ l units from s_i and passing through c_f . After simple algebraic manipulation we get

$$x_{i+1} = x_i \pm \frac{l}{\sqrt{m^2 + 1}} \quad (3.3)$$

$$y_{i+1} = y_i \pm \frac{ml}{\sqrt{m^2 + 1}} \quad (3.4)$$

These equations give two points because the equations do not consider the direction of the extrapolation. The correct one is taken by testing the relative position of c_f with respect to s_i . This process is repeated until we reach s_0 where

a special computation must be done in order to properly close the polygon. It is very unlikely that s_0 coincides with the last s_{i+1} as above obtained.

Before describing how the last polygon vertex is computed, we describe a general procedure to calculate the coordinates of a point which is at a distance l of two other points which in turn are at distance less than l of each other (see Figure 3.4). Let $s_p = (x_p, y_p)$ and $s_q = (x_q, y_q)$ be two points such that the distance between them is $e < l$. We want to find the position of the "intersection" point $s_r = (x_r, y_r)$ such that $d(s_p, s_r) = d(s_q, s_r) = l$. The algorithm uses equations 3.3 and 3.4 to find a point s_s which is beyond s_q and at a distance l from s_p . Then s_r is found by rotating s_s an angle α by computing

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} x_p \\ y_p \end{pmatrix} + \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} x_s - x_p \\ y_s - y_p \end{pmatrix} \quad (3.5)$$

where

$$\alpha = \cos^{-1}\left(\frac{e}{2l}\right)$$

and e is the distance from s_p to s_q .

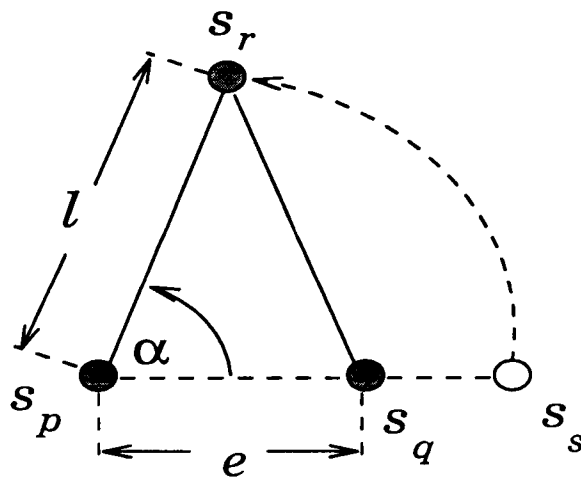


Figure 3.4: Computation of the last polygon point for the initial estimate.

Back to the computation of the last point s_{n-1} , let s_k be the last vertex found before reaching the initial contour point s_0 . The algorithm computes four intersection points using the procedure just described between s_{k-1} and s_0 , s_{k-2} and s_0 , s_{k-1} and s_1 , and s_{k-2} and s_1 . Then, s_{n-1} is the one for which the maximum distance from any point on the respective section of the contour to it is minimum.

In the next sections, we will develop a set of possible objective functions and we will discuss their merits and limitations.

3.2 A least-squares function

A common technique to approximate a continuous function of one variable by a piecewise linear function is by the method of least-squares. It consists in minimizing the sum of the squares of the difference between the real function value and the approximating one.

We apply the same principle here by taking the orthogonal distance from a contour point to its closest segment as shown in Figure 3.5. Notice that for each segment only a restricted number of points are considered to compute the error. These points are those for which their orthogonal projections are between the two end points of the segment. For example, contour point c_j is projected onto segment $\overline{s_i s_{i+1}}$ instead of onto $\overline{s_{i-1} s_i}$. Clearly this is a local property which is being minimized.

The equation of the line formed by two points $s_i = (x_i, y_i)$ and $s_{i+1} = (x_{i+1}, y_{i+1})$ is

$$L_{i,i+1} : by = ax + x_{i+1}y_i - x_i y_{i+1} \quad (3.6)$$

where $a = y_{i+1} - y_i$ and $b = x_i - x_{i+1}$. Then the distance from a point $c_j = (x_{jc}, y_{jc})$

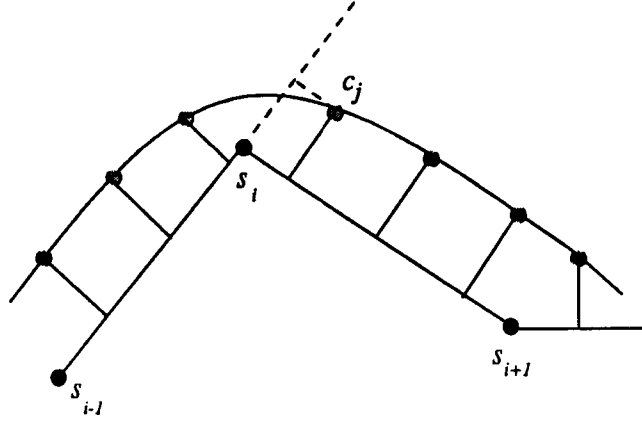


Figure 3.5: Curve points are orthogonally projected on their respective segments.

to $L_{i,i+1}$ is given by

$$d(c_j, L_{i,i+1}) = \frac{|by_{jc} + ax_{jc} + y_i x_{i+1} - x_i y_{i+1}|}{\sqrt{a^2 + b^2}} \quad (3.7)$$

Equation 3.7 is specially useful because it is valid even when the slope of the line is infinite. The formulation of the NLP becomes

$$\min F(X) = \sum_{i=0}^{n-1} \sum_{j \in R_i} d(c_j, L_{i,i+1})^2 \quad (3.8)$$

$$\text{s.t.} \quad \|s_i - s_{i+1}\| = l \quad i = 0, \dots, n-1 \quad (3.9)$$

where $L_{i,i+1}$ is segment $\overline{s_i s_{i+1}}$, c_j is contour point j , $d(c_j, L_{i,i+1})$ is the distance from point c_j to $L_{i,i+1}$, and R_i is the set of valid contour points for segment i . Because of the cyclic nature of the problem all summation indices are modulo n . Now we explain how the region R_i is computed.

Before computing the distance from a contour point to a given segment from 3.7, we calculate the projection of the point onto the segment. If the Euclidean distance from the projected point to the point which lies in the middle of the segment is greater than half of the segment length, the point is not included in R_i . In Figure 3.6 we show a point $c_j = (x_{jc}, y_{jc})$ which is to be projected onto

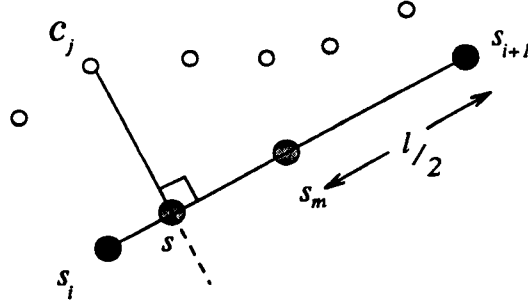


Figure 3.6: Computing if c_j is in R_i . In this case $c_j \in R_i$ because its projection s lies between the two end points of the segment.

the line $L_{i,i+1} : \overline{s_i s_{i+1}}$, with $s_i = (x_i, y_i)$ and $s_{i+1} = (x_{i+1}, y_{i+1})$. Let m_i be the slope of $L_{i,i+1}$, and assume that is not equal to infinite. The equation for $L_{i,i+1}$ is then

$$y = m_i x + b_i$$

where

$$\begin{aligned} m_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \\ b_i &= \frac{y_i x_{i+1} - x_i y_{i+1}}{x_{i+1} - x_i} \end{aligned}$$

The projection of c_j creates an orthogonal line L_j to $L_{i,i+1}$ with slope $m_j = \frac{1}{-m_i}$. Therefore the equation for L_j is

$$y = m_j x + b_j$$

where $b_j = y_{jc} - m_j x_{jc}$. Intersecting $L_{i,i+1}$ and L_j we get

$$m_i x + b_i = m_j x + b_j$$

And finally, the coordinates of the projected points are

$$\begin{aligned} x &= \frac{b_j - b_i}{m_i - m_j} \\ y &= \frac{m_i b_j - m_j b_i}{m_i - m_j} \end{aligned}$$

Let $s_m = (x_m, y_m)$ be the point lying in the middle of segment $Li, i + 1$, that is

$$\begin{aligned} x_m &= \frac{x_{i+1} + x_i}{2} \\ y_m &= \frac{y_{i+1} + y_i}{2} \end{aligned}$$

Then, point c_j is in R_i if and only if

$$(x_m - x)^2 + (y_m - y)^2 \leq (l/2)^2 \quad (3.10)$$

The formulation presented in this section, has a problem which is related to the mathematical definition of the objective function. If we factored out the distance function d in equation 3.8 for a given segment i , we would get a quadratic function on x_i and y_i , whose coefficients will be given by the contour points in R_i . When searching for a minimum of a function, we have to take derivatives with respect to the variables, and the improved point is a function of the previous point and the gradient at that point. But R_i is a changing set. Therefore, the quadratic functions change from iteration to iteration. This produces, for some cases, problems with respect to convergence to a global or even a local minimum.

3.3 The area as main feature

In this section we consider using an area difference to derive polygon approximations. With this approach we will be able to compute more appropriate error measures between the contour and the PA. We have found two different ways to compute this error: locally, by computing the area enclosed between the contour and polygon, and globally by computing a difference between total areas.

3.3.1 Global minimization

A very simple objective function is the area deviation between the original contour and the polygon. The main advantage of this function is that is easy and fast to evaluate. The area A of a polygon is defined by

$$A = \frac{1}{2} \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \quad (3.11)$$

This equation gives the signed area of a polygon. When the summation is taken in a clockwise fashion the area is negative, and positive when is counterclockwise. Assuming a positive calculation of areas, the objective function is defined as

$$F(X) = \left(AC - \frac{1}{2} \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \right)^2 \quad (3.12)$$

where AC is the constant area enclosed by the original contour which is computed in an initial phase.

This objective function will not necessarily produce the desired results because we are not taking into consideration the relative spatial distributions of the contour points. Area AC could well be the area of a rectangle, triangle, circle or any other shape, and we have no way to inform this to the NLP program. Therefore we are not assured that the polygon will converge to the desired shape. To solve this problem, we follow the approach that Montanari [9] used to find a polygon with minimum perimeter. The idea is to restrict the search space for the polygon vertices in such a way they are confined to be always very close to the contour. This can be done by defining circular regions as shown in Figure 3.7 whose centers are found and fixed in an initial phase using the same procedure to find the initial PA estimate described in Section 3.1. Let (x_{is}, y_{is}) be the center of search space region for $s_i = (x_i, y_i)$. The NLP formulation is then defined by objective function 3.12, constraints 3.9, and the new set of region constraints

$$(x_i - x_{is})^2 + (y_i - y_{is})^2 \leq r^2 \quad i = 0, \dots, n-1 \quad (3.13)$$

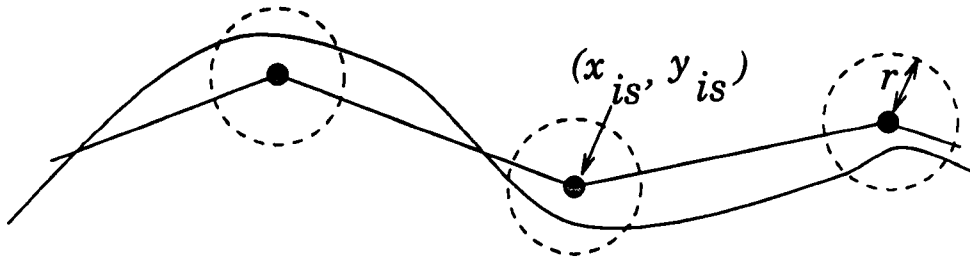


Figure 3.7: Defining circular regions to constrain area minimization. A polygon vertex can only be located inside its circular region.

where r is a predefined circle radius.

As we said before, one of main advantage of this function is that is easy and fast to evaluate: the complexity for evaluating $F(X)$ is $O(n)$, where n is the number of polygon vertices. Since the function is quadratic, we know that every local minimum is also a global minimum (there could be many optimum configurations). The function is also well defined. The main disadvantage lies in the fact that the approximation is driven by a global feature i.e area, which is not always good and complete enough to describe the shape to which the polygon should converge.

3.3.2 Minimizing local areas

The idea of this approach is to assign to each segment a section of the area enclosed by it and the contour. By drawing lines bisecting the interior angles for each polygon vertex, we can partition the total area deviation in a set of regions that can be seen as polygons and use equation 3.11 to compute the area. Each of these polygons is formed by: the two segment points, the intersection points between the bisecting lines and the contour, and the contour points.

For example, Figure 3.8 (a) shows segment $\overline{s_i s_{i+1}}$ and contour points $a =$

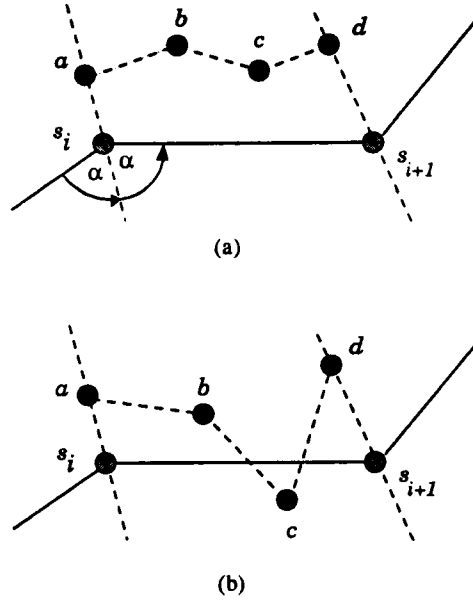


Figure 3.8: Curve points are considered as being polygon vertices. In (a) no intersections occur among segments. In (b) segments $\overline{bc}$ and $\overline{cd}$ intersect segment $\overline{s_i s_{i+1}}$.

(a_x, b_y) , $b = (b_x, b_y)$, $c = (c_x, c_y)$, and $d = (d_x, d_y)$. The area for this case is:

$$\frac{1}{2}(x_i a_x + y_i a_y + a_x b_y + \cdots + d_x y_{i+1} + d_y x_i + x_{i+1} y_i + y_{i+1} x_i) \quad (3.14)$$

This method works when all contour points are located on the same side of the segment as in Figure 3.8 (a). However, this procedure will not work for cases like Fig 3.8 (b), because the sequence of points $s_i, a, b, c, d, s_{i+1}, s_i$ is clearly not a polygon due to the intersection of the two segments $\overline{c_2 c_3}$ and $\overline{c_3 c_4}$ with segment $\overline{s_{i+1} s_i}$, and therefore equation 3.14 does not apply.

The solution adopted for this problem is to rotate all those points lying inside the polygon with respect to one of the segment ends, such that their new positions are outside the polygon and their respective contributions in area deviation are the same as before rotation.

We say that a point is inside the polygon when it is to the left of the segment or similarly when its signed distance to the segment is negative. Let $p = (x_p, y_p)$

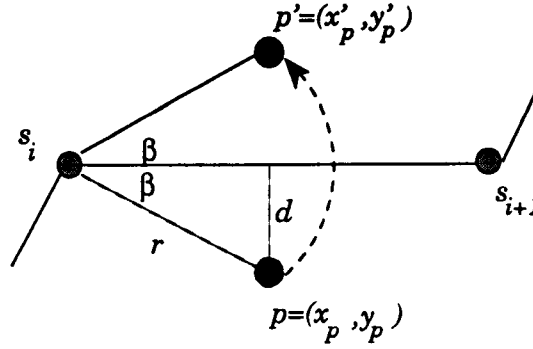


Figure 3.9: All contour points p inside the polygon are rotated out.

be a point to be rotate over $\overline{s_i s_{i+1}}$ in Figure 3.9. Let β be the angle between a line from s_i to it and segment $\overline{s_i s_{i+1}}$. The new position $p' = (x'_p, y'_p)$ of p after being rotated an angle 2β with respect to s_i is:

$$\begin{pmatrix} x'_p \\ y'_p \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} \cos 2\beta & -\sin 2\beta \\ \sin 2\beta & \cos 2\beta \end{pmatrix} \begin{pmatrix} x_p - x_i \\ y_p - y_i \end{pmatrix} \quad (3.15)$$

where $\beta = \sin^{-1}(\frac{d}{r})$, d is the distance from p to segment i , and r is the length of the ray $\overline{s_i p}$.

As we said before, the other points that must be included as polygon vertices are the intersection points between the polygon segment and the contour. This can be done during the rotation procedure. By keeping track of where (left or right) the last contour point was before rotation (if necessary), we know the segment intersects the contour when the positions of the current point and last point are different. Notice that this rotation procedure is only virtual in the sense that it does not modify the original contour.

Let us summarize the procedure to compute the “local” area A_i associated with segment i (see Figure 3.10):

- Draw a line from s_i to the contour such that it bisects the interior angle formed by segments $\overline{s_{i-1} s_i}$ and $\overline{s_i s_{i+1}}$.

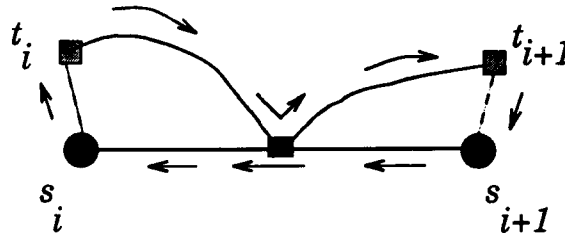


Figure 3.10: When computing the area enclosed between the contour and the segment, the points must be taken in the direction shown in the figure.

- Do the same for point s_{i+1} .
- Find intersection points t_i and t_{i+1} of these two lines with the contour.
- Find the intersection points between the contour and segment.
- Rotate all points to the right of the segment around $\overline{s_i s_{i+1}}$.
- Compute A_i using equation 3.11 with the ordered sequence of polygonal points

$$(s_i, t_i, \underbrace{c_{i1}, \dots, c_{im}}_{\text{contour and intersection points}}, t_{i+1}, s_{i+1})$$

starting from s_i .

Figure 3.11 shows the final sequence of points for example in Figure 3.8 (b). Two points p and q are generated when rotating contour point c . Note that these points are considered in the area equation when going from s_i to s_{i+1} , but not from s_{i+1} to s_i .

Finally, the objective function to be minimized is

$$F(X) = \sum_{i=0}^{n-1} A_i^2 \quad (3.16)$$

This function has much the same problems as the least-squares function: its derivatives are functions of varying coefficients from iteration to iteration.

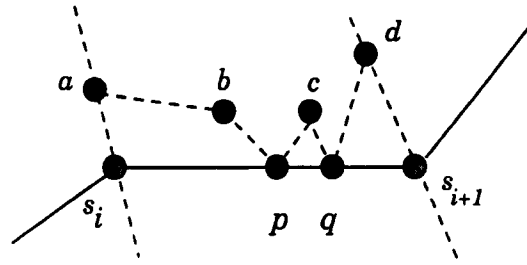


Figure 3.11: Resulting sequence of contour, intersection, and polygon points for previous example.

3.4 An energy function and an optimal solution

3.4.1 Motivation

In this section we present a solution for the Equilateral Polygonal Approximation Problem which is a natural consequence of a careful analysis of the results from previous solutions.

One way of describing the solution is in terms of a mechanical system of springs. We can think of each contour point as a fixed particle to which a spring is attached. Polygon segments are assumed to be massless rigid rods linking adjacent vertices, which in turn are considered as free massless particles. The vertices move under the action of the interacting forces in the system.

In the most general case, every polygon vertex s_i is attached to every contour point c_j through a nonlinear spring possibly having different natural lengths. This configuration can be thought as a fully connected system. For a system with m contour points and n polygon vertices there is a total of nm springs; each polygon vertex is “pulled” by m different forces and each contour point exerts a force over n polygon vertices. Nonlinear springs are those for which the force necessary to stretch them is a nonlinear function of the distance to be stretched. For instance, we can construct a function such that the longer a spring has been stretched, the

smaller is the force to stretch it further, as occurs with gravitational force.

The equilibrium state of the system corresponds to that which minimizes the overall potential energy stored in all the springs. For our purpose, this equilibrium state would give us the polygon approximation we are after.

3.4.2 Formulation

Instead of having a fully connected system, we will simplify the problem by introducing the following three restrictions:

- To each contour point, we will anchor only one spring, which is to be attached to a vertex lying close to it.
- We will assume that the springs are linear, that is the amount of force needed to stretch it one unit is always the same.
- All the springs will have the same length.

This decision is based on the fact that many of the forces (for the general case) will cancel out or will be so small that they will not have a considerable effect on the final vertex position. The idea is that a vertex is only affected (“pulled”) by those contour points close to it (see Figure 3.12). Notice that the total number of springs is now only n , thereby significantly reducing the complexity of the problem.

The potential energy V stored by an ideal linear spring when stretched a distance r beyond its natural length is:

$$V = \frac{1}{2}kr^2 \quad (3.17)$$

where k is the “spring stiffness” constant which is defined as the force required to stretch the spring one unit.

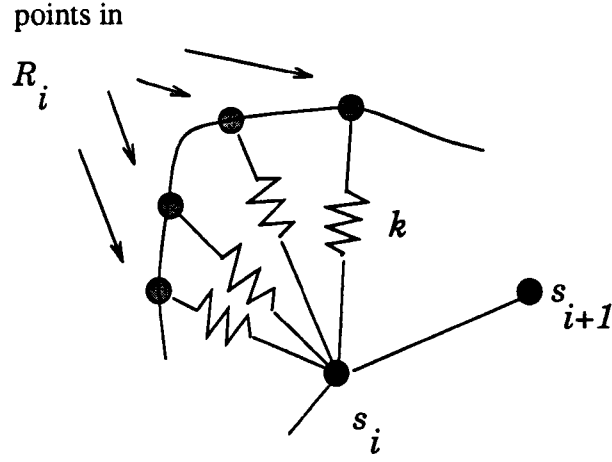


Figure 3.12: A vertex point is attached to only a subset of contour points

In order to formulate the objective function, assume we have an ideal linear spring with constant k and natural length L which is attached to contour point $c_j = (x_{jc}, y_{jc})$ and polygon vertex $s_i = (x_i, y_i)$ (see Figure 3.13). The potential energy is

$$V_{i,j} = \frac{1}{2}k_j \left(\left((x_i - x_{jc})^2 + (y_i - y_{jc})^2 \right)^{1/2} - L \right)^2 \quad (3.18)$$

and the objective function to be minimized then becomes

$$F(X) = \sum_{i=0}^{n-1} \sum_{j \in R_i} V_{i,j} \quad (3.19)$$

where R_i is the set of contour points to which a spring is attached from s_i . Since in this model there is only one spring for contour point c_j , we will refer to it as spring j .

Equation 3.19 is a function of two parameters: the spring natural length L (potential energy zero) and the number and positions of the springs to be attached to vertex point s_i , that is region R_i .

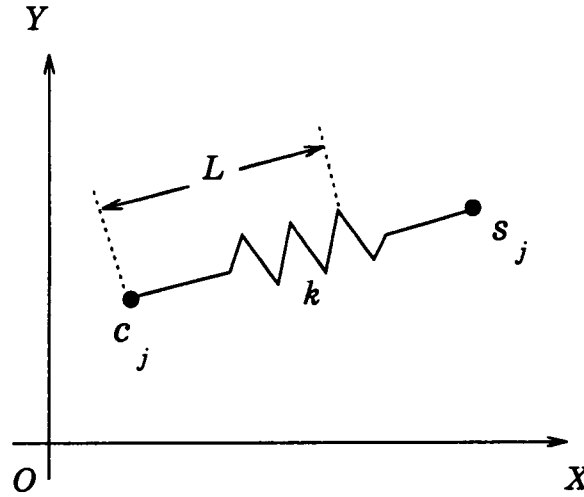


Figure 3.13: General case for computing potential energy of a spring

The spring length controls the distance from the polygon vertices to the contour points. If we want the polygon vertices to be very close to the contour, L should be set as small as possible. But this is not a problem, because we are dealing with ideal springs. Moreover, we can abandon the mechanical analogy and make all spring lengths equal to zero. The objective function is therefore just a sum of the distances from polygon vertices to contour points. Since the distance function is a convex function and the sum of convex functions is also a convex function, we get an objective function with a global minimum. Therefore, the solution obtained is optimal with respect to the setting of the problem.

In the following discussions we assume the energy function is defined with $L = 0$.

The set R_i controls the directions and number of forces pulling a polygon vertex. If we assign a large number of springs to a given vertex we will need a large opposite force to move it to another position. If we assign a very small number of springs to it, its final position will be more the result of forces acting over adjacent vertices than acting over it. We can see that an appropriate determination of R_i

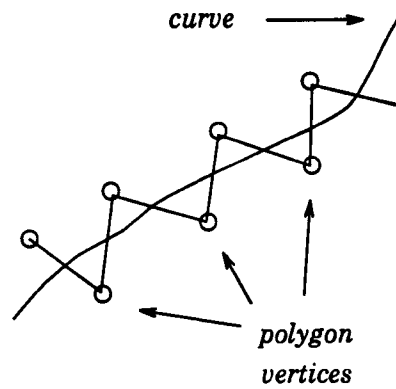


Figure 3.14: For some cases, a cluster effect is obtained in the final PA when the initial polygon is computed using previous algorithm.

is crucial to get a good polygon approximation.

3.4.3 Computation of the number of segments

Recall that for previous objective functions, the computation of the number of segments is a direct consequence of the computation of the initial polygon (see Section 3.1). This is not the case for the energy method. The initial polygon guess procedure tends to produce a number of segments always larger than the necessary when using Equation 3.19. The result would be that the polygon has a much larger perimeter than the contour such that some segments get clustered like Figure 3.14 shows.

The clustering effect is produced because a small number of contour points, relative to the segment length, is assigned to a polygon vertex. The problem is aggravated when this situation happens for a sequence of adjacent vertices. Thus, the perimeter covered by the segments is much larger than the one covered by the contour points and there is not enough room to accommodate the segments.

For the energy method, we will compute first the number of segments and then the initial polygon. The computation of the number of segments is carried out

by determining the number of sets R_i (and their corresponding members). The idea of the algorithm is to assign to adjacent regions R_i and R_{i+1} a number of contour points such that a segment with length l can move to the left or right without producing clustering. If a small number of contour points are assigned to R_i , it is desirable that a large number of contour points are assigned to R_{i+1} and vice versa. It is also possible that for certain cases depending on the contour characteristics many consecutive regions are assigned a small number of contour points producing potentially "dangerous" situations of clustering.

An accumulated error measure will keep track of this information. This error is increased or decreased depending on the error introduced by the new region. Let l be the segment length and suppose we have determined the first point in R_i , say c_i . The objective is now to find c_{i+1} , the first point in R_{i+1} , such that the accumulated error is minimized. Notice that the elements of R_i are known only after c_{i+1} has been found. That is, all points between c_i and c_{i+1} , excluding the latter, conform R_i .

The algorithm walks along the contour from c_i looking for the first point such that the distance from c_i to it is not greater than l . Let c_f be this point and let c_{f+1} be the next contour point. The algorithm will choose c_f or c_{f+1} as c_{i+1} depending on which of the two reduces the accumulated error most. The accumulated error is the sum of individual errors for each region R_i . The error for R_i is defined as the difference between l and the distance from c_i to c_{i+1} . The algorithm stops when reaching c_0 , the first contour point. If the last region, say R_{k-1} , is too small with respect to l , it is evenly split between R_0 and R_{k-2} .

The following are the main steps of the algorithm for computing the sets R_i :

1. Set $R_0 = \{c_0\}$. The first contour point becomes the first point assigned to s_0 . Set E_0 , the accumulated error up to segment 0, equal zero.

2. For the last vertex s_i , walk along the contour through points c_j looking for the farthest contour point c_f located not beyond a distance l from c_i . Update R_i by $R_i = R_i \cup c_j$, for all $i < j < f$.
3. Let d_f and d_{f+1} be the distances from c_i to c_f and c_{f+1} respectively. Compute $e_f = |l - d_f|$, the error introduced by c_f . Similarly, $e_{f+1} = |l - d_{f+1}|$ for point c_{f+1} .
4. Choose c_f or c_{f+1} as the initial point for R_{i+1} depending on the following criterion:
 - If $|E_i + e_f| < |E_i - e_{f+1}|$ then choose c_f . The new accumulated error is $E_{i+1} = E_i + e_f$
 - Otherwise, choose c_{f+1} and $E_{i+1} = E_i - e_{f+1}$.
5. Repeat from step 2 until all contour points are exhausted.
6. If last region found R_{k-1} is too small, split it between R_0 and R_{k-2} and the final number of segments (vertices) n is $k - 1$; otherwise $n = k$.

Some of the regions found by the above algorithm may have to be merged. The reason being that for certain cases the two end points of a segment are pulled in opposite directions such that the resulting configuration for a minimum potential energy causes two segments cross each other. Figure 3.15 (a) shows an example of this situation and (b) shows the undesirable final configuration.

The solution adopted for this problem is the following: Let c_{i-1} , c_i , and c_{i+1} be the first contour points in R_{i-1} , R_i , and R_{i+1} respectively. Region R_{i-1} will be merged with region R_i if the angle formed by the two rays $\overrightarrow{c_{i-1}c_i}$ and $\overrightarrow{c_i c_{i+1}}$ is less than a predefined threshold α . After merging, the number of segments (and

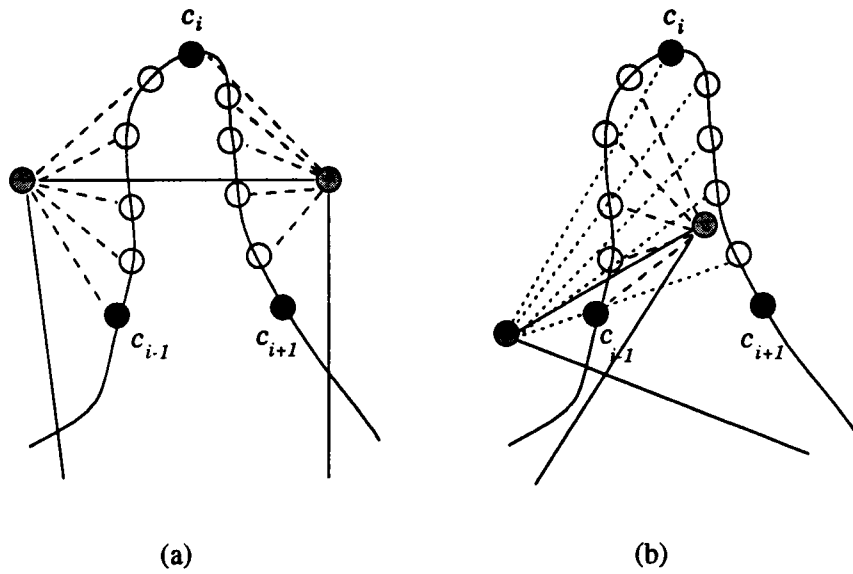


Figure 3.15: Example of how a segment cannot be appropriately fit to the contour.

vertices) is reduced by one (see Figure 3.16). For the experiments α has been set to 30° .

After computing the number of segments, we need to determine the initial polygon. Since the objective function is convex for spring length zero, any starting configuration satisfying the set of constraints will converge to the optimal solution. The initial guess is therefore simply computed as a regular polygon with n sides of length l centered at the contour's centroid.

3.4.4 Variations to the original model

Making use of the mechanical analogy again, we can think of two extensions to our original model.

1. Allow the springs to have different constants.
2. Use springs to link adjacent polygon vertices.

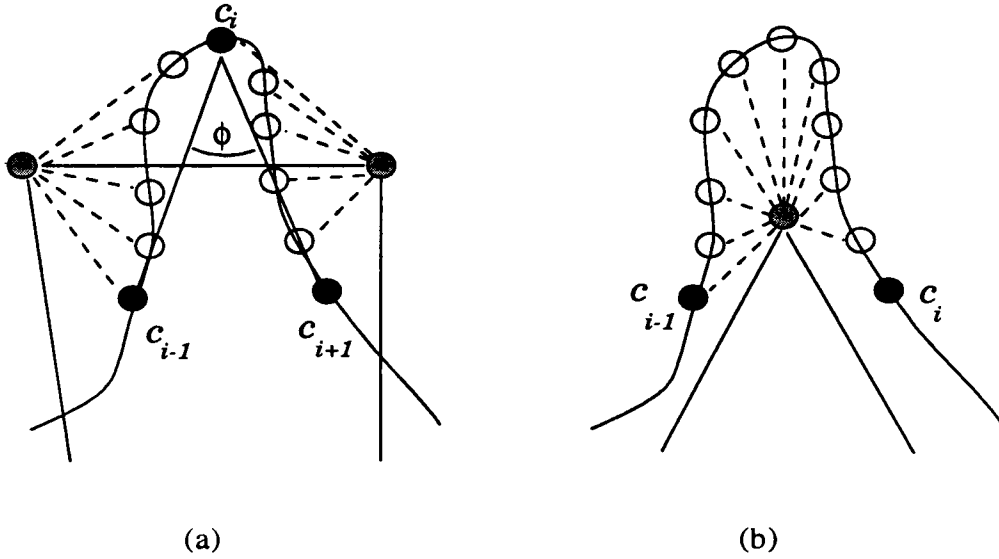


Figure 3.16: Two adjacent regions R_{i-1} and R_i are merged if ϕ is less than a given fixed value.

By having different constants for different springs we can control the force exerted by them. Intuitively, we want those points located in high curvature sections of the contour to have larger constants than those in small curvature sections. Therefore, a vertex polygon which is attached to one of these high curvature contour points will remain close to it unless we exert a very high force with other contour points in opposite directions. Since the curvature measure is good only for “rounded” smooth objects, we have decided to use a variation of it: a function of the average angle formed by the segment lines $\overline{c_{j-p}c_j}$ and $\overline{c_jc_{j+p}}$, where p ranges from 1 to g , and g is a predefined value. The potential energy $V_{i,j}$ in 3.19 for variation 1 is then

$$V_{i,j} = \frac{1}{2}k_j(x_i - x_{jc})^2 + (y_i - y_{jc})^2 \quad (3.20)$$

where

$$k_j = \left(\frac{1}{g} \sum_{r=1}^g \frac{\theta_r}{(g-r+1)^2} \right)^{-1} \quad (3.21)$$

and $\theta_r = \angle(\overline{c_{j-r}c_j}, \overline{c_jc_{j+r}})$.

Variation 2 allows the polygon vertices to be joined by springs instead of by rigid rods. The constant for these springs should be such that only small variations in length are permitted. For instance, we can choose the same constant for all of them as the maximum value of the contour springs obtained with the procedure above described. Then, a polygon segment will stretch only if necessary. The resulting NLP problem for this extension is unconstrained, because the constraints are now imposed by the resistance of the segment springs to stretch. The objective function becomes:

$$F(X) = \sum_{i=0}^{n-1} \sum_{j \in R_i} V_{i,j} + \sum_{i=0}^{n-1} U_{i,i+1} \quad (3.22)$$

where $U_{i,i+1}$ is the potential energy stored in “spring” segment $\overline{s_i s_{i+1}}$ of length l . The problem however is not more an equilateral polygon approximation, because we are allowing the segments to vary in length.

It is worth mentioning that the spring lengths have been assumed to be equal zero in both cases.

3.5 Concluding remarks

Three objective functions for the problem of Equilateral Polygon Approximation have been presented. By using NLP, the problem of equal segment lengths is easily handled: it reduces to a set of constraints. However, another problem arises, namely the definition of appropriate objective functions. In this chapter, an attempt has been made to develop such functions and it is believed that the energy method presented is the best suitable (among the three) for the problem.

All the approaches were designed with the intention of moving polygon vertices as close as possible to the object contour.

The first one is a least-squares function. A polygon segment is moved close to

the contour by minimizing orthogonal distances from contour points to it. Since each segment considers a variable set of points during the minimization, convergence is not guaranteed. The second and third objective functions are intended to minimize area differences. One of them minimizes areas of the regions enclosed between the original curve and the polygon. The other area function minimizes the difference of the total areas of polygon and object. An additional set of constraints had to be added in order to assure that the final polygon contour were close to the curve. Each polygon vertex was restricted to move in a fixed circular region whose center was located very close to the contour. The last objective function is based on a mechanical analogy. Each polygon vertex is attached to a number of springs which are anchored to the contour points. In this way the points are pulled close to the contour such that the potential energy stored in the springs is minimized and the system stabilized. By computing different spring constants we can weight the contribution of each spring to the final result. For those springs attached to high curvature points, we assign larger constants than for those attached to flat and long sections.

Each one of these methods deserve its own detailed study. Since this would be an enormous task beyond the scope of the present work, we have decided to study only the spring method in depth. This approach is selected because it is actually the most original of the three, it is mathematically well defined, and the results obtained in early experiments with it are much better than for the others in terms of looking appearance.

Chapter 4

Support Functions

In this chapter we describe two algorithms that were necessary to develop during the development of this work. These are a contour following algorithm to extract the contour of an object, and a contour coding algorithm in order to map a polygon into a sequence of symbols.

4.1 Contour following

For a fast execution of the polygonal approximation algorithm, the input must be provided as a sequence of points $C = c_0, \dots, c_{n-1}$ where each point is a pair (x, y) representing the coordinates of the corresponding pixel in the image, and c_{i+1} is one of the 8-neighbors of c_i .

The objective of a *contour following* algorithm is to produce this sequence.

The input to a Contour Following Algorithm (CFA) is generally an image which has been segmented in an appropriate way. For instance we can apply a region growing algorithm or a simple global thresholding procedure [2]. The purpose of this preprocessing step is to produce two well defined regions in the

Algorithm: *Contour following.*

Input: Segmented binary image $I(x, y)$, with $0 \leq x, y \leq M - 1$, and $I(i, j) = 0$ for background and $I(i, j) = 1$ for object $\forall 0 \leq i, j \leq M - 1$, and initial point c_0 .

Output: Sequence of contour points $S = c_0, c_2, \dots, c_{n-1}$, $c_i = (x_i, y_i)$ such that for all $0 \leq i \leq n - 1$, $\#\{q \mid q \in 8\text{-neighborhood}(c_i) \wedge I(c_i) = 1 \wedge I(q) = 0\} \geq 2$.

```

direction = 3;
p = c0;
while (NOT FOUND)
{
    direction = (direction+6) mod 6;
    while ( I(neighbor(p,direction)) ≠ 1 and neighbor(p,direction) ≠ c0 )
        direction = (direction+1) mod 8;
    if (neighbor(p,direction) ≠ c0)
    {
        p = neighbor(p,direction);
        i = i + 1;
        ci = p;
    }
    else
        FOUND= TRUE;
}

```

Figure 4.1: Contour following algorithm.

image: object and background. We assume that the input image is binary, where the background is white pixels (0) and the object is black pixels (1). Let O be the set of pixels belonging to the object and let B be the set of background pixels. Then in an 8-connectivity adjacency image a contour point is defined as a pixel from O whose 8-neighborhood has at least two pixels from B .

Figure 4.1 displays a pseudo-code for the contour following algorithm.

Notice that the algorithm is given the initial point c_0 where to start the search for adjacent points in the sequence. This point can be easily found using a left-to-right, top-to-bottom row scan search in the image.

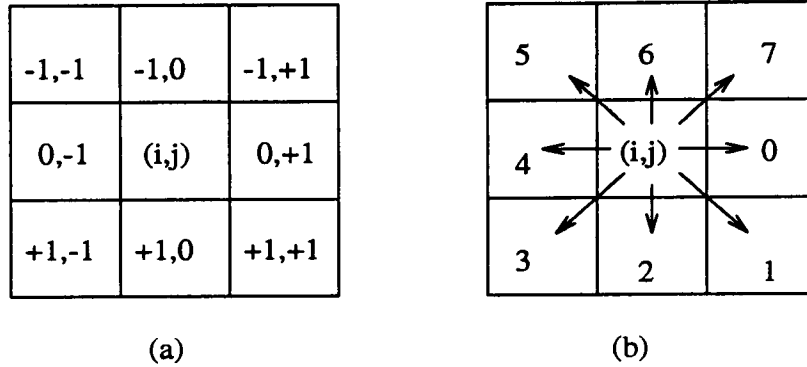


Figure 4.2: 8-Neighborhood and associated directions.

Starting at point c_0 the algorithm sets a *direction* flag in order to “remember” in which direction the last point in the sequence was found. This direction is very important because it drives the search for the next point in the neighborhood. Figure 4.2 shows the 8-neighborhood of a pixel at position (i, j) and the directions assigned to each neighbor. Since c_0 is always found by a left-to-right scan the initial flag is set to 0.

It is impossible for a contour point to be in a position that was already checked in a previous iteration. For instance, in Figure 4.3 c_{i+1} cannot be in position 6. Otherwise, it would have been found when searching c_{i-1} ’s neighborhood. The first position to check is is therefore always $(direction+6) \bmod 8$.

We will show how the algorithm works with an example. With reference to Figure 4.3 let c_i be the last point found. Then to find the next contour point we scan sequentially (in a clockwise fashion) through the neighborhood of c_i starting with its neighbor located at $(direction+6) \bmod 8$, that is 7. Directions 0 and 1 are also checked by incrementing the direction flag. Finally, c_{i+1} is found in the fourth iteration, and the new *direction* becomes 2.

The search is continued until a contour point is found or the initial point c_0 is reached, in which case the algorithm terminates.

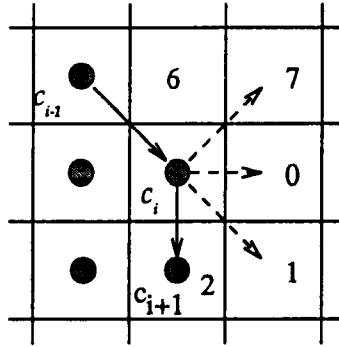


Figure 4.3: Illustration of Contour Following. c_{i+1} is found by a clockwise search in the neighborhood of c_i .

The maximum number of steps of this contour following algorithm is always less than $7n$, where n is the number of points in the input sequence. However, it is very likely that adjacent contour points are “along” similar directions and therefore the average number of steps is expected to be much less than $7n$.

The algorithm presented here assumes that there is only one object present in the image and that the contour is closed. For interior contours (holes in the object) the algorithm must be modified such that the spanning of the neighborhood is carried out counter clockwise. Also, it is assumed that the object is at least two pixels wide.

4.2 Contour coding

The purpose of a contour coding algorithm is to create a representation of the polygon other than the sequence of coordinate points. In the usual case the polygon is coded as a sequence of pairs (l_i, θ_i) , where l_i is the length of segment i and θ_i is the angle between the segment $\overline{s_{i-1}s_i}$ and some reference line. When the reference line is fixed, for instance one of the coordinate axis, the coding becomes highly dependent on the orientation of the object, and a registration preprocessing

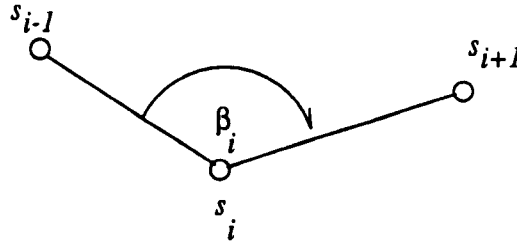


Figure 4.4: θ_i is the internal angle between two segments.

is necessary. This preprocessing consists of a spatial alignment between model and object. The object is translated and rotated in order to match the model's origin and reference line.

For our specific case of EPA, all the polygon segments have the same length; l_i is redundant and θ_i is the only parameter needed to be coded. If we allow the reference line to vary from angle to angle, such that it always coincide with segment $\overline{s_i s_{i+1}}$, we obtain a coding which is invariant to translation, rotation and scaling. Thus, θ_i is chosen to be the angle formed between $\overline{s_{i-1} s_i}$ and $\overline{s_i s_{i+1}}$. The angle is measured clockwise from $\overline{s_{i-1} s_i}$ to $\overline{s_i s_{i+1}}$ as shown in Figure 4.4.

The method to obtain the angles is quite simple. First, we recall that since the angles are allowed to take on any value in $[0, 2\pi]$, a simple cosine law will not work. Let $s_1 = (x_1, y_1)$, $s_2 = (x_2, y_2)$, and $s_3 = (x_3, y_3)$ be any three polygonal vertices, and let also θ_2 be the angle formed by the segments $\overline{s_1 s_2}$ and $\overline{s_2 s_3}$ as shown in the Figure 4.5.

The first step is to translate the origin to s_2 . This is done by simple differences:

$$s'_1 = s_1 - s_2$$

$$s'_3 = s_3 - s_2$$

Next we compute θ by taking the difference between β_1 and β_3 as follows:

$$\beta_1 = \begin{cases} \cos^{-1} \left(\frac{x'_1}{l} \right) & \text{if } y'_1 \geq 0 \\ 2\pi - \cos^{-1} \left(\frac{x'_1}{l} \right) & \text{otherwise} \end{cases}$$

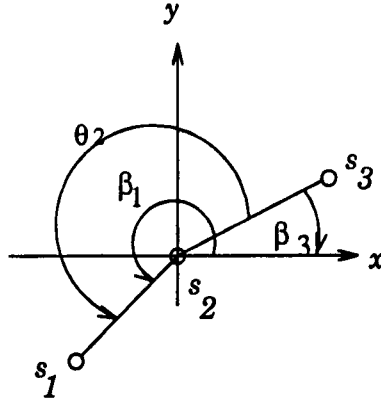


Figure 4.5: Computing angle θ_2 .

$$\beta_3 = \begin{cases} \cos^{-1} \left(\frac{x'_2}{l} \right) & \text{if } y'_3 \geq 0 \\ 2\pi - \cos^{-1} \left(\frac{x'_3}{l} \right) & \text{otherwise} \end{cases}$$

where l is the segment length. Finally, we compute:

$$\theta_2 = \begin{cases} \beta_3 - \beta_1 & \text{if } \beta_3 - \beta_1 \geq 0 \\ 2\pi + \beta_3 - \beta_1 & \text{otherwise} \end{cases}$$

This formulation allows the angles to take on the full range $[0, 2\pi]$ and they are in general real numbers. For string matching this would imply an infinite alphabet. In order to get a finite number of symbols (finite alphabet), the angles must be quantized. Quantization factor q is an integer value in $(0, 2\pi]$ indicating that all new angles must be integer multiples of q , that is

$$\theta'_i = \lceil \theta_i / q \rceil$$

A quantization value q implies the alphabet has $2\pi/q$ symbols.

The description of the algorithm and a pseudo code is shown in Figure 4.6.

Algorithm: *Contour Coding*

Input: A sequence of polygonal vertices $s_i = (x_i, y_i)$, $0 \leq i \leq n - 1$, quantization level q , and the initial vertex s_j to start the coding.

Output: A sequence of quantized angles θ_i , $0 \leq i \leq n - 1$ of interior angles formed between segment s_i and s_{i+1} ($0 \leq \theta_i \leq 2\pi$).

```
{
  for (each vertex  $s_i$ ,  $i = j, j + 1, \dots, 0, 1, \dots, j - 1$ )
  {
     $P_1 = s_i$ ;
     $P_2 = s_{i+1}$ ;
     $P_3 = s_{i+2}$ ;
     $P'_1 = P_1 - P_2$ ;
     $P'_3 = P_3 - P_2$ ;
     $\beta_1 = \cos^{-1} \left( \frac{x'_1}{l} \right)$ ;
     $\beta_3 = \cos^{-1} \left( \frac{x'_3}{l} \right)$ ;
    if ( $y'_1 < 0$ )
       $\beta_1 = 2\pi - \cos^{-1} \left( \frac{x'_1}{l} \right)$ ;
    if ( $y'_3 < 0$ )
       $\beta_3 = 2\pi - \cos^{-1} \left( \frac{x'_3}{l} \right)$ ;
     $\theta_i = \beta_1 - \beta_3$ ;
    if ( $\theta_i < 0$ )
       $\theta_i = 2\pi + \theta_i$ ;
     $\theta_i = \lceil \theta_i / q \rceil$ ;
  }
}
```

Figure 4.6: Contour coding algorithm.

Chapter 5

Experimental Results

5.1 Introduction

In this section we advocate to the task of evaluating the polygon approximations obtained with the energy method. After presenting the results of different polygon approximations, we evaluate their “quality” and study their pattern classification performance.

The PA’s were obtained using variation 1, which corresponds to assigning different spring constants to each contour point according to the curvature of the curve at that point. Some of the reasons for this decision are that in early experiments we found that the PA’s obtained with different spring constants were always better than the others in terms of visual appearance. Besides the complexity of the problem is not incremented with this variation. Only an $O(n)$ initial step is added to the total run time for computing spring constants.

The quality analysis consists in comparing the PA’s with the corresponding original contours. In this respect, we can think of this evaluation as computing similarity measures to see how closely the approximations resemble the true

shapes. The set of measures chosen are: the mean distance from the centroid to the contour, the variance of the distance to the centroid, three spatial second-order moments, and the area deviation. The mathematical definitions of these measures will be given in Section 5.4. The purpose of doing a classification analysis is to measure the recognition capabilities of the PA's. This is a very important evaluation because the purpose of developing an equilateral polygon approximation was to do pattern recognition of two dimensional shapes in only one dimension.

The nonlinear programming minimization of the objective function was accomplished by making use of NAG software ¹. In particular, routine *E04UPF* which minimizes a sum of squares was used. This routine is based on a successive minimization of a quadratic function to obtain the step length for the next iteration. Further details can be found in appendix B. All associated routines with this method were written in FORTRAN and executed on a VAX-6000 machine running the VMS operating system. All other routines were written in C programming language and executed on a SPARC SUN workstation under UNIX. A specialized software package called Khoros was used to handle graphics and image routines. All support functions were fully integrated in this environment. A description of Khoros is given in Appendix A.

5.2 Data material

The set of objects for the experiments consists of three pliers, six screwdrivers, and five wrenches. We only display some of these objects here. As the reader can see in Figure 5.1 two of the pliers are very similar in shape. The major

¹NAG is a registered trademark of Numerical Algorithms Group Limited, Numerical Algorithm Group Inc.

differences are that plier 1 is a little smaller than plier 2, and that plier 2 has two tips in its handle that are not in plier 1. A similar situation occurs with the two screwdrivers S_1 and S_3 . The extreme case occurs for W_4 and W_5 , which are only slightly different in size. Such small differences are very hard to detect when only a fraction of the total information is used to describe shapes. This is exactly what will happen during recognition, because the coding phase will reduce the information according the quatization level specified. Therefore, we will assume the these pliers will conform a *macroclass plier*, and letter P will be used as the name for that macroclass. Similarly, the six screwdrivers will conform *macroclass screwdriver* S , and the five wrenches *macroclass wrenches* W . Individual objects will be called *microclasses*. Therefore, in total we have the three macroclasses P , S , and W and fourteen microclasses $P_1, P_2, P_3, S_1, S_2, S_3, S_4, S_5, S_6, W_1, W_2, W_3, W_4$, and W_5 .

Digital images of the objects were captured by a solid-state CCD camera. Backlighting was used in order to enhance the contrast of the images. Some preprocessing was carried out in order to get binary images. First, gray level images were obtained by extracting the colormap from the images' headers, and then a global thresholding operation was done to get black and white images. An additional preprocessing step was necessary to reduce the size of the images by one half. For all these tasks the Khoros software was used. It was not necessary to write any new routine. In the last step, the contour following algorithm described in Section 4.1 was used to extract the contours of the objects. Although this function had to be written, it was finally integrated into Khoros. A complete description of Khoros can be found in [6].

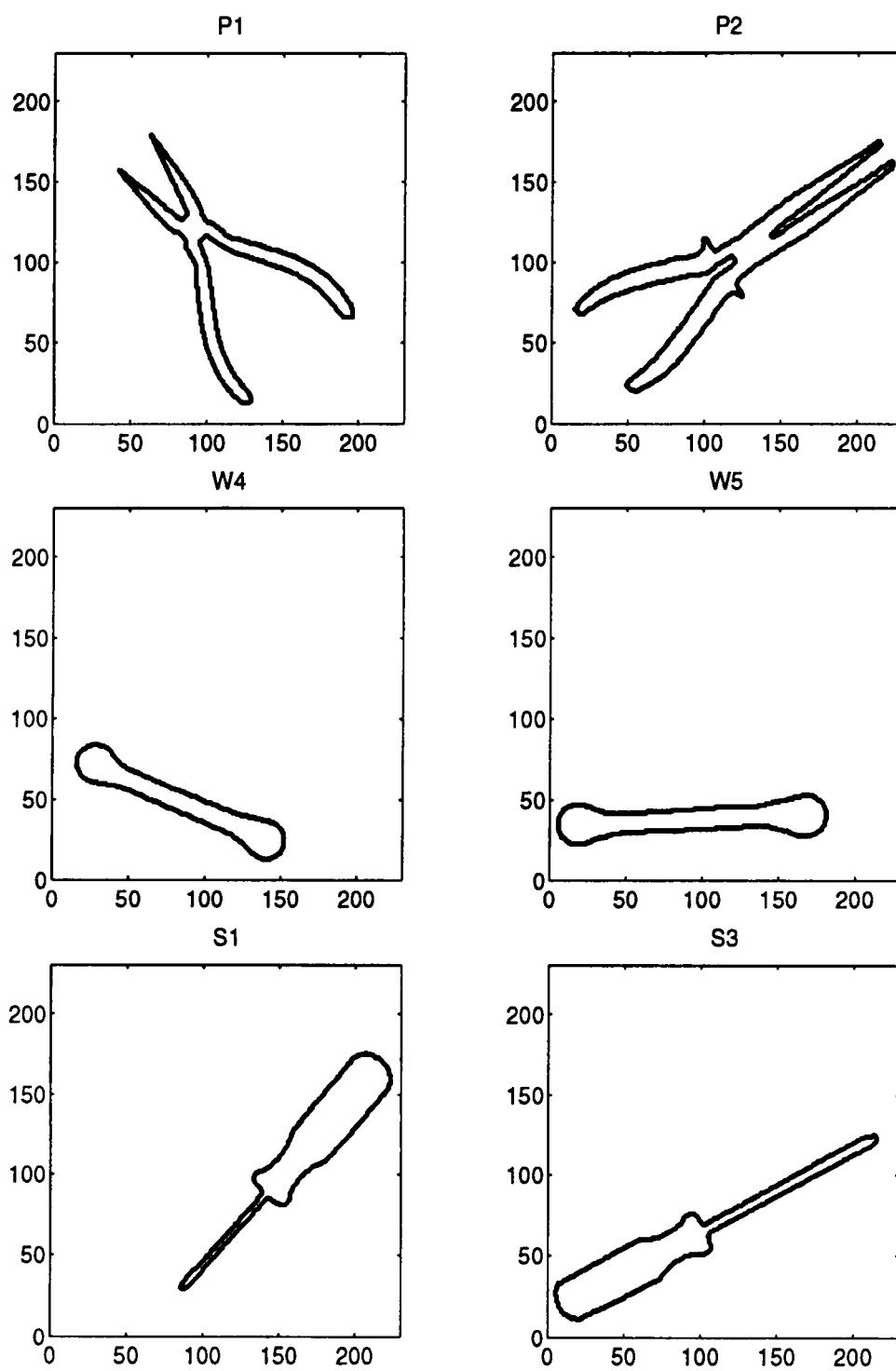


Figure 5.1: Some of the objects used in the experiments: pliers 1 and 2 (P_1 , P_2), wrenches 4 and 5 (W_4 , W_5), and screwdrivers 1 and 3 (S_1 , S_3).

5.3 Results and discussion

For each microclass PA's with segment lengths 5, 7, 9, 11, and 13 were obtained. Each PA is labeled with its macroclass name and segment length. Thus, $P_1(9)$ means PA of plier 1 with segment length 9. In total there are 70 PA's, but for reason of space we only show the results for lengths 5 and 11 for each microclass (see Figure 5.2 to 5.6).

In this section the only analysis about the PA's that can be done is by visual inspection. From the entire set of results we can conclude that for most of the PA's with segment length of 5 and 7, the polygons look very similar to the original shape. The only exception is for macroclass W . In particular, the algorithm was not able to capture important details when the approximations were done with relatively large segment lengths, like 9, 11, and 13 (see page 54 for $W_3(11)$). However, this behavior was expected. If an object has a characteristic feature of a certain length, it is very likely that an approximation with a larger segment length will not be able to represent that feature. A good approximation will be obtained if a priori information about global and local properties of the shape can be used to choose an appropriate segment length. Objects with a high degree of irregularities in the contour should be approximated with smaller segment lengths than objects with long and straight segments. However, the algorithm was able to retain the handle "tips" in microclass P_2 for all the lengths, which is one of the features that distinguish microclass P_1 from microclass P_2 .

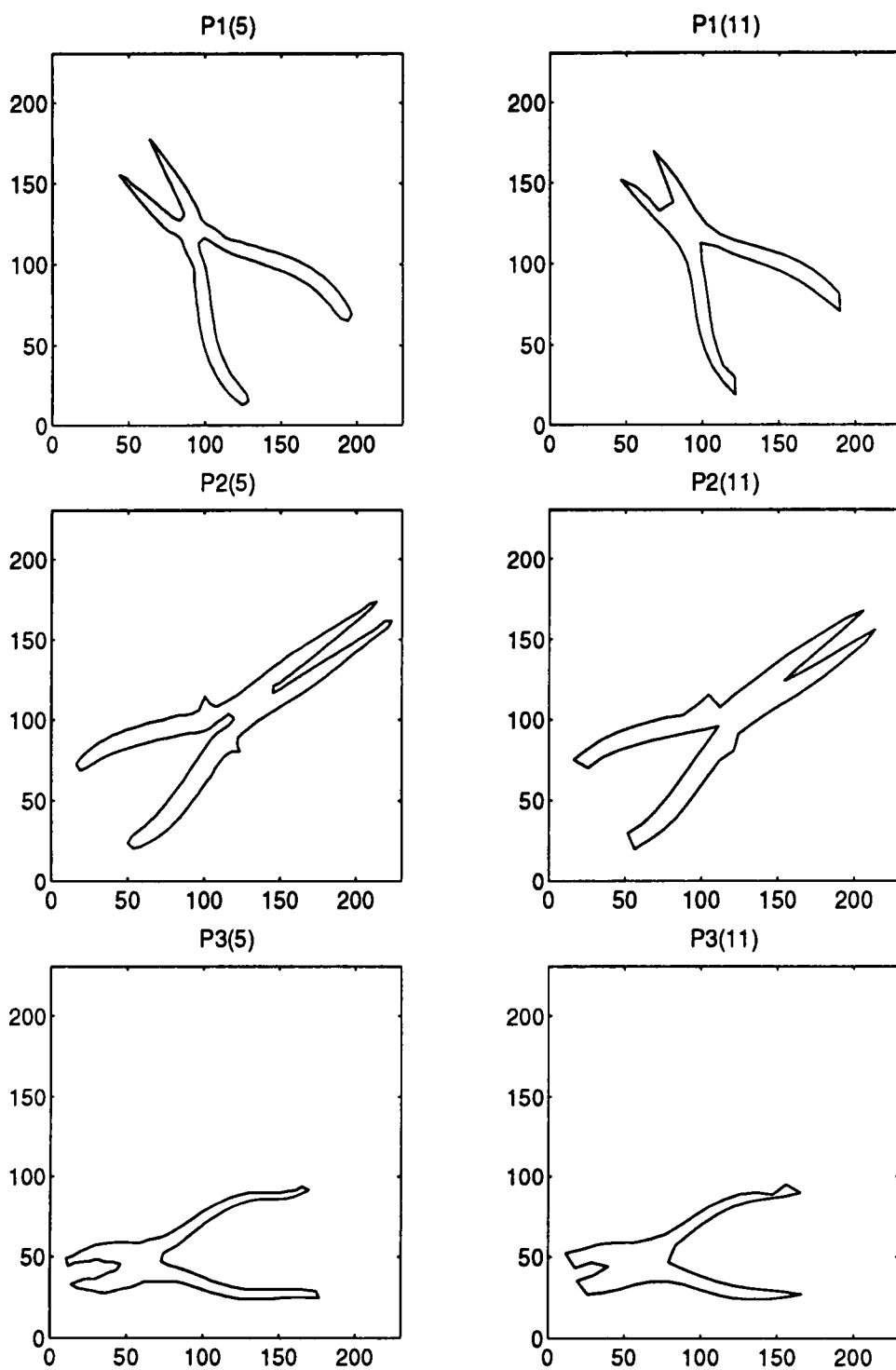


Figure 5.2: Pliers 1, 2, and 3 approximated with lengths 5 and 11

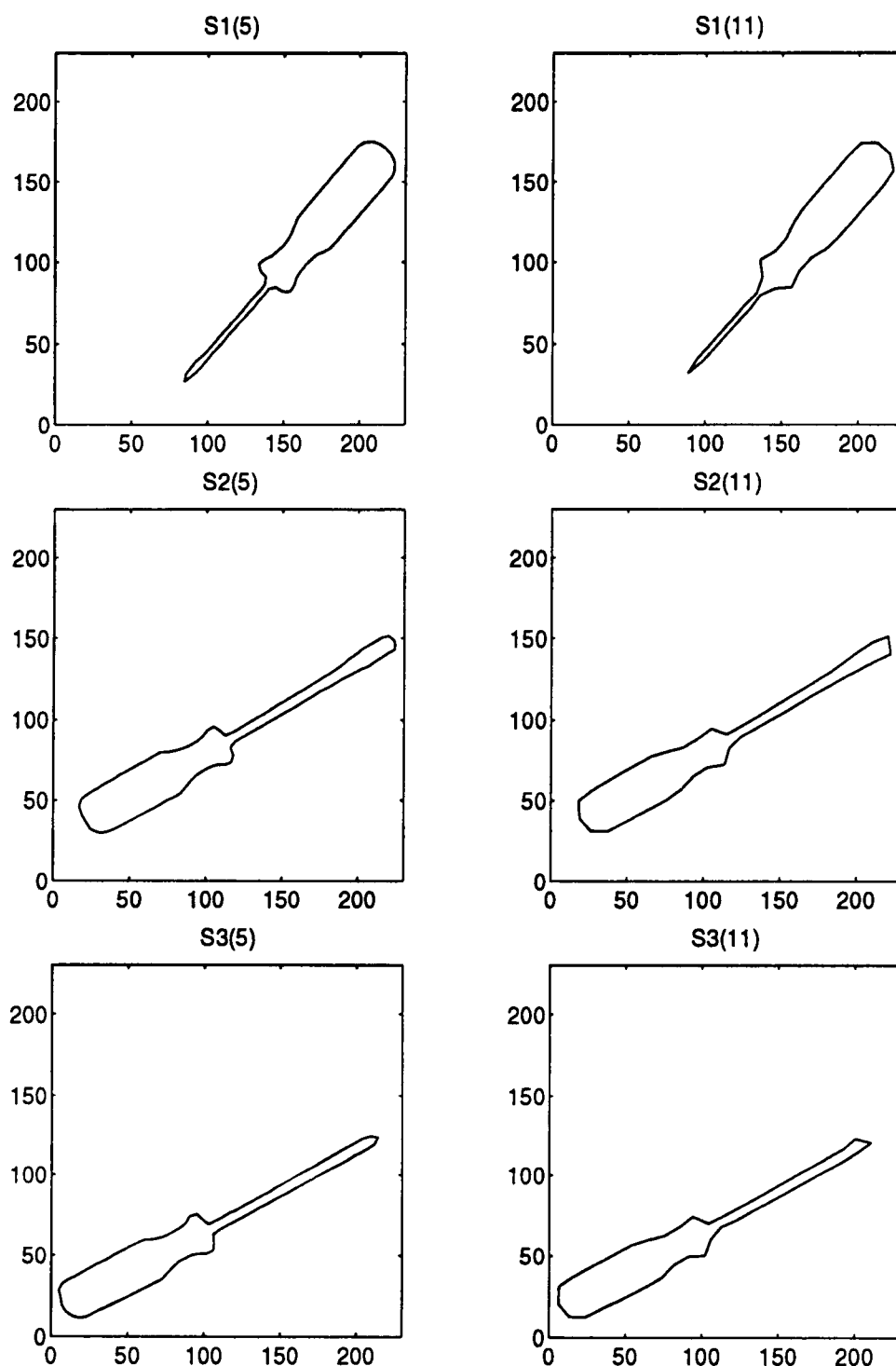


Figure 5.3: Screwdrivers 1, 2, and 3 approximated with lengths 5 and 11

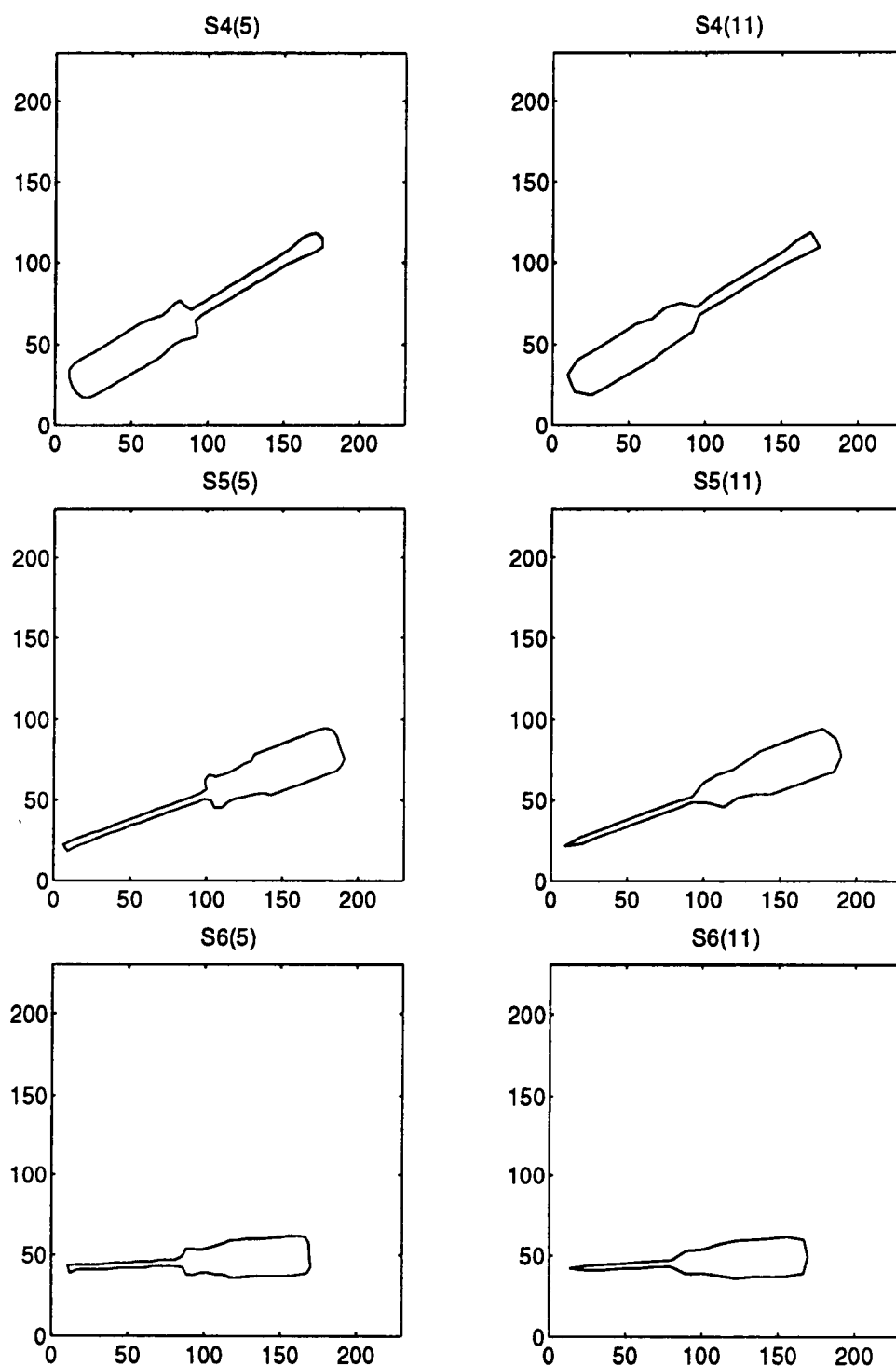


Figure 5.4: Screwdrivers 4, 5, and 6 approximated with lengths 5 and 11

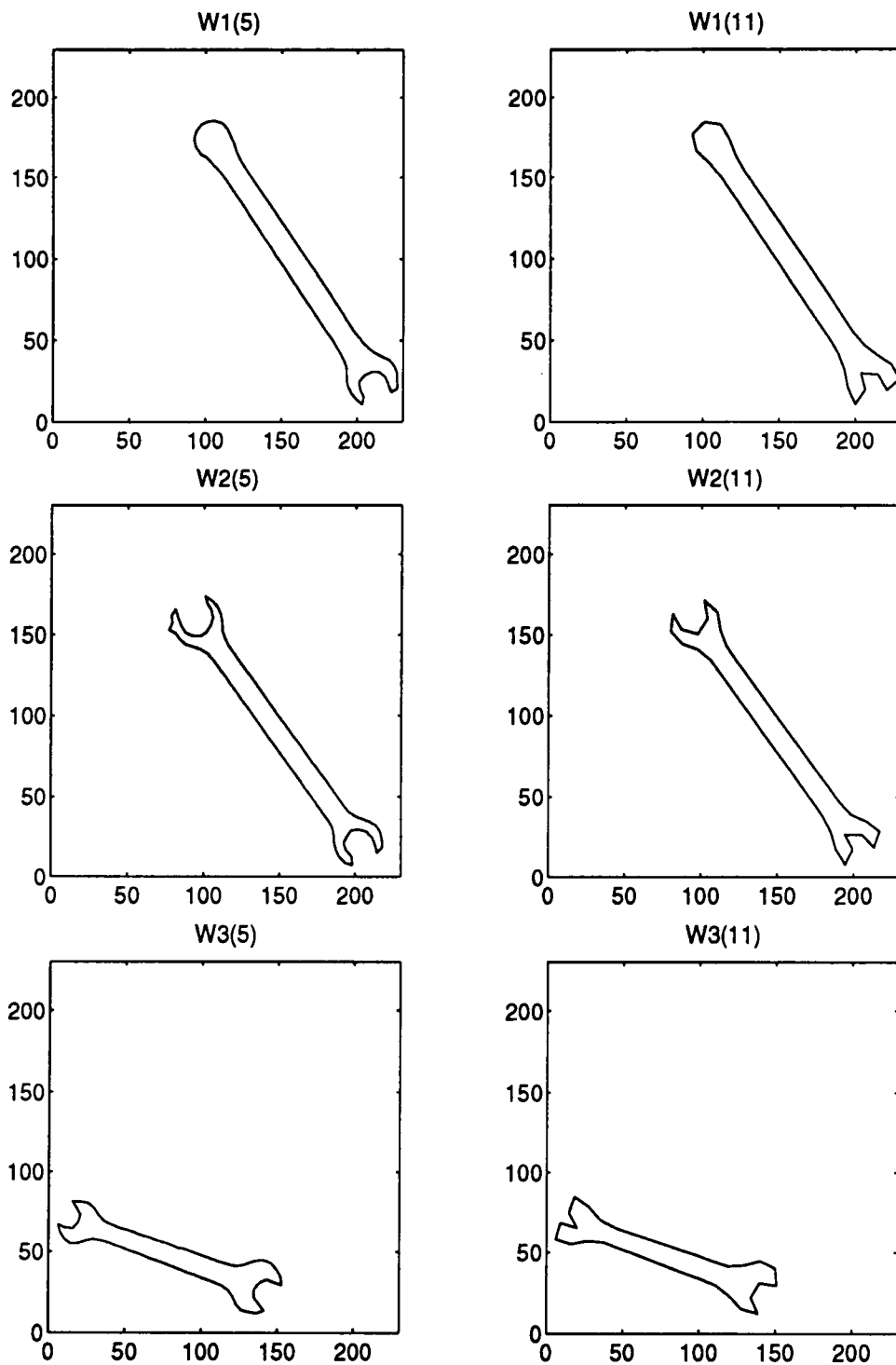


Figure 5.5: Wrenches 1, 2, and 3 approximated with lengths 5 and 11

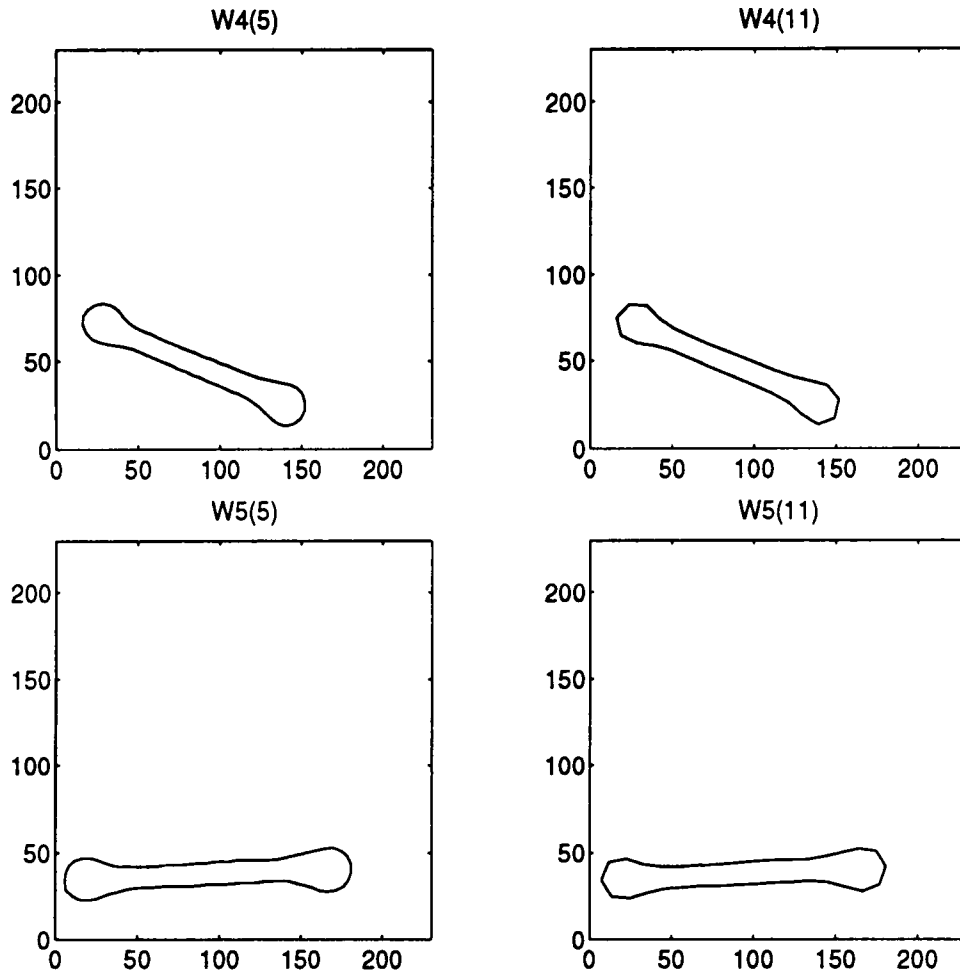


Figure 5.6: Wrenches 4 and 5 approximated with lengths 5 and 11

From the point of view of timing the method is slow when the approximation is made with a large number of segments (short segments). The main reason for this is the computational complexity of the NLP algorithm, which is $O(n^3)$, where n is the number of variables. This property is clearly reflected in the run time behavior of the algorithm. The evaluation of the objective function (potential energy) does not significantly affect the total execution time of the program. Although the objective function and the constraints are evaluated many times (as many as necessary to converge to an optimal solution), they have only a complexity of

$\Theta(m)$ and $\Theta(n/2)$ respectively, where m is the number of contour points and $n/2$ is the number of vertices. The evaluation of objective and constraint gradients were also provided explicitly and their evaluations are $\Theta(n)$ and $\Theta(n/2)$. The real “bottleneck” is found to be in the optimization process itself. Specifically, the algorithm takes a long time when doing a linear search and when optimizing a quadratic subproblem (see appendix B). Therefore, not much will be gained in optimizing the objective function and constraint evaluations. A real improvement in the execution time would be obtained if the NLP routine were parallelized.

5.4 Experiment I: Qualitative assessment

In this section we evaluate the resulting PA’s and compare them with the original contours. However, we already know that the quality of a PA cannot be computed exactly, that is there is no finite set of measures to compute it. If so, we would have used them in an objective function to get the PA’s. Despite of this, we have decided to compute a set of six measures for each approximation. These measures are usually used in shape analysis for measuring region and contour properties [7]. It is worth noting that none of the following quantities capture the complete description of an object. In order to do so, an infinite number of measures would be necessary. For instance, with an infinite number of projections an object can be reconstructed and therefore completely described. Nevertheless, the purpose of our experiment is not to reconstruct a contour from an approximation, but to see how close some of the PA’s global properties are to the actual values. In other words, we want to see if the PA algorithm is able to retain these properties and how they are affected by the segment length. Therefore, we consider only a reduced set of properties assuming they will give us enough information to extract

valid conclusions.

The measures are the following:

1. μ_s is the mean distance from the centroid to the contour.

$$\mu_s = \frac{1}{n} \sum_{i=0}^{n-1} \| (c_{ix}, c_{iy}) - (O_x, O_y) \| \quad (5.1)$$

where (O_x, O_y) is the centroid of the object.

2. σ_s^2 is the variance of the distances from the centroid to the contour.

$$\sigma_s^2 = \frac{1}{n} \sum_{i=0}^{n-1} \left(\| (c_{ix}, c_{iy}) - (O_x, O_y) \| - \mu_s \right)^2 \quad (5.2)$$

3. μ_{xx} , μ_{yy} , and μ_{xy} are the spatial second-order “row”, “column”, and “mixed” moments respectively.

$$\mu_{xx} = \frac{1}{n} \sum_{i=0}^{n-1} (c_{ix} - O_x)^2 \quad (5.3)$$

$$\mu_{yy} = \frac{1}{n} \sum_{i=0}^{n-1} (c_{iy} - O_y)^2 \quad (5.4)$$

$$\mu_{xy} = \frac{1}{n} \sum_{i=0}^{n-1} (c_{ix} - O_x)(c_{iy} - O_y) \quad (5.5)$$

The second spatial moments have value and meaning for a contour of any shape, the same way that the covariance matrix has value and meaning for any two-dimensional probability distribution [7].

4. δA is a measure proportional to the area deviation between the PA and the original contour. The same technique used in Section 3.2 to compute the least-squares is used here. Notice that this is not the same as difference of areas.

5.4.1 Results

The final results are tabulated in Tables 5.1 to 5.4. Each table contains values for five different segment lengths, namely 5, 7, 9, 11, and 13 corresponding to one microclass. Measures μ_s , σ_s^2 , μ_{xx} , μ_{yy} , and μ_{xy} are expressed as percentages relative to the true values of the original contours which are listed in the *original* column. An entry with value one means that the specific measure for the approximation has the same value as the original. Values near zero and much greater than one represent high disparities. For measure δA , the values are relative to the true area of the original object, which is listed in the *original* column. In this case, values near zero express that there is very small area deviation between the approximation and the original contour. This means that the polygon segments are close to the contour.

Again, we only show the results for some of the microclasses: P_1 , P_3 , S_3 , and W_1 . The behavior shown in these tables is also found to apply for the approximations of the other objects.

Since μ_s , σ_s^2 , and the second-order moments have been normalized, we can average them over all the microclasses and see how they change with respect to the segment length. Figures 5.7 and 5.8 show these results. Remember that the ideal case for these curves is the constant function one. These graphs show that as long as we keep incrementing the segment length, region properties for an approximation will be more and more different than the actual values. A detailed discussion of these results follows.

Table 5.1: Values for microclass P_1 .

<i>length</i>	5	7	9	11	13	<i>original</i>
R	0.22	0.16	0.12	0.09	0.08	616
μ_s	1.01	1.01	1.01	0.96	0.97	49.27
σ_s^2	0.97	0.96	0.96	0.83	0.85	586.31
μ_{xx}	1.04	1.03	1.03	0.96	0.95	1379.89
μ_{xy}	1.02	0.99	1.00	0.84	0.87	-763.99
μ_{yy}	0.99	0.98	0.98	0.85	0.91	1634.43
δA	0.02	0.03	0.03	0.07	0.07	2821

Table 5.2: Values for microclass P_3 .

<i>length</i>	5	7	9	11	13	<i>original</i>
R	0.21	0.15	0.11	0.08	0.07	611
μ_s	0.98	0.98	0.95	0.94	0.94	48.58
σ_s^2	1.01	1.01	0.98	0.94	0.94	482.03
μ_{xx}	0.96	0.96	0.92	0.87	0.88	2319.62
μ_{xy}	1.00	0.93	0.68	1.22	0.98	218.47
μ_{yy}	0.98	0.99	0.93	1.01	0.97	522.12
δA	0.02	0.02	0.04	0.06	0.07	2594.00

Table 5.3: Values for microclass S_3 .

<i>length</i>	5	7	9	11	13	<i>original</i>
R	0.22	0.16	0.12	0.10	0.08	449
μ_s	1.00	0.99	0.99	0.99	0.98	59.86
σ_s^2	0.98	0.96	0.95	0.94	0.92	1206.42
μ_{xx}	0.99	0.97	0.97	0.97	0.94	3760.86
μ_{xy}	0.99	0.97	0.97	0.97	0.95	1876.84
μ_{yy}	0.99	0.98	0.98	0.97	0.96	1028.54
δA	0.01	0.01	0.02	0.02	0.04	3374.5

Table 5.4: Values for microclass W_1 .

<i>length</i>	5	7	9	11	13	<i>original</i>
R	0.23	0.16	0.12	0.10	0.08	411
μ_s	0.99	0.98	0.97	0.98	0.95	59.59
σ_s^2	0.99	0.98	0.97	0.98	0.95	936.58
μ_{xx}	0.96	0.95	0.93	0.96	0.90	1553.99
μ_{xy}	0.96	0.94	0.92	0.95	0.91	-2048.97
μ_{yy}	0.96	0.93	0.91	0.95	0.91	2933.50
δA	0.01	0.01	0.02	0.03	0.0328	3322.5

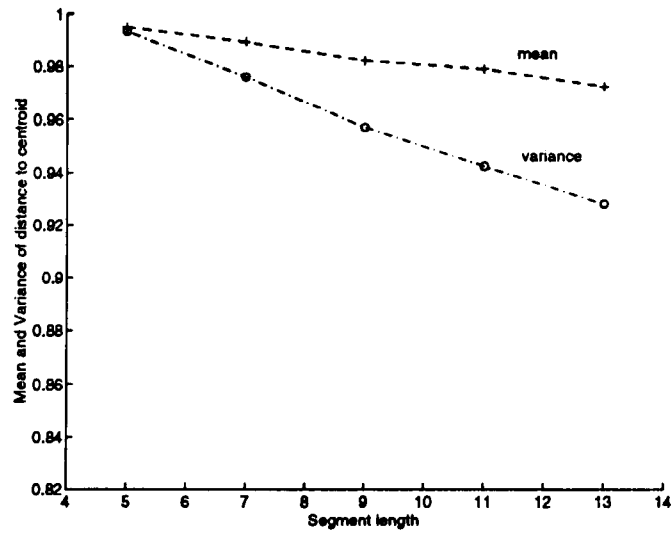


Figure 5.7: Variation of μ_s and σ_s^2 with respect to the segment length of an approximation. Data are averages over all 7 microclasses.

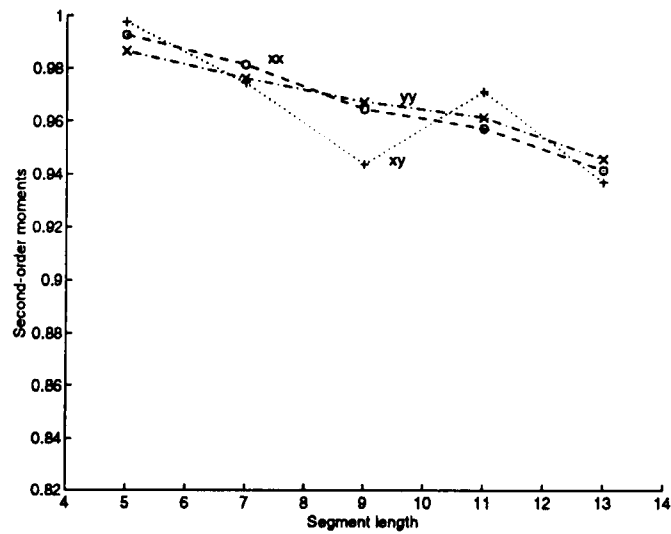


Figure 5.8: Variation of μ_{xx} , μ_{yy} , and μ_{xy} with respect to the segment length of an approximation. Data are averages over all 7 microclasses.

5.4.2 Discussion

The small set of measures we have chosen to measure contour properties have shown that by increasing the segment length required for a PA, we get approximations whose properties diverge more and more from those of the original contour, as expected. However, it is possible that for certain cases, a larger segment length will not necessarily produce “worse” individual measures, but a better one. For instance, in Table 5.1 there is an improvement in measure μ_{yy} when enlarging segment length from 11 (0.8525) to 13 (0.9102).

For the smallest segment length, all the PA’s closely resemble the shapes of their respective original objects. This conclusion can be drawn not only by a visual inspection of the polygons, but also from the tables. All the measures are within an absolute deviation of only 5% of their true values, and for some microclasses, like S_3 , the error is less than 2%. The largest error obtained is in the order of 18% and is found only for lengths 13 (see Table 5.1). It is possible that for certain applications an error of 10% is within permissible levels, but for error critical applications this would not be acceptable. Tradeoffs between percentage of information reduction (smoothing), error deviations, and computer run time clearly arises.

Another interesting observation that can be made is that the quantity μ_s is probably not appropriate for this kind of analysis. It can be seen from the “average” plots that it is not as affected as the other quantities are for increasing segment lengths. We say “probably” because there is still much study to be done with respect to region properties of PA’s and true shapes. The row and column second-order moments show the same pattern of behavior. Although they do not compute the same property, it is very likely that an average of them could reflect individual variations for larger lengths not shown here. We can also see from the

average plot, the abrupt change for the mixed moment for lengths 9 and 11. The main contributor to this change is produced by P_3 , whose values are shown in table 5.2. For a segment length 7 and for measure μ_{xy} there is an error of 7%. For the next segment length, namely 9, the error increases up to 32%, which is the largest error for all the approximations and for all microclasses. Notice that the next largest error for length 11 occurs for approximation W_2 and is only 10%.

The reader may notice that the variance also can be grouped with these three curves.

Of course, these observations are restricted to this particular experiment which is limited in number of macro and microclasses. A more definite study would require a large number of very different kinds of shapes, with different sizes and patterns. It also would require a more elaborate set of measures and a larger number of different segment lengths.

5.5 Experiment II: Classification performance

5.5.1 Method

The objective of a classification experiment is to determine class membership for a set of “unknown” samples given a set of prototypes. Although we have introduced the concept of macro and microclasses, we have been mainly concerned with the polygon approximation itself without considering which of the PA’s are class representatives or which are assumed to be unknown samples. One object has been chose from each macroclass as prototype (model) and for the sake of simplicity no special criterion was used to select them. Specifically, we have defined P_3 , S_4 , and W_3 as prototypes PA’s for macroclasses *Plier*, *Screwdriver*, and *Wrench* respectively and P_1 , P_2 , S_1 , S_2 , S_3 , S_5 , S_6 , W_1 , W_2 , W_4 , and W_5 as

test samples for their respective macroclasses.

The strings used for recognition were obtained with the coding method described in Section 4.2. This algorithm was also developed using some of the Khoros functions. Since the string matching algorithm implemented (see Section 5.5.2) does not support cyclic rotations, it is necessary to specify a start point (vertex) where to start the coding. When cyclic rotations are implemented, an algorithm can detect the best matching over all “shift” possibles and the start point need not be specified [4]. In our case, encoding of both unknown sample and prototype strings must begin at relatively close positions in their respective PA's. The program allows the user to specify the start point with a simple mouse operation. Moreover, a zoom of the polygon is supplied. This is a very convenient way of selecting a vertex when the image or the segment length is too small.

We have decide to use a very small quatization level, that is a small degree of smoothing. In particular, 360 symbols were chosen for the alphabet. This means that each angle was rounded to the nearest integer producing very fine-grain approximations.

As we said before, only one object from each macroclass is to be considered as prototype. The rest of them are to be classified as belonging to one of these prototypes. We could say that the purpose of this experiment is macroclass recognition (many to one) because two different objects can be assigned to the same class. It could also be the case, that the unknown samples are noisy versions of the prototypes. In that case the classification is aimed for microclass recognition (one to one), where two different objects should be classified in different classes.

The classification experiment is divided in 5 classification runs, one for each segment length. Thus, the first experiment consists in aligning all unknown strings produced from PA's of length 5 with all prototype strings produced from PA's with

length 5, yielding a total of 33 alignments. Then, the unknowns with length 7 with prototypes with length 7, and so on. The total number of alignments is 165.

5.5.2 String matching

In approximate string matching a test string is aligned with a prototype string through a sequence of edit operations. Each operation has a cost and the cost of the alignment is the minimum among all possible sequences of edit operations. Dynamic programming is used to find the sequence of edit operations yielding a minimum cost.

Let $\Theta = \theta_0, \dots, \theta_{n-1}$ be the prototype string representing a class polygon (macro or microclass) and let $\Phi = \phi_0, \dots, \phi_m$ be the string representing the unknown object. The objective of string matching is to find a sequence of edit operations to transform with minimum cost the string Φ into the string Θ . The algorithm makes use of a cost matrix $[d_{i,j}]$, $0 \leq i \leq n$, $0 \leq j \leq m$ to find such a sequence. Each entry $d_{i,j}$ represents the cost of aligning the prefix substring $\phi_0 \dots \phi_j$ with the prefix substring $\theta_0 \dots \theta_i$. Note that the first symbols in both strings are $\phi_0 = \theta_0 = \epsilon$. They are necessary as a starting condition in the matrix. The optimal cost of the alignment is the entry $d_{n,m}$. The basic idea of the algorithm is to find a path from $d_{0,0}$ to $d_{n,m}$ with minimum cost. A position (i, j) can be reached only from $(i-1, j-1)$, $(i, j-1)$, or $(i-1, j)$. Each transition from these predecessors corresponds to a specific edit operation. A *diagonal* move from $d_{i-1,j-1}$ to $d_{i,j}$ is a substitution of symbol θ_i by symbol ϕ_j . A *vertical* move from $d_{i-1,j}$ to $d_{i,j}$ is a deletion of symbol θ_i . An *horizontal* move from $d_{i,j-1}$ to $d_{i,j}$ is an insertion of symbol ϕ_j .

The cost function defines the costs for these moves and in our case it has been defined as:

$$d(\theta_i, \phi_j) = \min \begin{cases} \phi_j & + & d_{i,j-1} & \text{(Insertion)} \\ |\theta_i - \theta_j| & + & d_{i-1,j-1} & \text{(Substitution)} \\ \theta_i & + & d_{i-1,j} & \text{(Deletion)} \end{cases} \quad (5.6)$$

where θ_i and ϕ_j are the resulted quantized angles formed by segment $i - 1$ and segment i and segment $j - 1$ and j respectively. There are many other cost functions that could have been used. We have chosen this one because it has properties needed in our alignments. When the strings are very similar (small differences in angles) the function assigns low costs to substitute one symbol by the other one. It also penalizes the insertion and deletion of an angle proportionally to the angle itself.

Pattern classification is carried out by aligning an unknown sample with all the prototypes (modelling different classes). The decision rule is: *choose class k such that the cost of aligning the unknown sample to prototype string k is the minimum among all the classes.*

5.5.3 Results and discussion

The results are tabulated in Tables 5.5 to 5.9. Each entry in these confusion matrices is the minimum cost to align the test string to the prototype string. When an unknown sample is correctly classified, a $\checkmark$ sign is printed at the bottom of its column, otherwise an X sign is printed. For instance and with reference to table 5.5, the cost of aligning unknown sample S_1 to class prototype P_3 is 7747, to S_4 is 1675, and 3147 to W_3 . According to the classification rule in Section 5.5.2, S_1 is correctly assigned to class S_4 because is the minimum cost of the three.

Table 5.5: Confusion table for PA's with length 5 and quantization level 1

<i>Model</i>	<i>Unknown Samples- l = 5</i>										
<i>l = 5</i>	<i>P</i> ₁	<i>P</i> ₂	<i>S</i> ₁	<i>S</i> ₂	<i>S</i> ₃	<i>S</i> ₅	<i>S</i> ₆	<i>W</i> ₁	<i>W</i> ₂	<i>W</i> ₄	<i>W</i> ₅
<i>P</i> ₃	3028	9278	7747	5226	5356	8054	10455	5801	4897	11348	8969
<i>S</i> ₄	10178	16780	1675	3786	3572	1662	2645	3175	4461	3458	1465
<i>W</i> ₃	11606	18262	3147	5678	5470	3112	2073	4569	5673	2338	1947
<i>Status</i>	✓	✓	✓	✓	✓	✓	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>

Table 5.6: Confusion table for PA's with length 7 and quantization level 1

<i>Model</i> <i>l</i> = 7	<i>Unknown Samples- l</i> = 7										
	<i>P</i> ₁	<i>P</i> ₂	<i>S</i> ₁	<i>S</i> ₂	<i>S</i> ₃	<i>S</i> ₅	<i>S</i> ₆	<i>W</i> ₁	<i>W</i> ₂	<i>W</i> ₄	<i>W</i> ₅
<i>P</i> ₃	2012	6348	5652	3780	3913	5962	7619	4251	3722	7867	6258
<i>S</i> ₄	6941	11327	1199	2367	2294	879	2290	1960	2759	2518	1245
<i>W</i> ₃	8034	12622	2306	3968	3829	2000	1541	3105	3694	1471	1578
<i>Status</i>	✓	✓	✓	✓	✓	✓	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>

Table 5.7: Confusion table for PA's with length 7 and quantization level 1

<i>Model</i> <i>l</i> = 9	<i>Unknown Samples- l</i> = 9										
	<i>P</i> ₁	<i>P</i> ₂	<i>S</i> ₁	<i>S</i> ₂	<i>S</i> ₃	<i>S</i> ₅	<i>S</i> ₆	<i>W</i> ₁	<i>W</i> ₂	<i>W</i> ₄	<i>W</i> ₅
<i>P</i> ₃	2052	4735	3922	2240	2355	3871	5162	2753	2170	5453	4173
<i>S</i> ₄	5421	8028	901	2051	1866	614	1697	1478	2725	1978	896
<i>W</i> ₃	6073	8850	1725	3213	3044	1652	1193	2282	3243	1300	1244
<i>Status</i>	✓	✓	✓	✓	✓	✓	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>

Table 5.8: Confusion table for PA's with length 11 and quantization level 1

<i>Model</i>	<i>Unknown Samples- l = 11</i>										
<i>l = 11</i>	<i>P</i> ₁	<i>P</i> ₂	<i>S</i> ₁	<i>S</i> ₂	<i>S</i> ₃	<i>S</i> ₅	<i>S</i> ₆	<i>W</i> ₁	<i>W</i> ₂	<i>W</i> ₄	<i>W</i> ₅
<i>P</i> ₃	935	3619	2807	1590	1713	2789	3880	1721	1732	4026	3164
<i>S</i> ₄	3394	6004	760	1773	1586	600	1279	1502	1711	1433	749
<i>W</i> ₃	3936	6556	1520	2665	2498	1550	1147	2030	2077	1077	1001
<i>Status</i>	✓	✓	✓	✓	✓	✓	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>

Table 5.9: Confusion table for PA's with length 13 and quantization level 1

<i>Model</i>	<i>Unknown Samples- l = 13</i>										
<i>l = 13</i>	<i>P</i> ₁	<i>P</i> ₂	<i>S</i> ₁	<i>S</i> ₂	<i>S</i> ₃	<i>S</i> ₅	<i>S</i> ₆	<i>W</i> ₁	<i>W</i> ₂	<i>W</i> ₄	<i>W</i> ₅
<i>P</i> ₃	1024	2710	2596	1601	1632	2507	3537	1785	1739	3597	2749
<i>S</i> ₄	3132	4742	672	1213	1220	473	1219	935	1109	1369	677
<i>W</i> ₃	3812	5464	1368	2295	2304	1373	805	1741	1767	685	915
<i>Status</i>	✓	✓	✓	✓	✓	✓	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>

For classifications with length 5, the results are surprising. The total recognition rate is only 64%. Considering only prototype P_3 the recognition rate is however 100%. For S_4 is 80% and the lowest rate is obtained for W_3 , where only one test sample was correctly classified. Since experiments with the rest of the prototypes and test samples yield the same results the overall recognition rate is 64%.

The main performance degradation is due to the wrenches. W_4 is the only object recognized and is recognized for all lengths. On the other hand W_1 , W_2 , and W_5 are consistently misclassified for all lengths. The overall result seems to indicate that the problem could not be the polygon approximation but the choice of appropriate segment length to define a prototype and test samples. Figure 5.9 plots recognition performance for the three classes as a function of the segment length.

In order to support this idea, we analyze in more detail the behavior of recog-

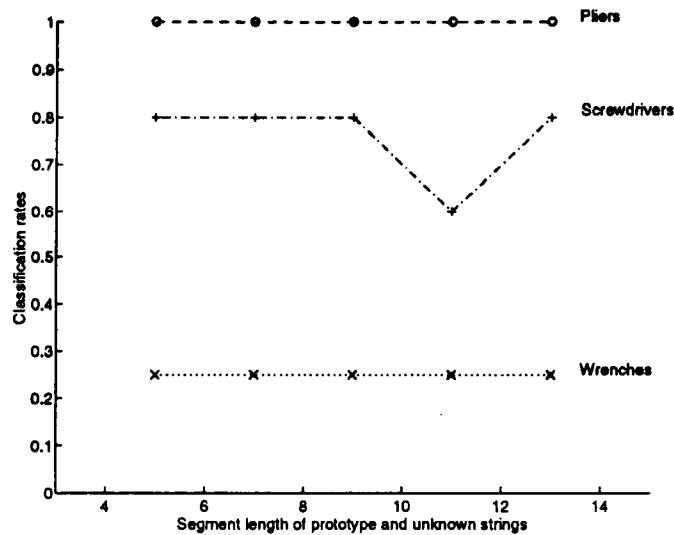


Figure 5.9: Total classification rates for the three macroclasses when prototype and unknown strings have the same segment length.

dition for class W_3 . The experiment consists now in classifying only wrenches with segment length 5 against the prototypes with different segment lengths. The objective of this experiment is to see how the recognition performance is affected for varying segment length. Tables 5.10 to 5.12 show the results for this experiment. The overall classification rate is now 85%. The only misclassifications were obtained for prototypes with length 5. If a similar experiment is run for wrenches with length 7 we get a lower value of 65% and for increasing lengths we get a decreasing sequence of recognition rates.

Similar experiments were run for the other two sets of objects and the results are plotted in Figure 5.10. A drastic decay in classification performance is observed for wrenches as the segment length of unknown samples is increased. An opposite behavior is observed for the pliers, for which increasing lengths mean better recognition performance. The curve for this set reaches the value 1 at lengths 11 and 13. This means that both unknown pliers are always correctly classified when they have length 11 or 13 and, more important for any prototype length.

Table 5.10: Confusion table for PA's with length 13 and quantization level 1

<i>Model</i> <i>l</i> = 5	<i>l</i> = 5				<i>Model</i> <i>l</i> = 7	<i>l</i> = 5			
	W_1	W_2	W_4	W_5		W_1	W_2	W_4	W_5
P_3	5801	4897	11348	8969	P_3	10819	10152	14517	12884
S_4	3175	4461	3458	1465	S_4	3307	2954	6557	4982
W_3	4569	5673	2338	1947	W_3	1883	1310	5181	3666
<i>Status</i>	X	X	$\checkmark$	X	<i>Status</i>	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$

Table 5.11: Confusion table for Macroclass W

<i>Model</i> <i>l</i> = 9	<i>l</i> = 5				<i>Model</i> <i>l</i> = 11	<i>l</i> = 5			
	W_1	W_2	W_4	W_5		W_1	W_2	W_4	W_5
P_3	13627	12826	16515	15203	P_3	15073	14982	17586	16690
S_4	5917	5434	8519	7225	S_4	7267	7426	9574	8678
W_3	4389	3842	7163	5851	W_3	5829	5786	8234	7338
<i>Status</i>	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	<i>Status</i>	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$

Table 5.12: Confusion table for Macroclass W

<i>Model</i> <i>l</i> = 13	<i>l</i> = 5			
	W_1	W_2	W_4	W_5
P_3	16497	16385	18465	17565
S_4	8603	8641	10453	9567
W_3	7205	7163	9113	8215
<i>Status</i>	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$

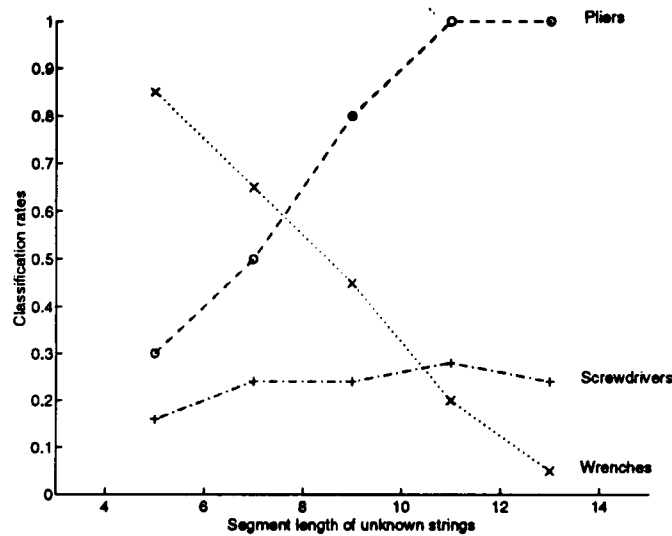


Figure 5.10: Classification rates when the segment length of unknown string is maintained fixed. The values shown are totals over all possible segment lengths for prototypes.

The behavior for screwdrivers is quite surprising. In the previous experiment, when prototype and test string had the same segment length, we obtained an 80% of recognition. Now using different lengths, the highest value is not greater than 30%.

It is clear from all these results that a more detailed and expanded set of experiments have to be carried out in order to extract more definite conclusions. The main observation that now can be made is that the segment length used to do a polygon approximation plays an important role in recognition environments. As we saw, the classification rates are very sensitive to these lengths. However, length by itself does not decide a recognition outcome. Shape is still one of the most important factors determining a classification result. For instance, consider the case of pliers. They are almost 100% of the time recognized with success for all lengths. It is not hard to see that they really conform a class with a very distinctive shape. On the other hand, wrenches and screwdrivers have some

similar contour properties. They are rather "elongated" and with a similar width in almost 50% of their body. When they are approximated with small segments, they can be easily differentiated, but as we increase the length they become more and more similar because the strong contour variations located in both extremes that distinguish a wrench from a screwdriver are rapidly lost.

It has been observed that when the number of segments of the unknown sample and the true prototype are similar, there is a high probability of a successful recognition if the rest of the prototypes are highly different. When an unknown sample is confronted with a set of prototypes with very similar shapes, there is high chance that the class selected is that with a similar number of segments. Therefore, it is desired that test objects and their respective prototypes be approximated with an equal or similar segment length.

Chapter 6

Conclusion

A novel approach has been presented to solve the problem of Equilateral Approximation for closed contours. The solution is based on the well-known mathematical technique nonlinear programming. The approximation has been modeled as a process of minimizing an objective function for which in the minimum the resulting polygon resembles the object shape and satisfies a set of nonlinear length constraints.

Four different objective functions were introduced. The first one is a least-squares function. A polygon segment is moved close to the contour by minimizing orthogonal distances from contour points to it. Since each segment considers a variable set of points during the minimization, convergence is not guaranteed. This is due to the fact that when a contour point is assigned to a different segment from that one of the previous iteration, the objective function is modified.

The second and third objective functions were intended to minimize area differences. One of them minimizes areas of the regions enclosed between the original curve and the polygon. We discovered that this approach suffers from the same deficiency of instability as the least-squares method. The other area function min-

imizes the difference of the total areas of polygon and object. An additional set of constraints had to be added in order to assure that the final polygon contour was close to the curve. Each polygon vertex was restricted to move in a fixed circular region whose center was located very close to the contour. Finding these centers turned out to be as hard as finding the initial polygon (which is required as a start guess of the solution). The points of this initial polygon were used as the centers of these regions. This area function is well defined, but the solution is not always satisfactory because of the region constraints.

The last and most valuable objective function is based on a mechanical analogy. Each polygon vertex is attached to a number of springs which are anchored to the contour points. In this way the points are pulled close to the contour such that the potential energy stored in the springs is minimized and the system stabilized. Different spring constants were assigned depending on a modified curvature function. Points with high curvatures are assigned larger constants such that the point attached to it will be located very close to it.

When having springs with length zero, the objective function becomes convex and an optimal solution is guaranteed.

Experiments were carried out using the energy method over a set of fourteen objects. The solutions obtained for small segment lengths are quite satisfactory for all the objects. When the segment length is large with respect to the object perimeter, some shape properties could be lost, thereby indicating that an adequate choosing of length is critical for obtaining good approximations.

Two assessment experiments were also made. The first experiment compared some shape measures of the approximations with their respective original objects. As expected, these properties became more and more different from the actual ones as we increased the segment length.

The second experiment was a classification performance evaluation where approximate string matching was used as the classification technique. The solutions for these experiments are not totally satisfactory. For some of the objects high recognition rates were achieved, but for others poor results were obtained. It was shown that the segment length is an important variable influencing the classification outcome. For some objects, longer segment lengths implied better results, but for others implied worse results. It is believed that this behavior is more related to the number of segments than to the length itself. It seems that the cost function used for classification prefers to match objects with similar number of segments over objects with similar shapes but with a very different number of segments.

We consider the latter experiment as very useful and not discouraging, because it has gave us valuable information to guide future research. Some of the questions that should be addressed are:

1. What is be the most appropriate segment length for a given object?
2. Should an unknown object be approximated with the same length as the prototype's to be aligned?
3. Are other cost functions more suitable for this type of problems?

The determination of segment length should be automatized. For instance, instead of specifying the segment length, we could specify the degree of information reduction required. This solution would be also helpful for question 2, but recognition would be scale dependent. If we restrict ourselves to the case of equal length for prototype and test sample, scaling would not be a problem. In that case the cost function should allow match with relatively low cost many symbols (angles) of one object with one or few of the other, producing thus a pumping

effect.

Future work should also study the polygon approximation algorithm. One important aspect that has to be improved is the execution time it takes to get the result. Significant reductions of time could be obtained if the optimization were parallelized.

The energy method approach to the solution of the PA problem represents the main contribution of this thesis. Even though some of the experimental results seem discouraging, we believe that the method is valuable in its own right because it introduces a new paradigm in the field of PA and it is therefore the first method attempting to solve the problem.

Bibliography

Bibliography

- [1] James George Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-8(1):67-75, January 1986.
- [2] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.
- [3] Jens Gregor. *Aspects of Data-Driven Inference and Dynamic Programming analysis of Pattern Structure in Strings*. Ph.D. dissertation, Laboratory of Image Analysis, Institute of Electronic Systems, University of Aalborg, Denmark, June 1991.
- [4] Jens Gregor and Michael G. Thomason. Dynamic programming alignment of sequences representing cyclic patterns. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-15(2):129-135, February 1993.
- [5] The Khoros Group. *Khoros Programmer's Manual*. University of New Mexico, 1991.
- [6] The Khoros Group. *Khoros User's Manual*. University of New Mexico, 1991.
- [7] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*, volume I. Addison Wesley, 1992.

- [8] Yoshisuke Kurozumi and Wayne A. Davis. Polygonal approximation by the minimax method. *Computer Graphics and Image Processing*, 19:248–264, 1982.
- [9] U. Montanari. A note on minimal length polygonal approximation to a digitized contour. *Communications of the ACM*, 13:41–54, January 1970.
- [10] T. Pavlidis and S.L. Horowitz. Segmentation of plane curves. *IEEE Transaction on Computers*, C-23(23):860–870, August 1974.
- [11] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.
- [12] Bimal Kumar Ray and Kumar S. Ray. Determination of optimal polygon from digital curve using L_1 norm. *Pattern Recognition*, 26(4):505–509, 1993.
- [13] Jack Sklansky, Robert L. Chazin, and Bruce J. Hansen. Minimum-perimeter polygons of digitized silhouettes. *IEEE Transaction on Computers*, C-21(3):260–268, March 1972.
- [14] Jack Sklansky and Victor Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331, 1980.
- [15] H. Stone. Approximation of curves by line segments. *Math. Comput.*, 15(3):40–47, 1961.
- [16] Yih-Tay Tsay and Wen-Hsiang Tsai. Model-guided attribute string matching by split-and-merge for shape recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 3(2):159–179, 1989.
- [17] R. A. Wagner and J. M. Fischer. The string-to-string correction problem. *Journal of The Association for Computing Machinery*, 21:168–173, 1974.

- [18] Karin Wall and Per-Erik Danielsson. A fast sequential method for polygonal approximation of digitized curves. *Computer Vision, Graphics, and Image Processing*, 28:220–227, 1984.
- [19] C.M. Williams. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing*, 8:286–293, 1978.

Appendices

Appendix A

Khoros

A.1 Introduction

The Khoros system is a comprehensive image/information processing tool for research and software development in Image Processing and Scientific Visualization. It integrates user interfaces, code generators and a big number of image processing and data visualization routines.

Khoros is an open software package developed by the *Khoros Group* at the University of New Mexico. All the source code is distributed free upon request and the user can easily develop his own libraries and integrate them within the software. The software is written in C and the graphical components are based on the X Windows network-transparent window system.

The principal components of Khoros are:

- **preview, composer**
- User Interface Specification (UIS)
- Program Specification (PS)

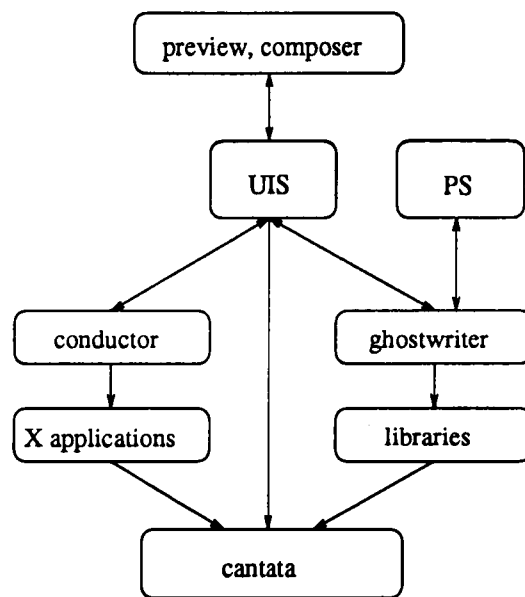


Figure A.1: Khoros architecture

- **conductor**
- **ghostwriter**
- **X applications**
- **data processing libraries**
- **cantata** visual language

A diagram displaying the inter-relationship among these components is shown in Figure A.1. For more information about this software the reader is referred to [6] and [5].

The visual programming language **cantata** is perhaps the most powerful component of the system. From it the user can access the libraries and visually program applications. The “programming” format consists of functional boxes representing specific tasks and data flow representing relations among the compo-

nents. One of the restrictions of this format is that the information the components interchange must be done only through files.

The system provides some X applications for image and data visualization. The user can inspect and modify an existing image with **editimage**. **putimage** only displays an image. Data plotting in 2D and 3D is provided through **xprism2** and **xprism3** respectively. **animate** displays a sequence of images with single bands or a single image with several bands.

Khoros provides a set of tools for creating and maintaining programs. However, the developer must know what tool to use for which applications. This is because Khoros makes distinction between a data processing routine, called a *vroutine*, and an X Window application called an *xvroutine*. A *vroutine* is a program that has no graphical component, that is it does not interact with X Windows. For instance, to make a binary operation between two images, we do not need to display them. The resulting image can be displayed by a *xvroutine* like **editimage**. This means that a *vroutine* is a program that need to be linked against the X libraries. **composer** helps the programmer to develop *vroutines* and **conductor** to develop *xvroutines*.

Every routine in khoros has a UIS file associated. In the UIS file the programmer specifies all the elements for the routine's interface: Input/Output files, command line options, default parameters, error checking, etc. Some "appearance" parameters can also be defined. The other file associated with every routine in khoros is the Program Specification (PS) file, where the source code giving functionality to the program is. These files are used by **ghostwriter** to generate code and man pages for the application program.

There are many other tools that will not be mentioned here. For more information the reader is referred to [6]

A.2 Programming in Khoros

As it was said above one needs to know whether it will have a Graphical User Interface (GUI) or not before coding a program. By a GUI we means any program that interacts with X Windows libraries. Although every single program in Khoros has a user interface that is realized through a window, it is mainly intended to input parameters. And what is more important, the program has no control over it. For example, **vfft** is a routine to compute the two dimensional Fourier transform. The program takes an input image and output the result in a file, but it does not display neither the input nor the output image. To see the resulting image we should connect the output from **vfft** to the input of **editimage**. When a routine provides a GUI but is non-interactive it is called *graphic hybrid* routine.

It is very important to know what type of program one wants to write because:

1. They have to be located in different sections of the Khoros source tree when installing them. *vrotines* can be automatically installed using **kinstall**, but *xvroutines* must be installed by hand.
2. They require a slightly different procedure to use the utilities available to aid the programmer in writing the source code and designing a form for the graphical user interface.

In the next section, we will describe two important elements in Khoros to program a *vroutine*: the User Specification file and the Program Specification file. It has been found rather complicated the respective procedure for a *xvroutine* and therefore will not be explained in this appendix.

A.2.0.1 The User Specification file (UIS)

An UIS file is an ASCII file that contains information necessary to generate both a command line user interface and a graphical form/menu interface. A *vroutine* has to have some interaction with its environment, like input parameters, options, output files, error checking, help files, etc. All this information is specified in the UIS file. It also contains information about the physical distribution of the different items in the window. A complete description for creating UIS files can be found in Chapter 2 of Volume II of Khoros User's Manuals.

The creation of a UIS is difficult and instead of describing how to write it we show the actual UIS file for **vadd**.

```
-F 4.2 1 0 170x7+10+20 +35+1 'Visual Programming Environment' cantata
-M 1 0 100x40+10+20 +23+1 'Binary Arithmetic Operations ' binary
-P 1 0 80x38+22+2 +12+0 'Add Two Images Pixel by Pixel' vadd
-I 1 0 0 1 0 1 50x1+1+2 +0+0 ' ' 'Input Image #1' 'input image 1' i1
-I 1 0 0 1 0 1 50x1+1+3 +0+0 ' ' 'Input Image #2' 'input image 2' i2
-I 1 0 1 0 0 1 50x1+1+5 +0+0 ' ' 'Masking Image' 'add masking image' m
-O 1 0 0 1 0 1 50x1+1+8 +0+0 ' ' 'Output Image ' 'output image' o
-R 1 0 1 13x2+1+11 'Execute' 'execute operation' vadd
-H 1 13x2+38+11 'Help' 'man page ' KHOROS_HOME/doc/manpages/vadd.1
-E
-E
-E
```

Figure A.2 shows how it looks when called from **cantata**.

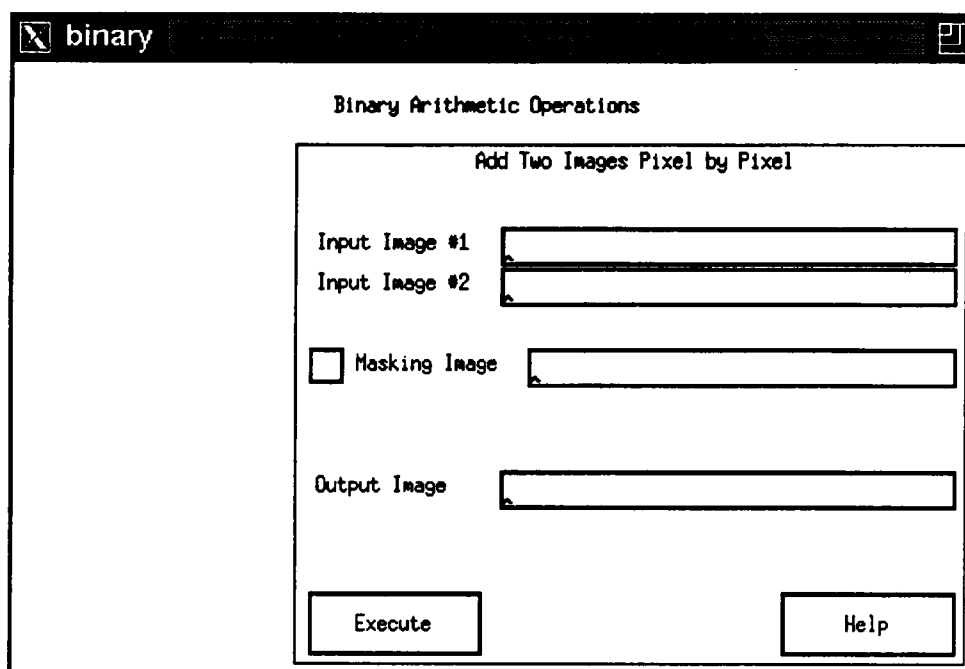


Figure A.2: The user interface window for `vadd`

A.2.0.2 The Program Specification file (PS)

A PS file is an ASCII file that contains code segments (C or FORTRAN code) that gives functionality to the program and documentation necessary to generate 100% of a *vroutine* program. The file is organized in different sections and each section is defined by a special begin and end key. As an example we show three blocks specifying author, purpose and the C code for calling the program `vadd`:

-authors

Danielle Argiro

-authors_end

-short_prog_description

{\bf vadd} computes the arithmetic sum between two images.

-short_prog_description_end

-main_library_call

```
if (! vadd(image1,image2,image3) ) {  
  fprintf(stderr,"Error found in vadd.\n");  
  exit( 1 );  
}
```

-main_library_call_end

There are two utilities for helping programmers to create a UIS file: **composer** and **preview**. **preview** is intended for advanced users (it provides less guidance) and it only works with UIS files. **composer** helps beginners to create both the UIS file and the (PS) file for a *vroutine*. It also provides a graphical interface to *ghost routines*. Chapter 4 Volume II describes how to use **composer** to

A.2.0.3 Ghost Routines

The next step in writing a new routine is to use the *ghost* tools: **ghostwriter**, **ghostreader**, and **ghostcheck**. These are a set of utilities (used with ALL Khoros programs) which are used to facilitate the writing of programs in the Khoros environment. The programmer is advised to always use these routines, because they are helpful in maintaining consistency in the code.

ghostwriter takes the UIS and PS files to create a set of files which will be used to install the final version of the new routine. The files generated by **ghostwriter** have the following extensions: ***.c**, **l*.c**, ***.1**, **l*.3**, and ***.h**.

***.c** is the *driver program* which calls the routine that actually will do the job. This file contains the main program. **l*.c** is called a *library routine* and is which actually do the job. ***.1** contains description of the program for man pages (1) accessible via **vman**. ***.3** contains detailed description of the library routine for man pages (3) accessible via **vman**.

Note that in order to use **ghostwriter**, a valid UIS file must exist in the current working directory. **ghostwriter** will look for a valid PS file in the current working directory; if one does not exist, a template PS file ("***.prog**") will be created. **ghostwriter** will also look for a configuration file *filename.conf*. This file contains the paths where all the files associated with the new routine will be installed. For instance, here is **vadd.conf**.

```
cfile:      $KHOROS_CONTRIB/src/vipl/arith_binary/vadd
hfile:      $KHOROS_CONTRIB/src/vipl/arith_binary/vadd
lfile:      $KHOROS_CONTRIB/src/vipl/Lib
man1file:   $KHOROS_CONTRIB/man/man1
man3file:   $KHOROS_CONTRIB/man/man3
progfile:   $KHOROS_CONTRIB/src/vipl/arith_binary/vadd
panefile:   $KHOROS_CONTRIB/repos/cantata/subforms/arith_binary
helpfile:   $KHOROS_CONTRIB/doc/manpages
subhelpfile: $KHOROS_CONTRIB/doc/cantata/subforms/arith_binary
topsrc:     $KHOROS_CONTRIB/src
```

If a ***.conf** file does not exist in the current working directory a template will be created. Usually this template specifies all the paths pointing to the current

working directory.

If it is the first time you use ghost routines with the new program, you have to run **ghostwriter** first. When modifications to any of the files ***.c** **l*.c**, ***.l**, **l*.3**, and ***.h** are made, **ghostreader** must be executed. Running **ghostreader** insures that all modifications are correctly updated in the **"*.prog"** PS file; thus when **ghostwriter** is run again the updates will be written to the appropriate files.

ghostcheck checks the dates on C source code and man pages versus the dates on ***.prog** and ***.pane** files for the application program that was generated using **ghostreader** and **ghostwriter**.

A.2.0.4 Installing the *vroutine*

Installing a new *vroutine* consists in compiling and linking the source program with existing libraries. It also consists in moving all the associated files to the appropriate places specified in the configuration file ***.conf**. It is strongly advised that all new routines and libraries should be installed in a toolbox instead of the source tree. This Toolbox should mimic the actual structure of **KHOROS_HOME** tree. For a detailed information of the **KHOROS_HOME** source tree see Chapter 9 Configuration Management.

Appendix B

Nonlinear programming

The general nonlinear programming problem is to calculate a vector of variables X that minimizes an objective function $F(X)$, subject to constraints of the form

$$c_i(X) = 0 \quad i = 1, 2, \dots, m' \quad (\text{B.1})$$

$$c_i(X) \geq 0 \quad i = m' + 1, \dots, m \quad (\text{B.2})$$

where the objective function and the constraint functions are assumed to be smooth, i.e. at least twice-continuously differentiable.

Most of the algorithms for solving this problem are iterative. Given a starting vector X_0 , they generate a sequence of points $\{X_k; k = 0, 1, \dots\}$ that is intended to converge to the required solution X^* .

A popular approach used for obtaining the sequence $\{X_k\}$ is by solving a quadratic programming (QP) subproblem. The basic structure involves major and minor iterations. At each major iteration, the new iterate X_{k+1} is determined by

$$X_{k+1} = X_k + \alpha_k d_k \quad (\text{B.3})$$

where X_k is the current iterate, α_k is the step length, and d_k is the search direction.

The search direction d_k is the solution that minimizes the quadratic function

$$Q(d) = \frac{1}{2} d^T B_k d + d^T \nabla F(X_k) \quad (\text{B.4})$$

subject to the linear constraints

$$c_i(X_k) + d^T \nabla c_i(X_k) = 0 \quad i = 1, \dots, m' \quad (\text{B.5})$$

$$c_i(X_k) + d^T \nabla c_i(X_k) \geq 0 \quad i = m' + 1, \dots, m \quad (\text{B.6})$$

B_k is a positive definite quasi-Newton approximation to the Hessian matrix of the Lagrangian function

$$L(X, \lambda, s) = F(X) - \sum \lambda_i (c_i(X) - s_i) + \frac{1}{2} \sum \rho_i (c_i(X) - s_i)^2 \quad (\text{B.7})$$

After computing the search direction d_k , each major iteration proceeds by determining a step length α_k in B.3 such that the new iterate X_{k+1} produces a sufficient decrease in the augmented Lagrangian merit function. This linear search corresponds to the minor iterations. The vector λ is an estimate of the optimal Lagrange multipliers of B.2. s is a vector of non-negative slack variables, and ρ is a vector of penalty parameters controlling the degree of penalizing the Lagrangian function when leaving the feasible region.

Vita

Fernando Rodrigo Rannou was born in Santiago, Chile, on July 11, 1965. He attended Colegio San Marcos from 1970 to 1981, where he recieved the Licenciatura de Enseñanza Media. In March 1982, He entered the Engineering School at Universidad de Santiago de Chile. In the fifth year at USACH, he received the degree of Licenciado en Ciencias de la Ingeniería, and in March of 1988 he received the Ingeniero Civil en Informática degree. The following Fall He joined the Departamento de Ingeniería en Informática as an instructor.

In January 1992, He entered the Graduate School at The University of Tennessee, Knoxville. He completed the requirements for the Master of Science degree with a major in Computer Science, in December 1993.

He is married to Jenny-Ann Marisol Morales Morrison.