Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author’s right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.
http://researchspace.auckland.ac.nz/feedback

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form.
FAST 2D SHAPE APPROXIMATION
Algorithms And Their Errors

Dipl.-Ing. W E Friedrich

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering
University of Auckland,
New Zealand, 1989.
ABSTRACT

In this thesis, efficient algorithms for fast shape approximation in the area of machine and robot vision are analysed. The core of the thesis consists of two main parts, an error analysis of chain coded silhouettes and a simplified, hence fast shape approximation scheme for practical applications.

The area of shape description and approximation is introduced. General techniques for processing of boundary data are outlined followed by a brief description of the image processing system which was developed during the study. Processing procedures developed and their options are presented. Two applications realized by the author have emerged from the algorithm development, firstly a semi-automatic shape registration and measurement procedure and secondly, a low-cost robot vision system. The advantage of the experimental robot vision system is an effective communication between the robot and the object recognition system.

The shortcomings of these applications, mainly inaccuracy of the shape description lead to the introduction of an error analysis for boundary descriptors. In contrast to existing methods, the error analysis can be applied to straight lines, circular arcs, and arbitrary shapes consisting of these two shape primitives. The analysis is comprehensive and covers all kinds of chain code sets representing various pixel shapes as well as different pixel configurations.

The results of the analysis lead to the development of simple ways of overcoming the accuracy limitations of conventional methods. Two algorithms for improving the length estimation of object outlines are introduced. A comparison with existing methods showing the effectiveness of these algorithms is made.

For practical applications of object recognition, a new, simple and hence fast and effective algorithm for polygon approximation, called the "arc operator" is developed. The combination of the arc operator and the length correcting algorithm directly improves the ability to identify partially visible shapes. The performance of the arc operator is compared to six other well known algorithms. Slight disadvantages in accuracy are outweighed by the enormous advantage in processing time, particularly with an integer implementation. An example of overlapping parts demonstrates the potential of the arc operator to extract characteristic shape descriptions.
ACKNOWLEDGEMENTS

I am indebted to the Department of Scientific and Industrial Research (DSIR) for providing the funding for this study. I would also like to thank the following people who have assisted me during the course of this study.

Associate Professor Gunter Arndt and Dr. Des Tedford of the Mechanical Engineering Department for their part as supervisors, their advice and encouragement.

Wayne Power for proof reading,

Mike Riley for expert advice on page layout, and last but not least

Denis Malone and John de Pont for helpful discussions.

I am very grateful to my wife Ute for her understanding and moral support.

To her and our son Stefen I dedicate this thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>ix</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1.1</td>
</tr>
<tr>
<td>2. SHAPE DESCRIPTION ALGORITHMS</td>
<td></td>
</tr>
<tr>
<td>2.1 Basics of Machine Vision</td>
<td>2.1</td>
</tr>
<tr>
<td>2.1.1 Processing stages of machine vision</td>
<td>2.1</td>
</tr>
<tr>
<td>2.1.2 Relation to adjacent fields</td>
<td>2.3</td>
</tr>
<tr>
<td>2.2 Digital Image Analysis</td>
<td>2.4</td>
</tr>
<tr>
<td>2.2.1 Segmentation</td>
<td>2.4</td>
</tr>
<tr>
<td>2.2.2 Shape description</td>
<td>2.5</td>
</tr>
<tr>
<td>2.2.3 Global parameters</td>
<td>2.5</td>
</tr>
<tr>
<td>2.2.4 Local descriptors</td>
<td>2.6</td>
</tr>
<tr>
<td>2.2.5 Chain codes and Crack codes</td>
<td>2.7</td>
</tr>
<tr>
<td>2.2.6 Object recognition</td>
<td>2.8</td>
</tr>
<tr>
<td>2.3 Industrial Vision Systems</td>
<td>2.9</td>
</tr>
<tr>
<td>2.4 Laboratory Equipment &amp; Development</td>
<td>2.10</td>
</tr>
<tr>
<td>2.4.1 Equipment</td>
<td>2.11</td>
</tr>
<tr>
<td>2.4.2 Kiwi fruit measurement</td>
<td>2.13</td>
</tr>
<tr>
<td>2.4.3 Robot vision</td>
<td>2.15</td>
</tr>
<tr>
<td>2.4.4 Shortcomings of the two systems</td>
<td>2.19</td>
</tr>
<tr>
<td>2.5 Shape of picture elements</td>
<td>2.19</td>
</tr>
<tr>
<td>2.5.1 Quantization</td>
<td>2.19</td>
</tr>
<tr>
<td>2.5.2 Square pixels</td>
<td>2.20</td>
</tr>
<tr>
<td>2.5.3 Rectangular pixels</td>
<td>2.20</td>
</tr>
<tr>
<td>2.5.4 Hexagonal pixels</td>
<td>2.20</td>
</tr>
<tr>
<td>2.6 Algorithms to Improve Chain Code Descriptions</td>
<td>2.21</td>
</tr>
<tr>
<td>2.7 Shape Approximation</td>
<td>2.22</td>
</tr>
<tr>
<td>2.8 Discussion</td>
<td>2.24</td>
</tr>
</tbody>
</table>
3. THEORETICAL ERROR ANALYSIS FOR CHAIN CODED STRAIGHT LINES

3.1 Introduction
   3.1.1 Pixel shapes and configurations
   3.1.2 Chain Code sets for different pixel configurations

3.2 Estimating The Length of Digitized Straight Lines
   3.2.1 Crack code errors
   3.2.2 Chain code errors
   3.2.3 Hexagonal code errors

3.3 The Fundamental Error Function
   3.3.1 Slope ranges for square and rectangular pixels
   3.3.2 Slope ranges for hexagonal pixels

3.4 Maximum and Average Magnitudes of Length Deviation
   3.4.1 Square pixel error function
   3.4.2 Rectangular pixel error function
   3.4.3 Hexagonal pixel error function
   3.4.4 Extended code sets

3.5 The Matrix Form of The Error Function

3.6 Conclusion: Error Analysis

4. PERIMETER ESTIMATION ERROR OF CHAIN CODED SILHOUETTES

4.1 Introduction
4.2 Principle of Length Deviation Using the Error Function
4.3 The Correlation of the Error Function
4.4 Examples of Vector Representations for Simple Shapes
   4.4.1 Regular polygons
   4.4.2 Arbitrary polygons
   4.4.3 Circular arcs
   4.4.4 Combined circular arcs and straight line segments
4.5 Examples of Specific Error Profiles
4.6 Conclusion

5. SILHOUETTE APPROXIMATION BY CHAIN CODES

5.1 Introduction
5.2 Description of Silhouette Coding Methods
   5.2.1 Crack code methods
   5.2.2 Chain code methods
5.3 Experiments with Simple Shapes
   5.3.1 Silhouette generation
   5.3.2 Code Generation
      5.3.2.1 Chain code generation
      5.3.2.2 Crack code generation
   5.3.3 Experiments with 4 chain coded shapes
      5.3.3.1 Circles
      5.3.3.2 Semi-circles
      5.3.3.3 Squares and triangles
5.4 Conclusion
6. THE ARC OPERATOR

6.1 Introduction
6.2 A New Approach to Shape Analysis
   6.2.1 Definition of the algorithm
   6.2.2 Distance function
   6.2.3 Running distance total
6.3 Parameter Selection
   6.3.1 Segment length
   6.3.2 Perpendicular distance
   6.3.3 Integrated distance
   6.3.4 Segment deviation
6.4 Implementation Considerations
6.5 Experimental Results
6.6 The Slope Angle Domain Application
   6.6.1 Feature recognition
   6.6.2 Slope angle approximation
   6.6.3 Overlapping objects
6.7 Concluding Remarks

7. CONCLUSION AND FUTURE WORK

8. REFERENCES

APPENDIX

Overview of appendix organization

A Error Function - Coefficients and Matrices.
B Catalogue of Error Function Profiles
C Software Library - Implemented Image Processing Functions
D Robot Sensor Communication and Protocol
E Publications

Vision-Aided Flexible Component Handling, Robhanz, Nov 1986
also in Automation and Control (NZ), 17(10):16-21, 1986.

A New Silhouette-Related Algorithm for Improving Vision-Aided

Integrated Robot Vision System for Flexible Component Handling
SPIE Conference on Visual Communications and Image Processing '89,
Philadelphia, USA.
List of Figures

| Figure 2-1: | Image analysis by machine vision | 2.2 |
| Figure 2-2: | Relation to adjacent fields based on Pavlidis. | 2.3 |
| Figure 2-3: | Chain code and Crack code definition. | 2.7 |
| Figure 2-4: | Hierarchy of program library and structure. | 2.13 |
| Figure 2-5: | Kiwifruit work station | 2.14 |
| Figure 2-6: | Average Kiwifruit shape | 2.15 |
| Figure 2-7: | Robot Vision set-up at Auckland University | 2.16 |
| Figure 2-8: | Rapid object position test | 2.17 |
| Figure 2-9: | Schematics of the robot vision system | 2.18 |

| Figure 3-1: | Pixel shapes and configurations | 3.2 |
| Figure 3-2: | Chain code sets for k = 1, 2 and 3 | 3.3 |
| Figure 3-3: | Code sets for hexagonal configurations | 3.3 |
| Figure 3-4a: | Error definition for the crack codes | 3.4 |
| Figure 3-4b: | Error function for a 4 direction code | 3.4 |
| Figure 3-5a: | Error definition for the 8 direction code | 3.6 |
| Figure 3-5b: | Error function for an 8 direction code | 3.7 |
| Figure 3-6: | Error function for hexagonal pixels (k = 1) | 3.8 |
| Figure 3-7: | Approximation triangle | 3.8 |
| Figure 3-8a: | Error function for rectangular pixels | 3.13 |
| Figure 3-8b: | Error function for digitized straight lines | 3.14 |
| Figure 3-9: | Average error for digitized straight lines | 3.16 |
| Figure 3-10: | Error function for 16 direction code | 3.16 |
| Figure 3-11: | Error function for 24 direction code | 3.17 |
| Figure 3-12: | Error function for hexagonal pixels | 3.18 |
| Figure 3-13a: | Error maxima for 72 direction codes | 3.18 |
| Figure 3-13b: | Error maxima for triplet and hexagonal pixels | 3.19 |
| Figure 3-14: | Angle intervals for k = 9 and k = 12 | 3.19 |
| Figure 3-15: | Average errors for k = 1 (8 dir) to k = 100 (800 dir) | 3.20 |
| Figure 3-16: | Flow chart for error matrix values | 3.23 |
| Figure 3-17: | Matrix values compared to error function | 3.24 |

<p>| Figure 4-1: | Deviation error of a square | 4.2 |
| Figure 4-2: | Deviation error of a rectangle | 4.3 |
| Figure 4-3: | Error values of a square at 60 degrees | 4.4 |
| Figure 4-4: | Error profile of a square (rect. pixels) | 4.4 |
| Figure 4-5: | Vector diagram of a regular triangle | 4.6 |
| Figure 4-6: | Vector diagram of a rectangle (ratio 2:1) | 4.7 |
| Figure 4-7: | Error profile of a semi-circle | 4.9 |
| Figure 4-8: | Error profile of a triangle | 4.10 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>Arc approximation with various code sets</td>
<td>5.2</td>
</tr>
<tr>
<td>5-2</td>
<td>Error function for a corrected square</td>
<td>5.3</td>
</tr>
<tr>
<td>5-3</td>
<td>Silhouette approximation by Dunkelberger</td>
<td>5.4</td>
</tr>
<tr>
<td>5-4</td>
<td>Silhouette approximation by Friedrich</td>
<td>5.5</td>
</tr>
<tr>
<td>5-5</td>
<td>Fish-bone pattern for code detection</td>
<td>5.9</td>
</tr>
<tr>
<td>5-6</td>
<td>Error comparison for chain coded circles</td>
<td>5.12</td>
</tr>
<tr>
<td>5-7</td>
<td>Error comparison for crack coded circles</td>
<td>5.12</td>
</tr>
<tr>
<td>5-8</td>
<td>Error comparison for crack coded semi circles</td>
<td>5.13</td>
</tr>
<tr>
<td>5-9</td>
<td>Error comparison for chain coded semi circles</td>
<td>5.14</td>
</tr>
<tr>
<td>5-10</td>
<td>Error comparison for crack coded triangles</td>
<td>5.15</td>
</tr>
<tr>
<td>5-11</td>
<td>Error comparison for chain coded triangles</td>
<td>5.16</td>
</tr>
<tr>
<td>5-12</td>
<td>Error comparison for crack coded squares</td>
<td>5.16</td>
</tr>
<tr>
<td>5-13</td>
<td>Error comparison for chain coded squares</td>
<td>5.17</td>
</tr>
<tr>
<td>5-14</td>
<td>Results of methods examined</td>
<td>5.18</td>
</tr>
<tr>
<td>6-1</td>
<td>Principle of the arc operator</td>
<td>6.3</td>
</tr>
<tr>
<td>6-2</td>
<td>Corner detection with the arc operator</td>
<td>6.5</td>
</tr>
<tr>
<td>6-3</td>
<td>Reconstructed outline of metal stamping</td>
<td>6.6</td>
</tr>
<tr>
<td>6-4</td>
<td>Curve detection using a Kiwi fruit silhouette</td>
<td>6.7</td>
</tr>
<tr>
<td>6-5</td>
<td>Polygon approximation of a Kiwi fruit.</td>
<td>6.7</td>
</tr>
<tr>
<td>6-6</td>
<td>Influence of a large region of support.</td>
<td>6.8</td>
</tr>
<tr>
<td>6-7</td>
<td>Distance threshold only</td>
<td>6.9</td>
</tr>
<tr>
<td>6-8</td>
<td>Integrated distance threshold</td>
<td>6.10</td>
</tr>
<tr>
<td>6-9</td>
<td>Segment distance.</td>
<td>6.11</td>
</tr>
<tr>
<td>6-10</td>
<td>Polygon approximation for a seat belt buckle</td>
<td>6.11</td>
</tr>
<tr>
<td>6-11</td>
<td>Simplified distance measure</td>
<td>6.12</td>
</tr>
<tr>
<td>6-12</td>
<td>Possible hardware implementation of the Arc operator.</td>
<td>6.13</td>
</tr>
<tr>
<td>6-13</td>
<td>Arc operator functions on a sharp corner sample.</td>
<td>6.14</td>
</tr>
<tr>
<td>6-14</td>
<td>Competitive break-point reporting</td>
<td>6.15</td>
</tr>
<tr>
<td>6-15</td>
<td>Split and merge method [Pavlidis 1982]</td>
<td>6.16</td>
</tr>
<tr>
<td>6-16</td>
<td>Arc operator with floating point calculations.</td>
<td>6.17</td>
</tr>
<tr>
<td>6-17</td>
<td>Arc operator with integer calculations.</td>
<td>6.17</td>
</tr>
<tr>
<td>6-18</td>
<td>Split and merge approximation (circular stamping)</td>
<td>6.19</td>
</tr>
<tr>
<td>6-19</td>
<td>Arc operator approximation (circular stamping)</td>
<td>6.19</td>
</tr>
<tr>
<td>6-20</td>
<td>Split and merge approximation (screw terminal)</td>
<td>6.20</td>
</tr>
<tr>
<td>6-21</td>
<td>Arc operator scheme applied to a screw terminal</td>
<td>6.20</td>
</tr>
<tr>
<td>6-22</td>
<td>Split and merge approximation (chuck key)</td>
<td>6.21</td>
</tr>
<tr>
<td>6-23</td>
<td>Arc operator approximation (chuck key)</td>
<td>6.21</td>
</tr>
<tr>
<td>6-24</td>
<td>Slope angle diagram of the switch cover - straight outlines</td>
<td>6.24</td>
</tr>
<tr>
<td>6-26</td>
<td>Slope angle representation of seat belt buckle</td>
<td>6.24</td>
</tr>
<tr>
<td>6-27</td>
<td>Approximating the slope angle profile (semi-circular shape)</td>
<td>6.26</td>
</tr>
<tr>
<td>6-28</td>
<td>Approximating the slope angle profile (circular stamping)</td>
<td>6.26</td>
</tr>
<tr>
<td>6-29</td>
<td>Overlapping objects</td>
<td>6.27</td>
</tr>
<tr>
<td>6-30</td>
<td>Slope diagram of overlapping objects</td>
<td>6.28</td>
</tr>
<tr>
<td>6-31</td>
<td>Slope diagram of circular stamping (conventional approx)</td>
<td>6.28</td>
</tr>
<tr>
<td>6-32</td>
<td>Approximated slope angle diagram of chuck key</td>
<td>6.29</td>
</tr>
<tr>
<td>6-33</td>
<td>Approximated slope angle diagram of screw terminal.</td>
<td>6.29</td>
</tr>
<tr>
<td>6-34</td>
<td>Matching trial in the slope angle domain.</td>
<td>6.30</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3-1:</td>
<td>Length estimation errors for chain codes.</td>
<td>3.15</td>
</tr>
<tr>
<td>Table 3-2:</td>
<td>Error matrix for short digitized straight lines</td>
<td>3.24</td>
</tr>
<tr>
<td>Table 5-1:</td>
<td>Code combinations and their length replacement factors</td>
<td>5.6</td>
</tr>
<tr>
<td>Table 5-2:</td>
<td>Code approximation replacement table [Dessimoz]</td>
<td>5.7</td>
</tr>
<tr>
<td>Table 5-3:</td>
<td>Organization of a lookup table for length adjustments</td>
<td>5.7</td>
</tr>
<tr>
<td>Table 5-4:</td>
<td>8 direction chain code from position differences</td>
<td>5.10</td>
</tr>
<tr>
<td>Table 5-5:</td>
<td>Chain code conversion</td>
<td>5.11</td>
</tr>
<tr>
<td>Table 6-1:</td>
<td>Performance comparison of the arc operator with various other algorithms from [Teh 1988].</td>
<td>6.18</td>
</tr>
<tr>
<td>Table 6-2:</td>
<td>Performance comparison of the arc operator with the split and merge algorithm [Pavlidis 1982].</td>
<td>6.22</td>
</tr>
</tbody>
</table>
List of Symbols

A, B and C = Error function coefficients
a = half the side length of regular geometric shapes considered
d = arc operator distance and its x and y components (index)
d_s = boundary element length
d_x = position difference in x direction
d_y = position difference in y direction
Err = error function (length deviation error)
(index indicates code set, e.g. Err4 for 4 direction code
   v for vertical
d for diagonal
h for horizontal
ave = average
tot = total
a, b, c, d, e, f, g, h, i, k, l, p, q, r, s =
   = are used as parameters to calculate the error matrix
f_c = Digitization / conversion frequency
f_s = image frame rate per second
i = pixel index
k = pixel neighbourhood size, and arc length
k = arc operator length
L_e = estimated length of a digitized line
l_f = lines per frame
l_i = individual length of code element
L_t = true length
m = number of equal sides of a polygon
n_i = frequency of code element
r_c = aspect ratio of camera
r = pixel aspect ratio
α = slope angle of straight line or shape segment
β = orientation of shape
φ = code set angle
v_p = vector descriptor
x(i), y(i) = x and y coordinates of the border pixel at the \(i^{th}\) location in the code set or boundary.
Shape description is an essential part of machine vision which is a general term describing acquisition, processing and interpretation of images by machine. As a part of the interpretation stage, shape description is a critical component in the process of analysing the image. This work is primarily concerned with fast shape description algorithms.

Processing of images by computer was first applied in areas such as quantitative microscopy, medical problems and recently by remote sensing, an area of aircraft and satellite imaging. Industrial applications received less attention until more recently. The increasing performance of computers, the development of special-purpose hardware and the availability of suitable algorithms from diverse application areas, sparked the development of machine vision systems for industrial applications more than a decade ago. Today there are numerous commercial vision systems available.

Common to many of the applications is shape description in one form or another. For instance, in medical applications, particularly in the area of cancer research shape description is used to find cell anomalies. In remote sensing, shape description is applied to condense the information of land form with a high demand on accuracy. Industrial applications require boundary related measurements for inspection, recognition or tracking purposes.

The earlier applications of machine vision were not time critical. In most cases the image analysis had to be performed some time after the picture was taken as, for example, in remote sensing or in many medical applications. Commonly, these procedures are interactive with an operator judging the result. If necessary, the analysis could be repeated for example, with another set of parameters to find the optimum result.

This is in contrast to industrial applications where the analysis is final with no interaction by an operator. The results are immediately used to control for example, production machinery. Two important areas of industrial applications have been established: inspection, also referred to as automated visual inspection and adaptive control of robots by means of robot vision. These two categories have fundamentally different objectives.

In automated visual inspection, the objective is to use vision to check component parameters for example, object dimensions, verify completeness or uniformity. While in robot vision, vision is used to provide sensory feedback to guide the robot.
A robot vision system with a stationary camera may be used in a component handling application to recognise the object by means of shape description and direct the robot in terms of object category, position and orientation. This configuration presents an open loop control system with no feedback. A robot vision system can also be used in a closed loop manner to provide feedback information as part of the servo system. For instance in welding, where a camera may be mounted on the arm of the robot to guide the motion directly. In this example, the camera is moving with the robot, a so-called "eye-in-hand" application, and images are processed in an even faster sequence in order to steer the welding process along the desired track.

In robot vision, adaptive control structures are required in order to process this type of sensory information. The performance of industrial robots have been constantly improved over recent years, mainly by more powerful controllers. Robot vision was only possible through the integration of sensory and motion capabilities into the control structure in the late 1970's. Robot vision is therefore a relatively new field derived from other application areas of image processing.

In all these tasks some form of shape description is employed. In automated visual inspection, shape description may be used to extract required measurements. In object recognition, the object silhouettes have to be described in a way to allow fast and reliable shape matching. In the example of weld tracking, a shape description algorithm may be used to describe the parts to be welded or identify the weld pool. The remaining gap can be located and any offset compensated by issuing a correcting motion command.

The high demands in industrial applications necessitates very short processing times in the order of 1 second in many cases even as short as 0.1 sec. This is one reason why most industrial image processing systems for object recognition rely on so-called global features, such as area and perimeter etc. as shape descriptors, which can be processed with special-purpose hardware in a fraction of a second. One major drawback of these systems has been that closely related or incomplete shapes are difficult to distinguish by global features. Further requirements such as separated parts, excellent constant illumination and high contrast, restrict the application areas of industrial machine vision systems even further.

Shape description techniques provide local boundary information which can be used to locate particular features or to identify partially visible shapes. Although algorithms for shape descriptions have been developed for a variety of earlier applications, the processing speed was certainly not as crucial as it is now for industrial applications. Computer processing times in the order of tens of seconds are still common [Teh 88] but completely inadequate for industrial purposes. In order to take advantage of existing algorithms, modifications to increase processing speed have to be made. These modifications may include simplified calculations for example, using approximations and avoiding floating point operations. However, simplifications have to be examined very carefully in order to keep possible accuracy losses to a minimum.

The essential requirements for industrial machine vision might be summarized as follows: fast processing (most important), reliable operation, fundamental simplicity, ease of scene illumination (a vital requirement), and low cost. Some of the rudimentary requirements can be met by a combination of special-purpose hardware and robust, simple and hence fast algorithms.
In all boundary related features such as corners it is very important to locate the measurement along the borderline as accurately as possible. The reason for this is that many features relate geometrically to other unique locations along the boundary. These features are the basis of further processing and data reduction in order to achieve an efficient recognition process.

This work examines principal deviation errors of common chain codes such as [Freeman's 1961], one of the most widely used, and introduces new fast algorithms for boundary approximations, to improve boundary length measurement for accurate feature location and to achieve very fast shape approximations.

Chapter 2 provides an introduction into image analysis and presents an extensive review of existing shape description algorithms. The laboratory set-up established during the course of this work is explained including the image processing system implementation along with two applications, one for fruit shape measurement and registration, the other for a vision-aided component handling system.

In order to find an accurate shape description, a theoretical error analysis of chain coded straight lines is presented in Chapter 3. In contrast to other publications which are briefly discussed in the introduction of Chapter 3, the validity of the error analysis includes all practical pixel shapes such as square, rectangular and hexagonal.

In Chapter 4 the error analysis is extended to arbitrary shapes comprising straight line segments and circular arcs. The perimeter deviation of the shape can be evaluated by applying a correlation between the vector description of the shape and the appropriate error function.

From these results it is obvious that chain code representation is not accurate enough. Algorithms to improve chain code descriptions are examined next. Four existing schemes are compared in Chapter 5 along with two new algorithms, one based on crack codes and the other based on chain codes. Comparison is achieved by using computer generated simple geometric shapes.

To find an acceptable shape approximation for the purpose of recognition, the arc operator - a new and very fast shape approximation scheme - is introduced in Chapter 6. Its performance is then tested both in terms of processing speed and approximation quality. A comparison of the arc operator to a conventional split and merge algorithm is presented using a variety of shapes including a shape consisting of four semi-circles. This particular shape is used as a basis to compare six other polygon approximation schemes. Processing times for the various algorithms and the arc operator are also included for comparison.

Chapter 7 summarizes this work and outlines proposals for future study.

Section 2.4 (Laboratory set up and developments) and Chapters 3 to 6 describe my original work.
CHAPTER TWO

2. SHAPE DESCRIPTION ALGORITHMS

2.1 Basics of Machine Vision

Vision combined with mental processing is the most powerful capability a human has. Information provided by vision sensing enables us to interact with the environment in an intelligent way. Processed visual information is the base for many human decisions necessary to avoid obstacles (guidance), to estimate distances, to recognize and locate objects, and their relationships in order to handle them. Much of these capabilities are required for automatic component handling or contour following and attempts have been made to provide machines with a similar sense of vision.

In fact, robots are working today successfully in factories with controlled environments using vision sensing for inspection, component handling and guidance applications. However, commercial machine vision systems are still an expensive option and offer only limited capabilities, as discussed below.

A machine vision system is based on special-purpose computer hardware which is able to store image information from a camera. The principal task of a machine vision system is to analyze the image data and to produce descriptions of the viewed objects. These descriptions reflect the essential object features in order to accomplish the given task. A typical machine vision system is part of an overall system which interacts with the environment. The vision system provides feedback information to the overall system which is concerned with the control of the task, such as rejection of false parts in automated visual inspection, and handling or track following in robot vision.

The input to a vision system is an image in digitized form, while the output is a description of the object(s) of interest. In effect, this is a data reduction technique where redundant information from the vast amount of picture elements is reduced to a set of shape descriptors.

2.1.1 Processing stages of machine vision

A number of key processing steps have to be performed in machine vision (refer to Figure 2-1):
1) Perception - forming the image
2) Pre-processing - reducing the image data to relevant object data
3) Extraction - of relevant object features
4) Analysing and interpretation of these results - decision generating
5) Interfacing - directing the handling system or operation to perform

![Diagram of Image Analysis by Machine Vision](image)

**Figure 2-1: Image analysis by machine vision**

Each of these steps is typically related to a specific field. The following paragraph reveals important similarities as well as differences of related areas in respect to robotic vision sensing.

1) Forming the image is the first significant stage. The better the image quality the simpler the processing requirements. Illumination techniques and appropriate optics are very important and can determine success or failure. A large number of these techniques are discussed in Batchelor [1985].

2) The pre-processing stage is largely concerned with improving image data, such as enhancing contour lines, removing blurring and reducing noise. These techniques are typical applications in the field of image processing, whereby improved image data are calculated from previous image data. Input and output data are images.
3) Relevant object features and shape parameters are extracted for subsequent analysis. Necessary properties of these features include independence of object orientation, of changes in size and of translational position.

4) Analysing and interpretation is largely the field of pattern recognition. Extracted object measurements are classified and assigned to a known class.

5) Directing machinery comprises decision generating and communication with the handling equipment.

From the above description it can be seen that the stages of processing are hierarchical: in the pre-processing stage an enormous amount of individual pixels are processed by simple point or local area operators. More sophisticated classification algorithms are employed in the final stages on a greatly reduced amount of data. The final result may comprise only a few or even a single parameter.

2.1.2 Relation to adjacent fields

The difference between the three systems (image processing, image recognition and robot vision) is highlighted by their different output requirements and time constraints.

Thus the output of an image processing device is a modified image, the output of a recognition system is typically an object number, whereas the output of a robot vision system is related to a physical action [Gonzalez 1983].

![Diagram](https://via.placeholder.com/150)

**Figure 2-2:** Relation to adjacent fields based on Pavlidis [1982].
The relationship between other important fields is shown in Figure 2-2. Note the opposite nature of object recognition and graphics in the synthesis branch of this diagram. In graphics, an image is generated from a design or object description, while in machine vision, the object is described by parameters based on measurements taken from an image of the object.

Clearly, the highest demands on processing speed are made by robot vision, including automated visual inspection (AVI) [Bowman 1985]. This is because the result of visual processing is used directly by machines and not checked by any operator as in most other fields. Hence, algorithms applied to automation must be very reliable and simple [Pugh 1985].

Ideally, vision sensing (as well as other sensing capabilities) should be an integral part of the actual handling device to facilitate the necessary close links between sensor and motor functions [Taylor 1983].

However, most of the visual sensing devices available on today's market are stand-alone sensors and may also be useful in other applications of industrial automation [Gonzalez et al 1982].

2.2 Digital Image Analysis

In machine vision, various techniques and approaches are categorized. These are basically divided into three subdivisions: low-, medium- and high-level vision [Fu 1987]. Low-level vision is concerned with basic sensing and pre-processing. This category deals mainly with image processing techniques such as contrast enhancement, noise reduction and other types of filtering. Low-level vision is often hardware supported by special-purpose computer architectures. The area of high speed machine vision attracts world-wide attention due to the availability of array processors which are ideally suited for the repetitive pixel-based calculations. The next level up comprises segmentation techniques and boundary descriptions - the area addressed in this work - and object recognition.

Finally, high-level vision introduces concepts of machine intelligence such as the ability to extract relevant information, to learn from examples and apply this knowledge in different situations. Research in this area has been very active and has generated considerable interest [Brady 1981].

2.2.1 Segmentation

In medium level vision one of the most critical tasks is segmentation. This process partitions the image into objects and background for further processing towards such activities as object recognition. Two basic concepts are mainly used in this category namely edge detection and thresholding.

The attention of recent international research has focused particularly on the theory of edge detection and their practical implementation [Marr & Hildreth 1980]. In edge detection, neighbouring picture elements (pixels) are examined in terms of discontinuity of their intensity. The resulting sets of pixels form an incomplete boundary due to noise and illumination effects [Friedrich 1985]. Thus, edge detection algorithms are typically followed by a linking operation to achieve a sensible object border.
Because of its high proportion of iterative calculations - often with floating point operations and therefore extremely long processing times in the order of minutes [Hildreth 1983] - this concept is not ideally suited for the high demands in machine and robot vision.

In contrast to edge detection, thresholding requires only very simple operations - namely the comparison of the intensity of a pixel with a fixed (global) or variable (local or dynamic) threshold [Weszka 1978]. Global thresholding is very sensitive to intensity changes, only high contrast images i.e. created by structured light or backlighting (as used in the laboratory set-up) can be successfully segmented by global thresholds. Higher requirements in terms of uniform lighting often presents a drawback of this concept in certain applications. Local thresholding (local area dependant) and dynamic thresholding (position dependant) are suited for all other cases where lighting is difficult to control and compensations have to be made for the different lighting levels.

Selecting a suitable threshold level can be found by examining the histogram of an image. The occurrence of intensity levels from background and object form two maxima. The minimum occurrence of intensity levels between the two maxima represents an good threshold intensity [Ballard 1982]. If the histogram is taken of the entire image, a global threshold is found. For local thresholds the image is divided into sub-images. The histogram of each of the sub-images are examined in the same way and these local thresholds are then applied to their respective sub-image. A dynamic threshold can be found by interpolating thresholds from both neighbouring sub-images and neighbouring pixels. These results form a threshold image which can be compared to the original image on a point to point basis [Rosenfeld 1982].

**2.2.2 Shape description**

In order to recognise shapes, relevant features have to be identified. The features extracted must be unique to the individual object. A desirable property of these features is that they be independent of object location, size and orientation. Object descriptors are very important as they affect the performance and complexity of recognition algorithms. The two major descriptor types (3D descriptors are not considered here) are based on either global or local features.

**Global Parameters** or Regional Descriptors, comprise statistical parameters or texture descriptors. These global features however require a fully visible object, whereas local parameters are based on boundary description and may be useful in cases where objects are partially visible.

**2.2.3 Global parameters**

Global parameters may be scalar, ranging from simple descriptors such as area, perimeter etc. to more complex descriptors such as Fourier descriptors or moments [Gonzalez and Wintz, 1980]. Other descriptors can be structural using a relational or tree structure, such as concavity trees [Batchelor 1979] where the nodes of the tree represent various levels of concavity within the shape. The medial axis transform is another structural descriptor [Blum 1976] where the object is shrunk until only the structure remains as thin lines. This method is however very noise sensitive and impractical without extensive processing [Pratt 1979].
Many existing industrial vision systems are based on simple statistical measures for computational reasons [Geissler 1982]. Applications are restricted to cases where objects are sufficiently different that they can be distinguished by global parameters. Moreover, objects can only be characterized if they are separated and fully visible. Global parameters were introduced into a complete system more than ten years ago by the Stanford Research Institute [Agin 1975]. These SRI parameters have been successfully applied in numerous industrial vision systems [Hewkin 1981 (BBC), Brune 1983 (Bosch), Doerr 1983 (Automatix)]. Frequently used parameters include area, perimeter, compactness (perimeter²/area), Euler number and shape moments. An Euler number is defined as the number of connected regions minus the number of holes. For example, for an object without a hole the Euler number would be 1, an object with two holes would have an Euler number of -1 etc. Statistical parameters can be calculated within a fraction of a second by hardware implementation and custom LSI (large scale integration) circuits. One particular system, developed in Australia, can extract these SRI parameters in 20 ms - as fast as a TV image can be scanned [Burford 1987].

2.2.4 Local descriptors

Local shape descriptors are parameters which relate directly to particular positions along the boundary. One of the major advantages over global parameters is that partially occluded shapes can be recognised provided unique characterizing features of the boundary are visible. However, these descriptors are generally computationally more expensive but provide, on the other hand, comprehensive information which can be - in contrast to global parameters - independent of size and shape of the object. The most common boundary descriptors are Boundary Functions (or Signatures), Fourier Descriptors, Chain Codes, Run length codes and Polygonal Approximations.

Boundary functions are one-dimensional functions of the boundary in the form of curvature, radial function (the distance between the outline and the centroid as a function of either the angle or the position along the perimeter) [Pratt 1979] or as a histogram of the tangent angle values referred to as the slope density function [Nahin 1974]. These functions are independent of the size of the object but depend on the location of the starting point. An orientation change results in an offset which is a measure of object orientation. Radial functions require a fully visible object in order to calculate the centre of area. Furthermore, this one-dimensional function still requires considerable computation (i.e. convolution) in order to fulfil the task of recognition [Horn 1986].

A Fourier transform of a two dimensional boundary reduces to a one-dimensional form by interpreting the boundary position (x, y) as a complex number (x + jy) [Pratt 1979]. In order to use the advantages of the fast Fourier Transform (FFT), the number of input parameters - here boundary locations - have to be of power 2, which requires a sampling adjustment before the actual processing can take place. The first few coefficients of the transform can be sufficient for determination of reasonably different shapes. Disadvantages in the use of this transform include the need for a fully visible object, extensive computing effort and considerable performance limitations. These drawbacks include the difficulty in describing fine details in the shape and an inverse transform often reproduces a curve which is not closed [Zahn et al 1972].
Run-length codes describe the length of each row or column belonging to the object. These Run-length codes may be generated during scan time with special-purpose hardware. The notation of these codes comprise the current line number and each transition between object and background with a column number. The run-length-code also contains the current shape number or component label allowing several transitions along the scanning line (hole, separate object, etc) [Bretchi 1981]. In hardware implementations run-length coding can be generated while the image is scanned i.e. in 20 ms (50 Hz raster frequency). Segmentation and chain code generation can be derived from run-length coding by comparison of scan data from the previous and current line. Segmentation and Chain codes can be derived from run-length coding by appropriate hardware [Marlow 1981].

Chain codes and polygonal approximations are dealt with extensively throughout this work and therefore reviewed in more detail in the following sections.

2.2.5 Chain codes and Crack codes

Chain codes are a set of predefined vectors to describe the outline of an object. Generally these vectors have a unit length of one pixel and are typically arranged in a regular way using four or eight basic directions. The unit length in 4 and 8 direction codes is therefore determined by the grid spacing i.e. the resolution available which corresponds to the digitization frequency and the frame store size as discussed below. Chain codes were first examined by Freeman [1961] and are a convenient measure and still commonly used in todays problems of image analysis.

![Chain code and crack code definition](image)

**Figure 2-3:** Chain code and Crack code definition.
These codes provide a standard base for further processing of boundaries. Chain codes can also be specified with more than 8 directions with a suitable unit length of more than one pixel in each direction. Accordingly, when using hexagonal shaped pixels, six (or multiples thereof) direction codes are applied (see also Chapter 3).

Crack codes are 4 direction chain codes which can be denoted as follows. In comparison with 8 (or more) direction codes, crack codes follow the "cracks" between the pixels rather than moving from one pixel to another. Figure 2-3 illustrates this in more detail. Pixels are essentially treated as tiny tiles. A boundary description using crack codes effectively encloses the group of object pixels whereas 8 direction codes describe the boundary on a pixel centre to centre basis.

Chain codes and crack codes are generated by tracing routines which track along the boundary once an object pixel has been identified. A large number of tracing algorithms are available. Common contour tracing algorithms move from pixel centre to pixel centre of an eight-connected neighbourhood (Figure 2-3). Horizontal and vertical moves have a length of 1 pixel and diagonal moves have a length of square root 2 (square shaped pixels assumed). The decision about which pixel to follow relies on the examination of up to 5 neighbours before the new tracing direction is established. In contrast to crack codes, where a "crack" between the pixels is followed, only one or two pixels have to be considered.

This technique is favoured by software implementations of contour followers [Pugh 1983]. For the implementation in our laboratory set-up such an algorithm was chosen. If required, an eight direction code can be generated during the image scan by comparing the last two of the "crack code" elements and assigning the corresponding 8 direction chain code element using, for example, a look-up table.

2.2.6 Object recognition

In object recognition, the final stage of many machine vision tasks suitable features from an appropriate shape description are matched against a library of different feature sets from different objects.

Statistical features

In simple cases global features are compared. The SRI parameter [Agin et al 1975] described above, provides the base for many industrial systems whereby statistical features are directly compared with a library of parameter sets. Often these parameter sets are learned by showing a sample a number of times. Average values of parameters taken of these samples are then held as a reference to recognise the object [Foith 1982]. As discussed above, statistical features are only useful if objects presented are always isolated and fully visible. In all other cases a suitable boundary description has to provide significant features.

Boundary \ local features

In all boundary related features it is very important to get the measurement along the borderline as accurate as possible. The reason for this is that many features relate geometrically to certain criteria, for example, the local-feature-focus method introduced by Bolles et al [1982]. Measurements from significant boundary locations such as corners to
geometrically to certain criteria, for example, the local-feature-focus method introduced by Bolles et al [1982]. Measurements from significant boundary locations such as corners to hole centres are taken as part of a recognition process. This procedure allows the identification of objects even if they are overlapping. However, accuracy of boundary measurements are crucial for a successful analysis.

Many other methods rely on the slope angle representation [Ballard 1982] which is particularly useful in combination with a polygon approximation scheme. A slope angle diagram presents the length of a polygon segment against its angle. Some papers refer to the describing polygon as polar vector description [Kammenos 1978]. As discussed above, most polygon approximation schemes suitable for machine vision applications require a boundary description in the form of a chain code. Now, in order to compare local boundary features in the slope angle domain the boundary length has to be as accurate as possible within reason. The segment length becomes a critical issue. In order to find a close description, a theoretical error analysis of chain coded straight lines is presented in Chapter 3. The error analysis and their approach is my own work. In contrast to other publications which are briefly discussed in the introduction of chapter 3, the validity of the error analysis includes all practical pixel shapes.

Extending the error analysis for circular arcs and combinations with straight lines leads to a correlation process between the error analysis and a vector description of the arbitrary shape. Details are presented in Chapter 4.

From these results it is obvious that chain code representation is not accurate enough. The logical next step is optimisation: algorithms to improve chain code descriptions. In Chapter 5 two new methods - one for crack codes and one for chain codes are developed and compared to 4 other suggested methods.

Finally to find an acceptable shape approximation for the purpose of recognition, the arc operator is introduced in Chapter 6. The performance is then tested and compared to Pavlidis's split and merge algorithm as well as to a shape taken from Teh [1988] who compares a number of angle detection algorithms with his own approach.

2.3 Industrial Vision Systems

The main advantages for industrial applications are seen mainly in the consistency of the operation, and the processing speed which is required for short manufacturing cycles. However, illumination, dirt and dust, the necessity for a carefully controlled environment and costs have to be considered [Rosen 1979].

To simplify industrial vision systems, the input is usually reduced to a binary (black and white) image, so that objects appear as a silhouettes [Foith et al 1981]. The majority of industrial vision systems represent the object in terms of their global characteristic features, such as area, number of holes, etc.. The resulting feature vector is then matched using a search process to the corresponding model.
Global thresholding is commonly used in factory applications for two reasons. One, the lighting can often be controlled sufficiently and two, the image processing operation can easily be implemented in hardware with very short processing times, for example, utilizing global thresholding [Bosch SAM system - Foith 1981]. The ASEA vision system on the other hand uses a gradient operation combined with thresholding in order to cope with a high degree of variation [Braggins 1984].

Since the availability of board level products, specialised hardware features for (low-level) image processing have become a standard. Typical features include look-up table techniques for intensity updates including global thresholding and image processing operations limited to a predefined region (window or area of interest). This kind of sub-image operation can be advantageous, in particular for inspection tasks with the object placed in a fixed position. More advanced features comprise tailored computer architectures to allow pixel based processing from one frame store via one or more programmable hardware filters into another frame store [Bowman 1986].

Industrial systems are generally available on three different levels.

a) Board level products are special-purpose computer interface boards typically complete with a comprehensive software package (except in the case of our laboratory system). Software packages generally provide basic (low-level) image processing routines combined with helpful user interfaces. However, shape description and recognition procedures are not generally available and left for the user to develop. These modular systems are ideally suited for research and development work.

b) Complete specialised task oriented systems for recognition or inspection - recently introduced even on a programmable logic controller (PLC) basis [Allan Bradely 1987]. Many of the vision systems in this category provide parameters from the SRI feature set embedded as part of the device. The systems have become very user friendly and easy to program [Doerr 1983]. Sample objects are presented for recognition in each of their expected stable states. The parameters are extracted from an average model for future reference.

c) Integrated robot vision systems are similar to the above category but provide the necessary link between motion and vision by default. The image processing instructions become part of the robot motion program in a suitable shop floor case [Leichsenring 1984]. However, severe restrictions in the transformation of image data into motion coordinates and the request for separated objects are limiting factors.

Silhouette or boundary related description is rarely exploited for industrial applications.

2.4 Laboratory Equipment & Development

This section briefly describes the set-up of the image processing system at Auckland University. This visual sensor system was employed for experiments described in chapters 3 to 6 as well as for two practical applications: firstly the optical measurement of geometric shape parameters of Kiwifruit - a commercial project with the Department of Scientific and Industrial Research (DSIR), and secondly a robot vision system, purely as a technology demonstration.
The first section describes the hardware of the vision sensor system and the software developed by the author. The second part presents its first application in early 1985: An automated measurement of Kiwifruit shape parameters for statistical purposes. The final section presents an application of a vision-aided flexible component handling system [Friedrich 1986].

2.4.1 Equipment

Hardware components

The computing part is based on a DEC micro 11 with two special-purpose image processing boards (Matrox): a real-time frame grabber with a 4-bit A/D converter and a frame buffer (memory mapped) for storing a 256 by 256 image.

The aspect ratio of the frame store corresponds to that of the camera in use (generally 4:3 for standard TV cameras). The shape of each individual pixel becomes rectangular and measurements taken from the image have to be calibrated appropriately.

The processing equipment is supplemented by appropriate illumination equipment such as a "light table". The table is designed with two sections offering both back and front lighting. Fluorescent tube lighting is mounted underneath one half of the table and the interior walls covered with aluminium foil for improved reflection. The compartment is topped with opaque glass to provide diffuse lighting. Additional lighting, for example spot lights, can be mounted alongside the reinforced table top. A versatile camera stand allows different viewing positions using either the back or front lit section of the table.

Software components

An universal software package has been developed featuring two kinds of command modes: an interactive mode and an automatic mode. The interactive mode is designed for stand alone operation useful for camera calibration and set-up procedures. In the interactive command mode, three different categories of routines are available: pre-processing functions, feature extraction programs and classification routines.

Development work has concentrated on implementing essential processing functions within a suitable software structure. Image processing routines are written in assembler for reasons of processing speed. The software frame is mainly implemented in Fortran and Pascal. A comprehensive list of available commands is enclosed (Appendix C). These routines have been developed by the author during the initial stage of this work.

Command modes

In the interactive command mode, available processing functions can be executed by entering any of the valid commands. A simple command interpreter verifies the input command and starts execution or issues an error message. One option allows for recording commands, to provide a fast repetition of a command cycle and for storing these commands on disk. Entering the indirect command mode causes the program to accept commands from a file, or from another device, such as a remote control station, a host computer or a robot controller. All commands are recorded in ASCII format so that they can be updated.
by any text editor program. Instructions listed in a command file are executed sequentially. After the end-of-file mark the program returns to interactive mode.

**Input/Output features**

The software package is capable of handling three different types of file I/O including image I/O, chain code I/O and command I/O. These features are only available in local mode, i.e. the sensor system is not communicating to, or receiving instructions from another control unit.

Reading images from, and writing images to the hard disk is achieved via double buffering and takes less than 1.8 sec for a complete image with 256 by 256 pixels at 4 bit resolution. Image input and output is provided in three formats. Two of them are compact codes, whereby in the first case grey level information of 8 pixels are packed into one byte for binary images and in the second case two intensity values are stored in one byte (for 4-bit images). The third format contains one grey-level per byte for each pixel. This category is designed for 6 and 8 bit images and provides compatibility with other image files or programs.

In many situations it is sufficient to store only the shape description and necessary parameters rather than the complete image. The information content is thereby considerably reduced and becomes a variable length file containing important parameters, such as object area, perimeter, centre coordinates, start location of the chain code, and the chain code itself. An ASCII file format was maintained to allow simple file exchange with other computer devices, as described in the application below.

In order to assist in the creation of command files for an automatic execution, a record facility is provided. All entered commands are executed at the same time. Recorded commands are stored in a readable (ASCII) format to allow alteration by editing.

All image processing functions embedded in the 'VISION' module are available to user programs from either a subroutine library 'VISLIB' or at the DEC operating system level. Additional routines have been developed for two special applications described in the following section. Furthermore, programs to generate the error function 'ERFUNC' (Chapter 3) and specific error profiles by applying a correlation 'ERFCOR' (Chapter 4) have been established. Routines for the evaluation of various chain code methods 'ESTPER' (Chapter 5), and programs to demonstrate both the simplicity and reliability of the arc operator for shape analysis, 'ARCOP' have been created and compared to a split and merge algorithm 'SPLMER' (Chapter 6). General routines, such as image printing 'IMAPRI', chain code conversions 'CCCHAIN', generation of polygon description and slope-angle diagrams 'SLOANG' are the result of this work.

These programs have been centred around chain code based shape descriptions as can be seen in Figure 2-4. Two different kinds of shape descriptors have been used for file notations, namely chain codes and polygon descriptors. Chain code notations have been used as an output from the low-level vision program 'VISION', and as an input to chain converting program 'CCCHAIN' and the shape approximation routines 'ARCOP' and 'SPLMER'. Polygon based shape descriptions have been used as output from the arc operator function and the split and merge routine. Polygon based shape descriptions can be read by the slope angle conversion routine to allow the display of the original (chain code) and the
approximated shape (polygon) in the slope angle domain which is explained in the next section.

![Diagram of program library hierarchy](image)

**Figure 2-4:** Hierarchy of program library for camera taken images.

The complete suite of routines including its options is documented in Appendix C. A comprehensive help facility is also available where the user can find complete explanations of the keywords used.

### 2.4.2 Kiwi fruit measurement

Compilation of data from optical measurements of Kiwifruit shape parameters was required for an evaluation of the application potential for sorting Kiwifruit by machine vision. The set-up of a comprehensive database was initiated to statistically analyze the relationship of the fruit weight and the area measured from one or two views. Furthermore, the database was used to optimize Kiwifruit packaging.

**Experimental Set-up**

The basic idea was to process the fruit profile represented by its shadow from each of the three views: "flat" view (showing the largest area), a "raised" view (showing a narrow silhouette) and the top view (showing the smallest shaped area). In the arrangement approximately parallel light is obtained from an incandescent lamp with frosted glass diffuser and an aperture plate. A sharply defined silhouette of the object is focused at the camera level, ready for processing by computer. This set-up features a very sharp silhouette
with a high contrast to ease low-level image processing operations by using a global threshold. Figure 2-5 shows the experimental set-up designed by the DSIR.

![Figure 2-5: Kiwifruit work station](image)

**Parameter extraction**

Since only one fruit silhouette is presented at a time, data extraction could be kept very simple. The location of the fruit remains approximately the same so that the boundary can be found by searching from the centre of the image until the intensity value exceeds a given threshold. From this first boundary location a simple tracing algorithm was used to generate the chain code description of the fruit. The tracing algorithm is based on the crack code in order to keep the number of decisions for each step at a minimum. All required parameters could be extracted from the chain code directly while the outline is traced essentially using an algorithm published by Cunningham [1981]. As described earlier, this tracing routine was extended to incorporate a thresholding function. This approach is very efficient because the algorithm decides whether a pixel belongs to the background or the object purely on a threshold basis. Therefore only boundary pixels are processed rather than the entire image.

The data extracted included area, perimeter, length and width in x and y directions, as well as an 8 direction chain code describing the shape. The actual data collection was tray oriented so that after entering the individual tray number, data files were created automatically with the actual tray identification. Data of all three views per fruit of each tray were collected and stored in an individual so-called logical disk unit. These individual disk partitions were tailored in size for each grade of fruit to optimize disk storage.
The image processing system formed the centre of this system and had been linked to the DSIR computer network via serial line for data transmission. The complete database was analysed by the DSIR and an average fruit shape for each grade was revealed as a reference size for packaging design. Figure 2-6 shows the average shape of a Kiwifruit (flat view) of a particular grade with its 2-standard deviation limits (inner and outer borders). A square of 40 mm side length is superimposed for size comparison. Since its first trial set-up in 1985, Kiwifruit measuring has become an annual service to local industry. In 1987 this service was expanded to exotic fruits, such as Nashi pears [Clist 1987].

Apart from assistance by data entry operators, data extraction, storage and communication were entirely my own work in the initial application.

![Figure 2-6: Average Kiwifruit shape](image)

2.4.3 robot vision

Robot vision sensors are often implemented in separate processors because visual sensing is processing intensive. Therefore, an additional requirement involves communication to the robot device. The robot system must be able to interface with the external system at the physical level as well as the level of symbolic data. Parameters and data have to pass in both directions. For example, commands or requests may be issued by the control unit and results or data returned to the controller. Moreover, in many applications sensors need to be programmed.

Programming can be achieved either manually (which may be required in the set-up stage only) or automatically via down-loading of program information from the host controller. Variable data exchange plays an essential role in such a flexible system.

A robot vision system was implemented by the author in which the visual sensor was activated at the request of the robot controller [Friedrich 1986 and Appendix E]. The results of image analysis, part number and position were then transmitted to the robot.
controller. For demonstration purposes it was decided to use a selective screwing operation, in which up to seven different nut sizes could be handled by the system, three of them were screwed onto the appropriate bolt, while the other four were sorted in pre-allocated positions.

![Robot Vision set-up at Auckland University](image)

**Figure 2-7:** Robot Vision set-up at Auckland University.

**Hardware Components**

The robot vision system comprises the following components (Figure 2-7):

- An industrial robot (IRb 6/2 - ASEA)

  - Visual sensor equipment including a DEC micro 11 (LSI 11/23+) with special-purpose interface boards and additional graphic hardware to display results of image analysis.
  - A TV-camera to capture scene images.
  - A control box to manually interact with the process.
  - Peripherals such as a light bench (pick-up location), a screw-plate, and storage bins as end positions.

**Overall function**

The TV camera delivered the scene image, which was stored in a frame store for further processing. Image analysis was activated on request by the robot control. Subsequently, the object silhouette was extracted by a contour follower which was a slightly modified version from the Kiwifruit application described above extended by the thresholding operation. In the first instance the algorithm searches the object from the edge of the image. Once an intensity transition occurs the boundary is traced.

A set of simple geometric features (area, perimeter and hole size) were extracted and matched against a library of reference models. Results were displayed graphically on a monitor. Upon request, location and identity of the object were transmitted to the robot.
The object is then picked up from the presented location and either screwed onto a fitting bolt or placed in the appropriate storage.

The 5-axis robot is equipped with a versatile pneumatic gripper, constructed at the University of Auckland, featuring interchangeable and adjustable fingers. For simplicity, the robot's wrist movement is chosen to perform the screwing operation, although in an industrial application a screwing head would be used instead. The operation mode and some interaction with the process, can be directed from a simple control box. As an operator interface, the control box allows various system options to be selected. It also displays the system state and robot or sensor data.

![Diagram](image)

(a) No object in pick-up position — analysis not required.
(b) Possible object to be identified.
(c) Object out of position/object unknown — analysis not required.

**Figure 2-8**: Rapid object position test.

Depending on input information the robot can perform alternative pre-programmed movements. The hardware and software option at the time of implementation allowed communication to external devices only on a single bit level. Optional firmware has been installed in the meantime to allow full data exchange with a host computer. Despite the limitations in communication at the time, commands to the sensor, such as calibration or image analysis, could be given by the robot control, and information such as object identity and pick-up position are received and processed in a simple binary format. Since positional information could not be processed directly, pick-up locations were indexed.

This restriction simplifies object position detection enormously, since the processing of picture data is concentrated on a small image area surrounding the actual pick-up location. The typical object silhouette also allows a rapid position check, by exploiting shape properties. The thread hole in the nut centre was used as a reference to roughly verify the position before the actual image analysis was started. Just one scan line in the centre of the area of interest needs to be examined to allow a number of conclusions to be drawn as shown in Figure 2-8.
Input and output lines are divided into sensor I/O and I/O for manual control (Figure 2-9). Operator requests, such as mode selection via control box, are processed by the robot control system, and commands are generated to activate the sensor system. A request/busy line synchronizes the robot/sensor data transfer. The sensor information is used to decide on the gripping position and the handling procedure.

![Figure 2-9: Schematics of the robot vision system.](image)

**Modes of operation**

The process is divided into three different operation modes, viz. calibration, automatic robot vision and interactive robot vision. In the **calibration mode** all handling positions are verified by the operator to allow a new set-up to be mapped into the program space. At the same time the vision sensor is also in calibration mode. A reference pattern is shown to the system to identify size (distance) and orientation.

In the **automatic robot vision mode** (ROBVIS), objects are analyzed and handled according to a predefined position sequence. In other words, only object identity is passed between sensor and robot control, because the pick-up sequence is fixed. A missing object causes the system to skip one position and issue a new request for object analysis.

The **interactive robot vision mode** (INROVI) is a direct simulation of a manufacturing situation, where several objects are required by a number of production machines operating at various cycle times. In this set-up the operator can request an object via keyboard input (softkeys). The request is handled by the image processing system and the results of the analysis, identity and position, are transmitted to the robot for further processing. In cases where the required object is not recognized, an error message is issued and a new input requested.
The operation mode, and some interaction to the handling system can be directed from a control pendant which also displays the system state and robot/sensor data.

2.4.4 Shortcomings of the two systems

In order to get reliable measurement results from the Kiwifruit study, each set of fruit silhouettes were accompanied by a reference shape. It was noticed that only a square lined up parallel to the scan direction produced accurate results. Circular reference shapes showed an average deviation of the perimeter in the order of 6% to 7%. Area data were accurate to an acceptable deviation of around 1%. These observations sparked the investigation of chain codes and the search for a better approximation technique. In particular, since around 2000 fruit were processed, each with three views and approximately 1 kByte in chain code and parameter information. This 6 MByte of data could be considerably reduced by applying a boundary approximation scheme (Chapter 5 and 6).

The accuracy of chain codes depend to some extend on the pixel configuration and on pixel geometry. The following section describes various pixel shapes and provides a background to the considerations in Chapter 3.

In the robot vision application, noticeable shortcomings included difficulties in recognizing even a very simple shape due to illumination changes and reflections. Object recognition based on statistical features is very sensitive to small changes. Furthermore, parts with similar statistics or touching parts cannot be distinguished. A more robust and versatile method would need to be based on local features.

The ultimate goal of recognizing partially visible shapes relies on a suitable boundary related description. It has been demonstrated that the length of object sides and their enclosed angles are essential and sufficient for shape recognition [Davis 1977]. Boundary segments in the form of a polygon contain exactly these two parameters, namely segment length and orientation. A polygon description can be derived directly from a chain code description. Algorithms for this are discussed in section 2.7.

2.5 Shape of picture elements

Image forming is one of the common steps shared by various visual sensing applications. Because it is the first step in obtaining relevant data, it is also the most crucial one. The shape of individual picture elements (pixels) is of particular importance for any physical measurement taken from a digitized image representation. The following considerations on pixel shapes are relevant in all cases where any geometric information is obtained from the digitized image.

2.5.1 Quantization

Measurements of light intensity from an image are quantized in order to store and process the information in a digital computer. The measurements taken are equally spaced along each scanning line, ideally on a square grid of points corresponding to a n by n matrix of integer numbers representing the light intensity.
However, since most applications rely on TV-based images the aspect ratio provided is commonly rectangular (4:3) rather than square (1:1) as some industrial solid state cameras are. The resulting array of numbers is therefore either rectangular with square pixels (more columns than rows) or the array is square but individual pixels are rectangular in shape.

Square pixel forms can still be obtained if an appropriate A/D conversion rate is applied and conversion pulses are masked out accordingly. In fact, any part of an image can be digitized using this kind of masking technique and areas of interest can be updated in the frame store without the necessity of over-writing a complete image.

2.5.2 Square pixels

The shape of the pixel element depends on both the aspect ratio of the camera used and the rate of A/D conversions. To obtain, for example, a maximum number of square pixels from a TV-based image the conversion frequency would need to be

\[ f_c = f_s \cdot l_f^2 \cdot r_c \]  \hspace{1cm} (2.1)

with \( f_c \) : conversion frequency
\( f_s \) : frame rate per second,
\( l_f \) : lines per frame and
\( r_c \) : the aspect ratio of the camera employed.

Equation (2.1) reflects the serial nature of the transfer of image information between the camera and the computer. For the European TV standard (CITT) the conversion rate amounts to 13.021 MHz and 11.025 MHz for the American TV standard (EIA).

2.5.3 Rectangular pixels

Frame stores are typically organized in a \( n \) by \( n \) (squared) manner whereby pixels are rectangular in shape unless a commercial camera with a square aspect ratio is employed or the above conversion rates and masking techniques are applied. Most calculations of measurements taken from the image therefore require scaling factors to compensate for the non-square pixel shape. Accurate compensating factors are obtained by measuring a known object in a calibration procedure which also maps the "pixel domain" into the "m-m domain" of the real world.

2.5.4 Hexagonal pixels

If the measurements taken are offset by half an interval every alternate line a hexagonal pixel pattern is obtained. Hexagonal sampling is facilitated by the field interlace mode of TV scanning, which means that offsetting the sampling time need be done only once every field rather than once every line (in TV scanning a frame consists of two fields, the first containing even numbered lines and the second containing the odd numbered lines).
Pixel arrays organized in hexagonal patterns are desirable because they provide equal spacing of the six neighbouring pixels and can therefore provide an equal sampled description of an object silhouette. Scholten [1983] examines chain coding in a hexagonal pixel environment.

The issue of various pixel shapes and their respective errors is explored in detail in Chapter 3, where a complete analysis of chain code errors is presented. The error analysis also includes other chain codes with multiples of 8 direction vectors. Moreover hexagonal pixel patterns are also examined and their respective 6 direction chain codes as well as multiples of 6 direction codes.

Chapter 3 presents a new approach to the error analysis of chain codes which includes practical pixel shapes such as square, rectangular, and hexagonal ones.

2.6 Algorithms to Improve Chain Code Descriptions

Due to the limited vector directions available, crack codes overestimate the boundary length or the perimeter. The extent of overestimating depends on the occurrence of diagonal boundary segments. Chain codes also overestimate the length of the outline to some lesser extent but still significantly, namely in the order of 5 to 6% [Ellis 1979]. This is due to the fact that chain codes can only accurately represent boundary segments which are directed in the same direction as the base vector of the chain code representation.

Proffitt [1979] presents an error analysis of chain codes examining the errors introduced by chain coding using a variety of directions. This approach is mainly a mathematical one and ignores practical situations where pixel shapes are not square. In fact, many (if not most) image sensing equipment is based on a 4:3 aspect ratio which results in most cases in rectangular pixels. In Chapter 3 an comprehensive error analysis which takes these facts into account is presented.

Horn [1986] suggests to use an average factor to compensate for perimeter overestimating using crack codes. This factor (\(\pi/4\)) is equivalent to an average of the deviation error over all directions. This approach is too general and only accurate for circular shapes. For other shapes the perimeter deviation would depend on the orientation of the square.

A crack-code based contour tracing is suggested by Dunkelberger [1985]. The approximation algorithm relies on half-sized pixel steps. This approach basically doubles the resolution which Dunkelberger claims would result in a more accurate contour description. For very small objects, as considered in the publication (up to 20 pixel side length), this is the case. However, in most cases a finer resolution is required and available. Shapes used in this work occupy a significantly larger pixel area. The calculations and samples given do not therefore apply in most practical cases. In fact there is no advantage in terms of accuracy over other contour approximation schemes (compare Chapter 5).

Dessimoz [1983] suggests a length adjustment of certain chain code combinations. The approximated segment length of a particular boundary configuration is organized in a look-up table for computing ease. A considerable advantage can be seen by applying this technique. Detailed comparison of the above methods can be found in Chapter 5.
2.7 Shape Approximation

For object identification, chain codes are not suitable due to their limited number of angle representations. A digitized boundary consisting of a set of pixels can be described most accurately by using small vectors one pixel in length pointing from one pixel position to the next along the boundary, i.e. chain codes. This type of boundary polygon contains much redundant information, for example repeating base vectors to describe a straight line in that direction. However, for the characterisation of objects it is important to describe the boundary with the fewest possible segments avoiding any redundancy. Techniques to combine repetitive code sequences are numerous.

Laing [1977] describes a simple method to convert a sequence of code vectors into a form of a "run length" code. A run length code consists of the length of the run - in this case the number of code vectors - followed by the direction - here the code number itself. In other words the length of the new boundary vector is adjusted to the maximum "run length" possible but still restricted to one of the four or eight base directions. For object identifying purposes the approach taken shows only minor advantages to the original chain code.

A more comprehensive approach is taken by Kaneko [1985]. Boundary segments are encoded in an efficient way containing length and orientation of the segment. The major advantage is the ability to accommodate a large number of possible directions and different segment lengths in a very compact format. The major application of the procedure is for image transmission. Processing requirements of this method are not favourable for industrial machine vision applications.

A method for finding the minimum perimeter polygon has been proposed by Slansky [1972]. The method relies on a one pixel wide strip of boundary pixels, analogous to a rubber band stretched inside the boundary chain. In concave segments of the outline the minimum perimeter touches the inside of this boundary strip and the outside in convex parts. The result is an increased accuracy for the perimeter length with the drawback of very high sensitivity to any noise introduced. The reason for this is that the description chosen is too closely linked to the original outline. Any minor change is reflected in this kind of shape approximation.

In fact, all algorithms relying on a pixel-to-pixel description are not tolerant enough to avoid miss-interpretation with noise presence. For any further processing the outline has to be smoothed. Polygon approximation techniques can overcome this kind of problem.

In general, there are two approaches to determine dominant points on a digital curve. Firstly, extremes of the curve can be found directly through angle or corner detection schemes. The second approach is to obtain a piece-wise linear polygonal approximation of the digital curve, subject to certain constraints on the goodness of fit. The first method relies on finding important direction changes in the boundary i.e. corners. The latter method is essentially that of performing a side detection [Davis 1977].

Since dominant points of a curve correspond to points of high curvature, various existing dominant point detection algorithms first compute an estimate of the curvature at each point on the curve [Freeman 1977]. Next, a two stage procedure is applied to choose dominant points. In the first stage, some input threshold is applied to the curvature estimates to
eliminate points with too low a curvature to be considered as corner points. In a second stage, a process of non-maxima suppression is applied to the remaining points whose curvature estimates are not local maxima in a sufficiently large segment of the curve.

Rosenfeld [1973] uses a cosine measure for finding angular changes. An appropriate smoothing factor has to be selected based on the level of detail represented in the digital curve. This factor is a function of the region of support as a base to compute the measure of significance. A major problem arises when the outline contains features on various levels of detail. Too large a smoothing factor will miss fine details, while too small a factor will give rise to non-dominant points, resulting in additional redundancy. Sankar [1978] employs the sum of local curvature as a significance measure. Anderson [1984] uses tangential deflection. These methods are both compared in [Teh 1988].

Teh [1988] suggests three different measures of significance. The first criteria is his k-cosine measure: the angle between pixels being +k and -k pixels apart. The second measure is based on curvature: the difference in mean angular direction of k pixels, and the third measure is the curvature of a small region of support. The method requires 3 passes with very good results. The algorithm is compared in his paper with the other algorithms briefly described above. A test shape which consisted of 4 semicircles was used to compare the Teh algorithm the other algorithms. This test shape has been implemented and used for comparison with the arc operator in Chapter 6.

Slansky and Gonzalez [1980] use a cone-intersection method to find the longest possible segment. The Euclidian distance between individual segments and the approximated curve should be below a preset value. An acceptable approximating polygon should pass through circles drawn about each point. Tangents are drawn to the circles in order to find sectors in which an acceptable segment must lie. Cone intersection methods require moderate processing time but do not produce optimum results, particularly inflections such as corners are not reproduced accurately.

Kurozumi [1981] uses a minimax method to find a polygon with a minimum number of sides and the maximum distance allowed between the original boundary and the approximating polygon. A vertex giving the maximum perpendicular distance to the proposed line is found. If the deviation of a parallel line passing through the mid-point of the perpendicular is not within the given tolerance, then the segment is discarded and the previous value is accepted. Minimax methods give optimum results but are rather complex. Boundary points are not consistently processed in sequence and several points are required for the calculations in each step.

Another method [Lide 1982], is based on the idea that a boundary segment can be approximated by a straight line segment if a strip can be found which contains all boundary pixels and is not wider than a given tolerance. The longest possible segment is searched to achieve the least number of vertices.

A widely published and very useful approach is the so-called split and merge algorithm [Pavlidis 77]. The merging procedure examines an error criteria, commonly the maximum Euclidian distance of the original outline and the line to be fitted. By exceeding an error threshold a new vertex is accepted. Some of the problems of this merging technique, for example, difficulties in locating corners, can be reduced by combining the method with a
splitting procedure. Initially the boundary is divided into arbitrary segments. Boundary segments are subdivided if the maximum perpendicular distance exceeds a preset threshold. Then subsets are split and merged in order to satisfy the error criteria. Split and merge algorithms produce good results but are time consuming because of the need for multiple passes.

Pavlidis [1982] presents a suitable algorithm for a straightforward computer implementation. In order to find vertices, the maximum perpendicular distance between object outline and approximating line has to be calculated for each point along the boundary. Exceeding the threshold will subdivide the current segment and calculations have to restart again from the starting point. This procedure is repeated until the new segment satisfies the distance criteria and the following segment is examined. The repetitive nature of this technique requires a reasonable processing time which tends to exceed industrial requirements for machine vision. This split and merge algorithm has been used as a comparison to the arc operator developed in Chapter 6.

2.7 Discussion

Algorithms for the approximation of shape boundaries tend to be rather complex and time consuming. The main reason for this is that most techniques are developed for various other related application areas where operational speed is not the most significant requirement. The principal criteria for an industrial application are firstly processing speed and secondly reliability and consistency. In many cases existing algorithms have to be adapted to suit these requirements.

Existing commercial systems tend to use statistical measures for shape description. This approach is very fast but also limited in a number of ways. A more versatile approach is local boundary description. The most simple boundary description, the chain codes are not sufficiently accurate for recognition purposes and tend to overestimate perimeter length. Chapter 3 investigates the errors made by chain and crack code representations. From the error analysis formulated for straight line and curved segments, Chapter 4 extends the theory to arbitrary shapes. Existing algorithms to improve errors of the chain code description are investigated in Chapter 5 and compared to two new schemes, one for chain codes and another one for crack codes.

Chain codes and crack codes are impractical for recognition because the description of the boundary is too detailed and too sensitive to noise variation. Since object outlines can be described adequately by their angles and sides, a smoothed boundary description such as a polygon approximation is required for this purpose. Chapter 6 introduces a new method for fast shape approximation suitable for local feature recognition and therefore able to deal with overlapping parts.
CHAPTER THREE

3. THEORETICAL ERROR ANALYSIS FOR CHAIN CODED STRAIGHT LINES

3.1 Introduction

Computer vision systems applied to recognition, inspection or measurement tasks frequently rely on silhouette information. Many commercial vision systems, often referred to as early binary vision systems, utilize the SRI feature set [Agin 1975] including boundary related parameters such as perimeter, compactness and excursions from the centre of area. Other shape recognition systems require local geometric properties, for example the curvature as a function of arc length.

Where object recognition or feature measurement is not limited to isolated fully visible components but aimed at recognizing overlapping, touching or partially visible objects, boundary related parameters are extremely important.

The computed length of a digital line can only be approximate because of the discrete nature of the underlying continuous image. An object outline can be described by parameter functions x(s) and y(s). Digitization of x(s) and y(s) yields piecewise constant functions and the reconstructed outline consists of individual picture elements (pixels).

This chapter presents a new method for analysing errors introduced by the length estimation of digitized straight lines. The new approach applies to various code sets such as the most common 8 direction chain code and the 4 direction crack code. Moreover, the analysis includes other code sets as well which consider more than one neighbour pixel in each direction. Furthermore, the analysis is not restricted to digitization schemes based on a square raster. The new approach verifies results for square pixel configurations published previously [Proffitt 1979].

In contrast to [Kulpa 1977 and Proffitt 1979] this new method allows for any aspect ratio (rectangular pixel shapes) as well as different pixel shapes (e.g. hexagonal shaped pixels) and various pixel configurations including triplets. Discrete and continuous versions of the approximating error function are introduced and analysed. The approach to the error analysis as well as the extension to other code sets, pixel shapes and configurations, represent original work.

3.1.1 Pixel shapes and configurations

Picture digitization is achieved almost exclusively by applying a raster (grid) digitization scheme. A constant sampling/digitization clock results in square or rectangular shaped pixels. By offsetting every alternate row by half a pixel width, a triplet pixel configuration can be generated. In standard interlace mode which is common to all TV and video devices for the consumer electronic market, this can be achieved simply by an offset of half a pixel width every alternate field. Figure 3-1 shows a survey on pixel shapes and configurations,
including square pixel shape (a), rectangular pixel shape (b), hexagonal pixel shape (c), triplet configuration based on square pixels (d) and triplet pixel configuration based on rectangular pixels (e).

Although the pixel layout in triplet configurations is applied in solid state camera technology (colour cameras) and hexagonal pixel configurations are technically feasible, practically all computer systems up to now employ square or rectangular shaped raster digitization schemes simply because these pixel configurations can be handled more easily by computer hardware using column and row addressing principles, similar to memory addressing methods.

![Pixel shapes and configurations](image)

**Figure 3-1:** Pixel shapes and configurations

### 3.1.2 Chain Code sets for different pixel configurations

Most methods of estimating digital lines therefore rely on 8-direction chain codes or the 4 direction crack code [e.g. Cunningham 81], where pixels are treated as tiny tiles and the object is traced along the "cracks" rather than from pixel centre to pixel centre.

In both cases the outline is divided generally into small line segments reflecting the move from one pixel to the next along the border line of the object of interest. The parameter \( s \) may be defined as an element of curve arc length:

\[
ds = \sqrt{dx^2 + dy^2}
\]

Estimating the curve length of digitized curves, e.g. for perimeter extraction, is achieved simply by adding the distances between centres of subsequent pixels forming the edge of an object. This method corresponds to the 8 direction chain code where one pixel \( k = l \) in each direction is considered or in other words pixels in a 3 by 3 neighbourhood are
investigated. Other code sets include larger neighbourhood areas including 5 by 5 \((k=2)\) and 7 by 7 \((k=3)\) as shown in Figure 3-2.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
</tbody>
</table>

**Figure 3-2:** Chain code sets for \(k = 1,2\) and 3

The number of different possible direction vectors, which may be taken from a point within a defined neighbourhood, conveniently describe these coding schemes. For \(k\) neighbour pixels in a square/rectangular raster configuration there are exactly \(k \times 8\) direction vectors available. The corresponding code sets for hexagonal pixel configurations amount to \(k \times 6\) code elements. Figure 3-3 shows code sets for hexagonal pixel configurations for \(k = 1,2\) and 3.

**Figure 3-3:** Code sets for hexagonal configurations

3.2 Estimating The Length of Digitized Straight Lines

A digitized straight line is generally approximated by a sequence of elements belonging to the specific chain code set applied. Estimation errors introduced by digital representations of lines are subject to truncation and rounding errors due to their discrete nature.
Furthermore, pixels generated by a solid state camera, for example, do not map onto the digitized picture elements stored in the computer memory. This effect occurs because the analogue but discrete CCD signal is digitized with an independent clock rate. Therefore, a camera generally provides different numbers of pixels and a different pixel layout to those in a frame store.

The following analysis ignores these effects of digitization and noise content. Theoretical error magnitudes refer, therefore, only to the minimum possible error values. The analysis is presented in a continuous rather than discrete form because the discrete version is a special case of the overall continuous function at a certain angle resolution. However, initial discrete equations are shown to illustrate the derivation of the general method.

3.2.1 Crack code errors

For a 4 direction code, such as the crack code, the length of a digital line is estimated by adding the number of horizontal \( n \) and the number of vertical \( m \) elements multiplied by their individual unit length \( H \) and \( V \) respectively. For any angle \( \alpha \) of a digital line the estimated length \( L_e \) amounts to:

\[
L_e = n \cdot H + m \cdot V 
\]

(3.2)

corresponding to

\[
L_e = L_t \cdot (\sin \alpha + \cos \alpha) \]

(3.3)

where \( L_t \) represents the true Euclidian distance (length).

![Figure 3-4a: Error definition for the crack code](image-url)
The error contribution for horizontal elements amounts to

$$Err_{4h} = \cos \alpha - \cos^2 \alpha$$

and for vertical elements to

$$Err_{4v} = \sin \alpha - \sin^2 \alpha$$

combining to

$$Err_4 = Err_{4h} + Err_{4v}$$

The normalized deviation (Figure 3-4b) for a 4-direction code yields

$$Err_4 = \sin \alpha + \cos \alpha - 1$$

(3.4)

![Graphical error contribution Crack code (4 directions)](image)

Figure 3-4b: Error function for a 4 direction code
3.6 Error analysis

3.2.2 Chain code errors

Now consider an 8 direction chain code. The length estimation will be more accurate by introducing diagonal line elements. Any digital line with a slope angle of $0 < \alpha < \pi/4$ is represented by:

$$L_e = (n-m) \cdot H + m \cdot D$$

and for slope angles of $\pi/4 < \alpha < \pi/2$ by

$$L_e = (m-n) \cdot V + n \cdot D$$

(3.5)

where $D$ represents the unit length for diagonal elements (i.e. $\sqrt{2}$) for square pixel grid. The corresponding Error functions are:

$$Err_8 = \cos \alpha + (\sqrt{2} - 1) \sin \alpha - 1$$

$$Err_8 = \sin \alpha + (\sqrt{2} - 1) \cos \alpha - 1$$

(3.6)

These definitions are illustrated in Figure 3-5 (a + b) including a graphical representation of segment contributions as a function of the slope angle $\alpha$ of the approximated line. The deviation error of the available code elements are shown below.

![Figure 3-5a: Error Definition for the 8 dir code](image)

For horizontal elements:

$$Err_{8h} = (\cos \alpha - \sin \alpha) \cdot (1 - \cos \alpha)$$

for diagonal elements:

$$Err_{8d} = (\sqrt{2} \cdot \sin \alpha) \cdot [1 - \cos (\pi/4 - \alpha)]$$
in the range of $0 < \alpha < \pi/4$ and

$$\text{Err}_{8d} = (\sqrt{2} \cdot \cos \alpha) \cdot [1 - \cos (\pi/4 - \alpha)]$$

in the range of $\pi/4 \leq \alpha \leq \pi/2$ - and for vertical elements:

$$\text{Err}_{8v} = (\sin \alpha - \cos \alpha) \cdot (1 - \sin \alpha)$$

3.2.3 Hexagonal code errors

In a hexagonal grid the length deviation for the basic 6 direction code amounts for $0 \leq \alpha \leq \pi/3$ to

$$\text{Err}_{6} = \cos \alpha + 1/\sqrt{3} \cdot \sin \alpha - 1$$

and for $\pi/3 \leq \alpha \leq 2\pi/3$ to

$$\text{Err}_{6} = 2/3 \cdot \sqrt{3} \sin \alpha - 1$$

(3.7)

Individual error contributions of the available code elements (Figure 3-6) are

$$\text{Err}_{6h} = (\cos \alpha - \sqrt{3} \sin \alpha) \cdot (1 - \cos \alpha)$$

$$\text{Err}_{6v} = 2\sqrt{3} \sin \alpha \cdot [1 - \cos (\pi/3 - \alpha)]$$
3.8 Error analysis

Graphical error contribution
Chain code (6 directions)
Hexagonal pixels

![Graphical error contribution](image)

Figure 3-6: Error function for hexagonal pixels (k = 1)

3.3 The Fundamental Error Function

In general, a digital straight line can be represented by only two elements of a particular code set: the code element with the nearest slope angle ($\phi_1$) lower than, and the neighbouring element with a slope angle ($\phi_2$) larger than the original slope of the approximated line. This setup constitutes a triangle (Figure 3-7) and presents the basis for the following error analysis.

![Approximation triangle](image)

Figure 3-7: Approximation triangle
The original length \( L \) (here \( c \)) is normalized to 1. Sections \( a \) and \( b \) are the neighbouring code elements \( i \) and \( i+1 \) which approximate the line best, so that the best estimate is:

\[
\text{Err} = L_c - L \quad \text{with} \quad L_c = a + b \quad \text{and} \quad L = c = 1
\]

\[
\text{Err} = a + b - 1
\]

with the sine law

\[
a = \frac{\sin \alpha}{\sin \gamma} \quad \text{and} \quad b = \frac{\sin \beta}{\sin \gamma}
\]

The error function becomes

\[
\text{Err} = \frac{\sin \alpha + \sin \beta}{\sin \gamma} - 1
\]  
(3.8)

The angles \( \phi_i \) and \( \phi_{i+1} \) represent the two nearest available slope directions of code elements with \( \phi_i < \alpha < \phi_{i+1} \). The angle \( \alpha \) corresponds here to \( (\alpha - \phi_i) \) and the angle \( \beta \) to \( (\phi_{i+1} - \alpha) \). The angle \( \gamma \) is given by \( (\pi - \phi_i + 1 - \phi_i) \). Substituting yields:

\[
\text{Err} = \frac{\sin(\alpha - \phi_i) + \sin(\phi_{i+1} - \alpha)}{\sin(\phi_{i+1} - \phi_i)} - 1
\]

or using the following substitutes

\[
A = [\sin(\phi_{i+1}) - \sin \phi_i] / C
\]

\[
B = [\cos \phi_i - \cos(\phi_{i+1})] / C \quad \text{where}
\]

\[
C = \sin(\phi_{i+1}) \cos \phi_i + \cos(\phi_{i+1}) \sin \phi_i
\]

therefore

\[
\text{Err} = A \cos \alpha + B \sin \alpha - 1
\]  
(3.9)

Applying the sine law again to Figure 3-7, the error contribution of the two code elements yield:

\[
\text{Err}_i = \frac{\sin(\phi_{i+1} - \alpha)}{\sin(\phi_{i+1} - \phi_i)} \left[1 - \cos(\alpha - \phi_i)\right]
\]

for the \( i \)th code element in \( \phi_i \) direction and similarly

\[
\text{Err}_{i+1} = \frac{\sin(\alpha - \phi_i)}{\sin(\phi_{i+1} - \phi_i)} \left[1 - \cos(\phi_{i+1} - \alpha)\right]
\]  
(3.10)

for the \( i+1 \)th code element in \( \phi_{i+1} \) direction, where

\[
\text{Err} = \text{Err}_i + \text{Err}_{i+1}
\]
3.3.1 Slope ranges for square and rectangular pixels

For code sets using $k$ neighbouring pixels (Figure 3-2, 3-3) the available element directions are expressed in the following equations. A rectangular pixel shape (Chapter 2.3.2) with an aspect ratio $r$ gives the available slope angles of:

$$\phi_i = \arctan\left(\frac{i}{rk}\right) \quad \text{with} \quad i = 0...k$$

(3.11)

with the aspect ratio

$$r = \frac{dx}{dy}$$

where $r = 1$ for square pixels and $r \neq 1$ for rectangular pixels (e.g. 4/3).

The valid range of (3.9) can therefore be summarized as:

$$i \ (r \ k) \leq \tan \alpha < \ (i+1) \ (r \ k) \quad \text{with} \quad i = 0...k \quad \text{for} \quad 0 \leq \alpha \leq \pi/4$$

and as

$$(r \ k) / (2k-i) \leq \tan \alpha \leq (r \ k) / [2k-(i+1)]$$

with

$$i = k ... 2k \quad \text{for} \quad \pi/4 \leq \alpha \leq \pi/2$$

Coefficients for the complete error function, valid for all raster code sets of any $2k + 1$ by $2k + 1$ neighbourhood (for example the 8 direction code for $k = 1$, the 16 direction code for $k = 2$ or the $n*8th$ direction code for $k = n$) are shown below. The coefficients of the error function (3.9) valid for $0 \leq \alpha < \pi/4$ are:

$$A = \{ \cos \left[ \arctan \left(\frac{i}{rk}\right) \right] - \cos \left[ \arctan \left(\frac{i+1}{rk}\right) \right] \} / C$$

$$B = \{ \sin \left[ \arctan \left(\frac{i+1}{rk}\right) \right] - \sin \left[ \arctan \left(\frac{i}{rk}\right) \right] \} / C$$

(3.12)

where

$$C = \sin \left[ \arctan \left(\frac{i+1}{rk}\right) \right] \cos \left[ \arctan \left(\frac{i}{rk}\right) \right] -$$

$$- \cos \left[ \arctan \left(\frac{i+1}{rk}\right) \right] \sin \left[ \arctan \left(\frac{i}{rk}\right) \right]$$

valid for $i = 0...k$ in

$$Err = A \cos \alpha + B \sin \alpha - 1$$

(3.9)
3.3.2 Slope ranges for hexagonal pixels

In hexagonal patterns pixel neighbours are equally distant in slope intervals of $\pi/3$. The base units in $x$ and $y$ direction depend on the pixel shape. For hexagonal shaped pixels:

\[ dx_h = \cos \left( \frac{\pi}{3} \right) = 0.5 \]
\[ dy_h = \sin \left( \frac{\pi}{3} \right) = \sqrt{3}/2 \]

For square and rectangular pixels (aspect ratio $r$) configured as triplets the corresponding values are:

\[ dx_t = 0.5r \quad \text{and} \quad dy_t = 1 \]

Considering $k$ neighbouring pixels in each direction the available direction vectors are defined for slope angles between 0 and $\arctan (dy_h/dx_h)$ as:

\[ \phi_i = \arctan \left( \frac{i dy_h}{k + i (dx_h - 1)} \right) \quad \text{for} \quad i = 0 \ldots k \]

and for slope angles between

\[ \arctan \left( \frac{dy_h}{dx_h} \right) \quad \text{and} \quad \arctan \left( \frac{dy_h}{-dx_h} \right) \]

\[ \phi_i = \arctan \left( \frac{k dy_h}{k (dx_h - 1) - i} \right) \quad \text{for} \quad i = k \ldots 2k \] (3.13)

3.4 Maximum and Average Magnitudes of Length Deviation

In order to find maximum error values, the first derivative of (3.9) is set to zero for each of its definition ranges, where the second derivative has to be positive. The latter condition need not be proved because within each valid range there is only one maximum and all minimum values are zero and occur at both borders of its definition range.

The first derivative of the error function is:

\[ Err' = \frac{\partial Err(\alpha)}{\partial \alpha} = B \cos \alpha - A \sin \alpha \] (3.14)

with a local maximum of

\[ Err_{\max} = A \cos (\alpha_{\max}) + B \sin (\alpha_{\max}) - 1 \] (3.15)

at the slope angle $\alpha$

\[ \alpha_{\max} = \arctan \left( \frac{B}{A} \right) \]
Total error values are obtained by a piecewise integration of the error function (3.9) over all valid ranges from 0 to \(\pi/4\) for square pixels (because of symmetry) and from 0 to \(\pi/2\) for rectangular pixels. Using symmetric hexagonal pixels it is sufficient to integrate the error function to \(\pi/6\). However, for triplet configurations based on either square or rectangular pixels, the integration limit is \(\pi/2\) due to the common symmetry of square or rectangular pixels to the hexagonal grid pattern. The resulting formulae are shown below.

\[
E_{\text{err}} = A \sin \alpha - B \cos \alpha - \alpha
\]  
(3.16a)

or equivalent to

\[
E_{\text{err}} = \int_{\alpha=0}^{\pi/2} E_{\text{err}}(\alpha)d\alpha - \pi/2 = \sum_{i=0}^{n} \phi_{i+1} \int_{\alpha=\phi_{i}}^{\phi_{i+1}} E_{\text{err}}(\alpha)d\alpha - \pi/2
\]

Evaluating the integrals yield

\[
E_{\text{err}} = A \sum_{i=0}^{n} (\sin \phi_{i+1} - \sin \phi_{i}) + B \sum_{i=0}^{n} (\cos \phi_{i} - \cos \phi_{i+1}) - \pi/2
\]  
(3.16b)

with \(\phi_{i}\) as formulated in (3.11) and (3.13).

The normalized average error in % amounts to:

\[
E_{\text{ave}} = \left( \frac{E_{\text{err}}}{\pi/2} \right) \times 100\%
\]  
(3.17)

3.4.1 Square pixel error function

The error function (3.9) for all code sets which are based on a common raster grid is naturally symmetric about both the x and \(\mathbf{y}\) axes, so that the function needs to be specified for only one of the quadrants. In addition, the function is also symmetric about all diagonal lines for code sets using square pixels with the exception of the 4 direction code. This code is a special case of the code distinction by neighbourhood size. Hence, the expression for all other code sets can be reduced to \(k\) equations, e.g one for an 8 direction code \((k=1)\), two for a 16 direction code and so forth.

Substituting \(\beta\) for \(\alpha + \pi/4\) produces the following error function for the first quadrant using the 8 direction code \((k=1)\) with square pixels (Figure 3-5a):

\[
E_{\text{err}} = \cos \beta + (\sqrt{2} - 1) |\sin \beta| - 1
\]  
(3.18)

valid in the range of \(-\pi/4 \leq \beta \leq +\pi/4\).

The maximum error is 8.24% at \(\beta=\pi/4 \pm \pi/8\) or \(\alpha=22.5\) and \(\alpha=67.5\) degrees respectively. The average error (3.17) is 5.5% for square pixels.
3.4.2 Rectangular pixel error function

Rectangular pixel shapes require a set of $2^*k$ equations to describe the error function considering $k$ pixels in each direction. The results of numeric calculations for a common case of an aspect ratio of 4:3 and $k=1$ are shown below (Figure 3-8a).

![Graphical error contribution Chain code (8 directions) Rectangular pixels](image)

Figure 3-8a: Error function for rectangular pixels

The error function can then be expressed as follows:

For $0 \leq \alpha \leq \arctan 1/r$ (here 3/4)

\[
Err_8 = \cos \alpha + [\sqrt{(r^2 + 1)} - r] \sin \alpha = \cos \alpha + 1/3 \sin \alpha
\]  

(3.19a)

with a maximum error of 5.4% at 18.4 degrees and

\[
Err_8 = \sin \alpha + [\sqrt{(r^2 + 1)} - 1]/r \cos \alpha = \sin \alpha + 1/2 \cos \alpha
\]  

(3.19b)

where $\arctan (1/r) \leq \alpha < 1$

with a maximum error of 11.8% at 63.4 degrees.

The mean error for rectangular pixels applied to an 8 direction code is slightly higher than that for square pixels, namely 6.1%.
3.4.3 Hexagonal pixel error function

The error function of hexagonal shaped pixels is naturally symmetric about \( \pi/3 \). The basic equations for 6 direction code are:

\[
Err_6 = \cos \alpha + (1/\sqrt{3}) \sin \alpha
\]  

(3.20)

For \( 0 < \alpha < \pi/3 \) with a maximum error of 15.5\% at \( \pi/6 \), and for \( \pi/3 < \alpha < 2\pi/3 \)

\[
Err_6 = (2/3 \sqrt{3}) \sin \alpha
\]

with a maximum error of 15.5\% at \( \pi/2 \).

The mean error for hexagonal pixels applied to a 6 direction code is 10.2\%. Figure 3-8 below surveys the error function for length estimation considering one neighbouring element in each direction for square, rectangular and hexagonal pixels.

![Error function for square, rectangular and hexagonal pixels](image)

Figure 3-8: Error function for digitized straight lines

3.4.5 Extended Code Sets

The following table summarizes maximum and average errors in \% introduced by applying chain code sets for the length estimation of digital straight lines for 4 to 24 code directions.
### Maximum estimation errors in %

<table>
<thead>
<tr>
<th>pixel shapes and configuration</th>
<th>Code set</th>
<th>no. of directions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>square pixels</td>
<td>41.4%</td>
<td>-</td>
</tr>
<tr>
<td>rectangular p.</td>
<td>41.4%</td>
<td>-</td>
</tr>
<tr>
<td>hexagonal pix.</td>
<td>-</td>
<td>15.5%</td>
</tr>
<tr>
<td>triplets (squ)</td>
<td>-</td>
<td>17.5%</td>
</tr>
<tr>
<td>triplets (rec)</td>
<td>-</td>
<td>20.2%</td>
</tr>
</tbody>
</table>

### Average estimation errors in %

<table>
<thead>
<tr>
<th>pixel shapes and configuration</th>
<th>Code set</th>
<th>no. of directions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>square pixels</td>
<td>27.3%</td>
<td>-</td>
</tr>
<tr>
<td>rect. pixels</td>
<td>27.3%</td>
<td>-</td>
</tr>
<tr>
<td>hexagonal pix.</td>
<td>-</td>
<td>10.3%</td>
</tr>
<tr>
<td>triplets (squ)</td>
<td>-</td>
<td>10.5%</td>
</tr>
<tr>
<td>triplets (rec)</td>
<td>-</td>
<td>10.6%</td>
</tr>
</tbody>
</table>

\[ k = 1 \quad k = 2 \quad k = 3 \quad k = 3,4 \]

<table>
<thead>
<tr>
<th>no of neighbour pixels considered</th>
</tr>
</thead>
</table>

**Table 3-1: Length estimation errors for chain codes.**

The large deviation values (c.f. 4-direction code) arise because of the large angle difference between the direction vectors available in crack coding. These errors are reduced significantly by introduction of additional direction vectors (Figure 3-9).

Each direction vector is represented by one zero value in the error function. Additional direction vectors therefore produce extra zero values in the error function at their specific slope angles. In a 3x3 neighbourhood \((k = 1)\), there are exactly two zero points in each quadrant, e.g. at \(\pi/4\) and \(\pi/2\) in the first quadrant. For \(k = 2\) there are 4 zero points in each quadrant giving 16 directions (Figure 3-10), and so on.
Figure 3-9: Average error for digitized straight lines

Figure 3-10: Error function for a 16 direction code
Figure 3-11: Error function for a 24 direction code

Figure 3-11 shows the error function for a 24 direction code. Each angular interval between direction vectors corresponds to a section of the error function with one maximum between both definition borders where the error function is equal to zero. Note the error function is always positive because the considered straight line is approximated by a limited number of direction vectors (chain codes) in a discrete pixel environment. The estimated length of the ragged approximated line is therefore always longer than the original straight line (see also Figure 3-5a).

The pixel configurations based on a hexagonal pattern are symmetrical about π/3. Hence for each additional neighbouring pixel six additional directions are introduced. Figure 3-12 shows a survey of the error function for 6 directions or \( k = 1 \) (a), 12 directions or \( k = 2 \) (b), and 18 directions or \( k = 3 \) (c).

Each additional direction vector produces a new section of the error function including a zero point at the new direction and a maximum at half the angle difference. Figures 3-13 (a + b) show the maxima of individual angular sections for a 72 direction code. 9 neighbouring pixels are considered for square and rectangular pixels and 12 pixels are considered for a hexagonal pixel environment.

The section maxima are higher at quadrant borders 0 and \( \pi/2 \). The reason for this is that the angle differences of available direction vectors are larger around zero and \( \pi/2 \) and lowest around \( \pi/4 \). For hexagonal pixels the angle differences are higher around \( \pi/6 \) and lower around 0 and \( \pi/3 \). It is interesting to see that the envelope of individual section maxima (Figure 3-13a) correspond to angle differences of available code vectors (Figure 3-14).
Error analysis

3.18

*Figure 3-12:* Error function for hexagonal pixels

*Figure 3-13a:* Error maxima for 72 direction codes
**Envelope of error maxima**
for 72 direction code
Hexagonal pixel configurations

![Graph showing error maxima for 72 direction code with hexagonal and rectangular pixels.](image)

Figure 3-13b: Error maxima for triplet and hex. pixels

**Angle between code vectors**
(72 direction code)

![Graph showing angle intervals for rectangular, square, and hexagonal pixels.](image)

Figure 3-14: Angle intervals for $k = 9$ and $k = 12$
With the increasing number of code directions, the error reduction follows a straight line in a double logarithmic diagram of the average error versus the number of neighbouring elements (number of directions) [Proffitt 79]. These findings apply for square and hexagonal pixels where the aspect ratio is 1:1. Although with an increasing aspect ratio this function is distorted, it follows the same principle with a higher number of direction vectors (which is, however, not practical). Summarizing the error function, it should be noted that the average error decreases considerably with an increasing number of directions available to match a given slope angle of the digital line. In fact, the error value reduces roughly exponentially with increasing numbers of available code elements, giving the following approximate expression:

$$Err_k = \frac{Err_1}{k^r}$$  \hspace{1cm} (3.21a)

where $Err_1$ is the average error introduced using 8/6 directions ($k=1$) which is 0.055 for square pixels, 0.061 for rectangular pixels (aspect ratio 4:3) and 0.103 for hexagonal pixels, so that:

$$\log Err_k = \log Err_1 - r \log k$$  \hspace{1cm} (3.21b)

where $r=2$ for aspect ratios around 1 (incl. 4:3).

Figure 3-15: Average errors $k=1$(8dir) to $k=100$ (800dir)

However, for higher aspect ratios $r$ reduces, for example, to 1.92 for an aspect ratio of 2:1, to 1.82 for an aspect ratio of 3:1, and to 1.67 for an aspect ratio of 5:1. Higher aspect ratios up to 2:1 are very common in graphic display devices and high resolution screens. Equation 3.21 applies practically to all pixel configurations. Square and hexagonal pixel shapes are beneficial because equally distributed angles result in a lower average deviation as shown
in (Figure 3.9). This result is plotted in Figure 3.15 showing average errors of various pixel shapes and configurations ranging from the 8 direction code \((k = 1)\) to an 800 direction code \((k = 100)\).

However, since the final result is still subject to noise and distortion, the computing effort required has to be carefully considered to match the application. It may not be useful, for example, to apply a chain code set with an average error rate far below the expected noise and distortion rate of the image, because error sources are independent and therefore cumulative. In practice, the limiting factor for an estimation of the line length is the resolution and accuracy of the quantization for both end positions of the straight line.

3.5 The Matrix Form of The Error Function

The continuous error function is strictly valid only for infinitely spatial resolution. The sampled version, by contrast, represents the limitation in spatial resolution which introduces additional errors in practice. A length tolerance of \(\pm 1\) pixel, for example, is due to the quantization process involved. Other additional errors are caused by the restriction of direction vectors limiting in turn the number of slope angles which can be approximated error free by combinations of code vectors.

The following section introduces a discrete version of the error function. This approach takes some of these additional factors into account, such as spatial resolution and ideal quantization. However, deviations arising from noise and illumination or object separation deficiencies are excluded.

For sampled straight line boundaries, slope angles can have only a limited number of values due to the discrete nature of the pixel elements. The estimation error can therefore be expressed as a series of discrete function values at the available inclinations. Alternatively, deviation errors can be arranged as a function of discrete coordinates in a two dimensional table form. An error matrix is a suitable presentation, where the indices represent the digital (integer) values of horizontal and vertical line projections respectively. The matrix presents error values as a two-dimensional function of coordinates \(x\) and \(y\), rather than as a function of the slope angle \(\alpha\) as developed in the previous section.

The error matrix is similar to a sample error function. However, the algorithm to establish the matrix also takes practical limitations into account. The code sequence is built up from a 4 and 8 direction code. Suitable code sequences are replaced by code elements from the higher code set. Code elements from a lower order are left over in cases where available code sequences do not fit.

The line length deviation is taken into account by the fact that code elements - if necessary from different code sets - are fitted to approximate the line between its starting and end point. In fact, the algorithm converts location differences of the line end points into crack code vectors. In this way the digitized line is matched by chain codes as closely as possible.

In order to establish the error matrix for an 8 direction code, pairs of vertical and horizontal elements are replaced by a diagonal vector. Similarly, for higher level codes suitable code combinations of the lower level code are replaced by appropriate new code elements.
Obviously, the error function may be developed from this point of view in a straight forward, simple manner which in fact is a suitable algorithm for a computer implementation, as described below.

For a given slope to be approximated, suppose \( n \) and \( m \) represent the numbers of horizontal (H) and vertical (V) pixel units respectively. The approximation for a 4 direction code simply yields:

\[
L_4 = nH + mV
\]

- \( n \): no. of horizontal elements (H)
- \( m \): no. of vertical elements (V)

Considering an 8 direction code with \( (D) \) being the diagonal line element. The approximation results in:

\[
L_8 = aD + bH + cV
\]

- \( a = \min (n,m) \): no. of diagonal elements (D)
- \( b = n - a \): no. of horizontal elements (H)
- \( c = m - a \): no. of vertical elements (V)

The number of diagonal code elements is simply the minimum of both vertical and horizontal elements. The remaining elements are either of horizontal or vertical direction. Hence, one of the coefficients \( a, b, \) or \( c \) is always zero, since the slope of the straight line will be approximated by the neighbouring code directions on either side.

Other code sets can easily be calculated by applying this concept more often as illustrated in more detail by the following flow chart.

For a 16 direction code \( (k = 2) \) two additional vectors per quadrant are introduced: one vector between the diagonal and the horizontal direction called \( (DH) \), the other one between the diagonal and the vertical direction called \( (DV) \).

The number of DV code elements is the minimum number of diagonal elements and vertical elements. The estimated length amounts to:

\[
L_{16} = dDH + eDV + fD + gH + hV
\]

- \( d = \min (a,b) \): no. of diag./horiz. elements (DH)
- \( e = \min (a,c) \): no. of vert./horiz. elements (DV)
- \( f = a - \min (a,b+c) \): remaining diagonal elements (DD)
- \( g = b - d \): remaining horizontal elements (H)
- \( h = c - e \): remaining vertical elements (V)

Only 2 coefficients for the nearest declination angles to the approximated slope angle differ from zero, all other coefficients are zero. This pattern is repeated for larger code sets in the same manner. Thus, the 24 direction code can be established from an eight direction code in a simple method shown in the flowchart below (Figure 3-16).
Figure 3-16: Flow chart for error matrix values

\[ L_{24} = i \cdot DHH + k \cdot DDH + l \cdot DVV + p \cdot DDV + q \cdot H + r \cdot V + s \cdot D + d \cdot DH + e \cdot DV \]

- \( i = \min \left( \frac{g}{2f} \right) \) : no. of diag./horiz. elements (DHH)
- \( k = \min \left( \frac{f}{2g} \right) \) : no. of diag./horiz. elements (DDH)
- \( l = \min \left( \frac{h}{2f} \right) \) : no. of diag./horiz. elements (DVV)
- \( p = \min \left( \frac{f}{2h} \right) \) : no. of diag./vertical elements (DDV)
- \( q = (g - 2i + k) \) : remaining vertical elements (V)
- \( r = (h - 2l + p) \) : remaining horizontal elements (H)
- \( s = (f - (i + 2k + l + 2p)) \) : remaining diagonal elements (D)

Applying this new method, the error function can be calculated very rapidly for common code sets. The error matrix represents a discrete form of the error function at a given resolution. Table 3-2 shows a matrix for a 10 by 10 pixel neighbourhood in each quadrant for an 16-direction code \((k = 2)\). The zero sequences in the matrix correlate exactly with slope angles at which the error function is zero. In Appendix A a complete set of matrices for 4 to 24 direction codes is included.

The matrix form may be compared with theoretical results of the continuous form by choosing a line length and an orientation. Suppose the length is chosen as \( l = 10 \) pixels, and the slope angle as \( \alpha = 20 \) degrees. The approximating coordinates for the error matrix are found by applying (3.22) with the results: \( x = 9, \ y = 3 \).

\[
\begin{align*}
  x &= \text{Round} \left( l \cos \alpha \right) \\
  y &= \text{Round} \left( l \sin \alpha \right)
\end{align*}
\] (3.22)
Chain code error matrix for digital straight lines

Estimation error of 16 code in % - aspect ratio: 1:1

<table>
<thead>
<tr>
<th>Y \ X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.85</td>
<td>2.69</td>
<td>2.57</td>
<td>1.62</td>
<td>0</td>
<td>0.95</td>
<td>1.30</td>
<td>1.18</td>
<td>0.71</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
<td>2.74</td>
<td>2.33</td>
<td>0.97</td>
<td>0.61</td>
<td>1.24</td>
<td>1.24</td>
<td>0.78</td>
<td>0</td>
<td>0.71</td>
</tr>
<tr>
<td>8</td>
<td>2.16</td>
<td>2.74</td>
<td>1.92</td>
<td>0</td>
<td>1.09</td>
<td>1.29</td>
<td>0.86</td>
<td>0</td>
<td>0.78</td>
<td>1.18</td>
</tr>
<tr>
<td>7</td>
<td>2.33</td>
<td>2.64</td>
<td>1.21</td>
<td>0.75</td>
<td>1.31</td>
<td>0.95</td>
<td>0</td>
<td>0.86</td>
<td>1.24</td>
<td>1.30</td>
</tr>
<tr>
<td>6</td>
<td>2.52</td>
<td>2.33</td>
<td>0</td>
<td>1.24</td>
<td>1.06</td>
<td>0</td>
<td>0.95</td>
<td>1.29</td>
<td>1.24</td>
<td>0.95</td>
</tr>
<tr>
<td>5</td>
<td>2.69</td>
<td>1.62</td>
<td>0.95</td>
<td>1.18</td>
<td>0</td>
<td>1.06</td>
<td>1.31</td>
<td>1.09</td>
<td>0.61</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2.74</td>
<td>0</td>
<td>1.29</td>
<td>0</td>
<td>1.18</td>
<td>1.24</td>
<td>0.75</td>
<td>0</td>
<td>0.97</td>
<td>1.62</td>
</tr>
<tr>
<td>3</td>
<td>2.33</td>
<td>1.24</td>
<td>0</td>
<td>1.29</td>
<td>0.95</td>
<td>0</td>
<td>1.21</td>
<td>1.92</td>
<td>2.33</td>
<td>2.57</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1.24</td>
<td>0</td>
<td>1.62</td>
<td>2.33</td>
<td>2.64</td>
<td>2.74</td>
<td>2.74</td>
<td>2.69</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.33</td>
<td>2.74</td>
<td>2.69</td>
<td>2.52</td>
<td>2.33</td>
<td>2.16</td>
<td>2.00</td>
<td>1.85</td>
</tr>
</tbody>
</table>

Table 3-2: Error matrix for short digitized lines

The quantization process may be simulated by rounding the results of equation 3.22. The corresponding value in the matrix (Figure 3-17) represents the deviation error in % of a 16 direction code for square pixels for which the matrix was established. The error value for the considered straight line amounts to 2.33% compared to 2.04% of the theoretical value. Figure 3-17 shows the deviation error of this line over half of the first quadrant (0 to π/2) compared to the theoretical result, which is length independent due to the infinite spatial resolution.

![Figure 3-17: Matrix values compared to error function](image-url)
The magnitudes of quantization errors are comparatively higher at this short line length and are not shown for this reason. However, practical experiments which take quantization errors also into account are discussed in Chapter 5.

The described discrete procedure is generally suited for various code sets and pixel configurations including hexagonal based configurations, provided a suitable matrix addressing is employed in this latter case.

3.6 Conclusion: Error Analysis

A new approach for analysing deviation errors on the estimation of chain coded lines has been introduced. The estimation is based on the fact that a digital line is approximated by the two nearest available chain code elements. The error function expresses the length deviation, introduced by chain coding, as a function of slope angle and pixel configuration. Square, rectangular and hexagonal based pixel grids are discussed in detail. Two approaches to generating the error function have been developed. One leads to a continuous form and the other to a matrix form. The continuous form can be seen as a digitization scheme with an infinitely high spatial resolution. The practical oriented matrix form is an approximately sampled version of the continuous form.

The results of the new method confirm earlier published results for square pixel configurations. In addition, the behaviour of different pixel shapes and configurations is analysed and mathematically described. The outcome basically verifies the hypothesis that symmetrical pixel shapes and patterns are advantageous over pixel configurations which are based on rectangular pixel shapes. However, hexagonally based grids are less sensitive to aspect ratio variations than orthogonally based grids. The main advantages are a lower mean error and a smaller maximum error because deviations are distributed evenly.

The following chapter examines approximation errors of chain coded arbitrary curves and outlines which can be described by straight line segments and circular arc segments. A new algorithm based on these results takes advantage of a combination of chain codes. This method is introduced and discussed in Chapter 5.
CHAPTER FOUR

4. PERIMETER ESTIMATION ERROR OF CHAIN CODED SILHOUETTES

4.1 Introduction

The previous analysis has shown that the expected error in estimating the length of a digitized straight line can be described by the error function (Chapter 3). The error function \( \text{err}(c, \alpha) \) presents deviation errors as a function of the inclination angle \( \alpha \) of the straight line. The shape of this function is unique to the pixel shape, pixel configuration and coding scheme used.

In order to estimate the expected perimeter deviation of a chain coded polygon, each side (vector) of the polygon has to be treated as an individual straight line. The error function returns a deviation value for each side of the polygon. The error components, which are specific to the orientation of the considered side, have to be normalized to perimeter length. The total of the scaled error values represents the perimeter deviation of the polygon.

Circular arcs can be included in the analysis and be described by an integration of the error function. Hence, arbitrary silhouettes consisting of straight line segments and circular arcs can be evaluated by this new approach. The procedure can be described by a correlation which will be discussed in this chapter.

4.2 Principle of Length Deviation Using the Error Function

Polygons can be examined by considering each vector (side) as an individual straight line. For example, the perimeter deviation for a square shape in a regular array of square pixels can be calculated quite simply as follows. Each side of the shape (square) contributes a quarter of the total deviation error.

In this particular case, the perimeter deviation for a square and/or a rectangle corresponds exactly to that of a straight line with the corresponding sloping angle \( \alpha \). This is due to the quadrant symmetry of both the considered shape and the properties of the error function associated with the particular pixel shape and configuration. The total deviation error for the perimeter of a square at a given orientation, \( \alpha \), is then expressed as:

\[
\text{Err}_{\text{sq}}(\alpha) = \frac{1}{4} \text{err}(\alpha) + \frac{1}{4} \text{err}(\alpha + \pi/2) + \frac{1}{4} \text{err}(\alpha + \pi) + \frac{1}{4} \text{err}(\alpha + 3\pi/2)
\]

which is equivalent - due to symmetry - to

\[
\text{Err}_{\text{sq}}(\alpha) = \text{err}(\alpha)
\]

with \( \alpha \) being the orientation of the considered square and its corresponding straight line. Figure 4-1 shows the deviation error of a square oriented at \( \alpha \) employing an 8 direction code. The error function value is in this case identical to that of a straight line at the same slope angle \( \alpha \).
For a rectangle, the result will be exactly the same, assuming again square pixels in a regular array configuration. The new side length ratio causes only the error distribution to change for each side appropriately, but does not change the total length deviation. For example, a 1:2 side ratio will result in an error contribution of the same ratio. In other words, each side of the rectangle contributes the equivalent amount of error according to its share of the overall perimeter length.

In the examples above, the longer sides of a rectangle with a 1:2 ratio contribute 1/3 each to the total perimeter, while the shorter sides contribute only 1/6 of each. The error contribution of each side is therefore identical: 1/3 of the total error is contributed by each of the longer sides and 1/6 by each of the shorter sides. The deviation error amounts therefore to:

\[
Err_{Rec} (\alpha) = \frac{1}{3} err (\alpha) + \frac{1}{6} err (\alpha + \pi/2) + \frac{1}{3} err (\alpha + \pi) + \frac{1}{6} err (\alpha + 3\pi/2)
\]  

(4.2)

which again is equivalent to:

\[
Err_{Rec} (\alpha) = err (\alpha)
\]

**Figure 4-2** shows graphically the deviation error for a rectangular shape at an orientation angle \( \alpha \) using an 8 direction chain code.

Note, this result is only valid for square pixels in a regular array. For any other pixel shape or configuration, the associated error function has to be taken into account. For rectangular pixels, for example, the error contribution is no longer split evenly between the sides of the square. The error function is mirror symmetric to multiples of \( \pi/2 \), but is asymmetric within
each quadrant. The error values are lower at orientations closer to the longer side of the rectangular shaped pixels, i.e. near 0 and \( \pi \).

Since the error function is periodic with \( \pi \), opposite sides obviously contribute equal shares to the total error value, while neighbouring sides contribute different amounts.

**Figure 4-3** shows the error function for the first 2 quadrants with the orientation of the square (60 degree or \( \pi/3 \)) highlighted. Two opposite sides of the square oriented at \( \pi/3 \) and \( 4\pi/3 \) contribute much more to the total error value than their neighbouring sides due to the rectangular pixel environment.

In order to describe the perimeter deviation error over the rotation of a particular polygon the above procedure has to be repeated for each orientation. The result is a dedicated error function describing the perimeter deviation error over the orientation of the particular shape. This unique error function is derived by a correlation process and presents the error variation only for the considered shape described by the chosen chain code set with its associated pixel shapes and configuration.

The result of a correlation procedure for a square and/or a rectangular shape in a rectangular pixel environment is shown in **Figure 4-4**. The above procedure can be applied in a similar manner to any other shape.

The integration of this new error function gives the average length deviation for the given shape and pixel environment.
The following section describes the general method for obtaining specific error profiles for any arbitrary polygon in various pixel environments.
4.3 The Correlation of the Error Function

The procedure above can be formulated mathematically by a correlation between the error function and a vector description of the considered polygon. The vector description presents a diagram of vector length versus slope angle. A polygon with n vertices can be expressed as a series of length and slope angle values representing the length and angle of each side/vertices:

\[ v_p (\alpha) = [ (l_1, \alpha_1), (l_2, \alpha_2), ... (l_n, \alpha_n) ] \]  

The correlation between two functions \( f(x) \) and \( g(x) \) is defined (Gonzalez et al. 1977) for two continuous functions as:

\[ f(x) * g(x) = \int_{-\infty}^{+\infty} f(\alpha) g(x + \alpha) \, d\alpha \]

and in the case of discrete functions

\[ f_e (x) * g_e (x) = \sum_{m=0}^{M-1} f_e (m) g_e (x+m) \]

This definition can be readily applied to the error function in order to compute the perimeter deviation error of a polygon. The correlation of the error function with a vector description \( V_s \) of the shape can be described by the following equation

\[ Err_{poly} (\beta) = Err (\alpha) \ast V_s (\alpha - \beta) \]

For an arbitrary polygon with a vector diagram such as illustrated in equation 4.3, the correlation yields

\[ Err_{poly} (\beta) = \sum_{i=1}^{n} V_p (l_i, \beta_i + \alpha) \, err (\beta_i + \alpha) \]

The correlation can also be illustrated by a graphical method. The vector diagram slides over the error function. For any given value of \( \beta \), the offset angle, every vector length \( l_i \) (normalized to the overall perimeter length) has to be multiplied by the corresponding value of the superimposed error function. The total of all these multiplications presents the deviation error for the considered shape at the orientation \( \beta \). By adding the orientation angle \( \beta \) to each individual angle \( \alpha_i \) of the polygon description, a polygon rotation is achieved.

The vector diagram of a polygon is generally valid between the angles 0 and \( 2\pi \). Since the error function is periodic with \( 2\pi \) - in most cases even with \( \pi \) - there are neither overlapping nor wrap around effects by applying the correlation. Hence, the correlation needs to be calculated only over \( \pi \) or \( 2\pi \).

The following two sections present a few examples of simple geometric shapes, their vector description (4.4) and their correlation with the error function (4.5) in various "pixel environments".
4.4 Examples of Vector Representations for Simple Shapes

4.4.1 Regular polygons

The vector description of a simple geometric form can be expressed as a series of paired segment length and angle values. For example a symmetrical triangle possesses therefore, according to equation 4.3, the following discrete vector description $V_t$:

$$V_t = [(l/3, \beta), (l/3, \beta + 2\pi/3), (l/3, \beta + 4\pi/3)]$$

(4.6)

with $l$ being the common side length and $\beta$ the base orientation.

From this, the general vector representation of a regular polygon will be quite obvious. Assuming a n-sided polygon with a side length of $l$, the expression becomes:

$$V_p = [(l/n, \beta), (l/n, \beta + 2\pi/n), \ldots (l/n, \beta + (n-1)2\pi/n)]$$

(4.7)

Figure 4-5 shows the vector diagram of a regular triangle at an orientation of $\beta = \pi/3$ (60 degrees).

4.4.2 Arbitrary polygons

Any arbitrary polygon can be described similarly, using the individual side length values rather than a constant and slope angle values for each vertex (refer also to equation 4.3).

$$v_p = [(l_i, \alpha_1), \ldots (l_i, \alpha_i), \ldots (l_i, \alpha_n)]$$

(4.8)

with $i$ from 1 to $n$. 
From (4.8) it is clear that equation (4.7) is a special case of the general form above. Figure 4-6 shows a vector diagram for a rectangle with a side ratio of 1:2 at an orientation of 20 degrees.

4.4.3 Circular arcs

The deviation error of circular arcs can be evaluated by integrating the error function (Chapter 3). The vector representation can be seen mathematically as a continuous function with an infinitely small vector length dl. The correlation of this function with the error function presents an integration procedure for each orientation of the considered arc.

The error function for a full circle is therefore constant over all sloping angles and equal to the average error value of all possible slope angles of a straight line using the appropriate error function for the particular pixel environment.

The deviation error for any part of a circular arc can then be calculated by evaluating the integral over the error function with the integral limits corresponding to the arc length and the particular orientation $\alpha$.

$$Err_{\text{cir}} = \int_{\alpha=0}^{2\pi} Err(\alpha)\,d\alpha = 4\int_{\alpha=0}^{\pi/2} Err(\alpha)\,d\alpha$$

(4.9)
4.4.4 Combined circular arcs and straight line segments

A semi-circular shape, for example, may be presented by a combination of the two methods above. The vector representation consists of a single, 2 radii long vector at the given orientation for the straight line part of the shape and an infinite series of $dl$ vectors for the circular arc.

The perimeter deviation of this vector representation is equal to a single value at the declination angle $\beta$ plus the integration of the error function from $\beta + \pi/2$ to $\beta + 3\pi/2$.

The error value for the considered shape at the given orientation $\beta$ becomes:

$$Err_{sc} (\beta) = \frac{1}{2 + \pi} \left[ \pi \int_{a}^{b} err (\alpha) \, d\alpha + 2 \, err (\beta) \right]$$

(4.10)

The general form of 4.9 for any m-th part of a full circle yields:

$$Err_{arc} (\beta) = \frac{2 \pi/ m}{tpl} \left[ \pi \int_{a}^{b} err (\alpha) \, d\alpha + \ldots \right]$$

(4.11)

with the integral limits $a = \beta + \pi/2$ and $b = \beta + 3\pi/2 + 2\pi/m$ and $tpl = the total perimeter length.

Equation 4.11 shows only the circular arc part of a considered shape.

4.5 Examples of Specific Error Profiles

Using the above concept the perimeter deviation of all kinds of shapes can be evaluated. The shape description has to be based on the two shape primitives: straight line segments and circular arc segments. The error function for the particular pixel environment and the chosen code set describes length deviation errors for straight line segments at a given orientation. The error value for the perimeter of the shape is derived from an integration of the error function for circular arc segments and the appropriate error value for each straight line segment.

The individual error components have to be scaled relatively to the total perimeter length in order to take their individual error contributions into account. The sum of all error components gives the perimeter deviation at the orientation $\beta$ of the shape (equation 4.3). The correlation is a repetition of this procedure for all possible orientations of the shape resulting in an integral over the full range of $2\pi$.

Naturally, the perimeter deviation for a circular shape is constant due to rotational symmetry. The error value is equal to the average deviation of a straight line over all possible slope angles (refer to Chapter 3).

The error profile of a semi-circle combines the profiles of a full circle and that of a straight line. The circular arc segment contributes a constant offset scaled down to the relative
of the arc in relation to the total perimeter length. The straight line segment produces an error profile with two (4 direction code), three (6 direction code), or four (8 direction code) maxima over two quadrants or 180 degrees ($\pi$). The magnitude of this error contribution is also scaled down relatively to its share of the overall curve length. The form of the error profile is therefore expected to be similar to that of a straight line, but with a smaller magnitude and a constant offset value (Figure 4-7).

![Semi Circle Rectangular Pixels - Array Configuration](image)

Figure 4-7: Error profile of a semi-circle

The error profile of a triangle is the result of the combination of the three sides considered individually as straight lines (Figure 4-8). For symmetry reasons this profile is similar to that of a straight line in a hexagonal pixel environment. For a 6 sided regular polygon (hexagon) the outcome is again similar to a straight line because the main directions of the side vectors are identical to the available code directions of a hexagonal pixel environment.

The error profile of a square can be obtained by considering the the four sides individually as straight lines. For symmetry reasons this profile is identical to that of a straight line in a square pixel environment (Figure 4-3). For an 8 sided regular polygon (octagon) the outcome would be identical because the main directions of the side vectors are identical to the available code directions in a regular square or rectangular pixel array.

A comprehensive catalogue of error profiles can be found in Appendix B. The collection of error profiles include the following basic shapes: full circle, straight line, triangle and square. These shapes are examined in various pixel environments. The considered pixel shapes include square, rectangular and hexagonal forms. Two different pixel configuration are also included, namely the regular array configuration and the triplet configuration.
4.6 Conclusion

This chapter introduced a practical application of the error function (Chapter 3). A new method, based on a correlation procedure, enables the calculation of principal perimeter deviation errors of chain coded silhouettes. These errors are due to the limited number of vector directions available for the particular chain code set.

Arbitrary shapes which may be described with straight line segments and circular arcs can be included. A vector description of the shape showing segment length over the orientation of the segment is necessary to perform the anticipated correlation.

The error estimation is achieved by separately considering individual shape segments. The segment deviation errors are scaled down according to the length proportion of the considered segment to the entire outline. In other words, the error contribution of each segment is determined by the fraction of the segment length to the overall outline length.

The error contribution of a straight line segment can be looked up in a error function table and scaled accordingly. However, the error contribution of a circular arc has to be calculated by an integration of the error function within its angular orientation limits before scaling can take place.
The main findings can be summarized as follows:

- Deviation errors are zero for all straight line segments in one of the available code directions. Hence, the error for a polygon can only be zero if all straight line segments are orientated at available code directions. For example, this can be the case for a square coded with 4 directions, 8 or multiples of 8 directions at certain orientations, namely the code directions. Similarly, the perimeter length of a triangle or hexagon may be estimated error free if the shape is in line with the code directions in a hexagonal pixel environment.

- Deviation errors of straight line segments are length independent for an infinitely fine resolution, but contribute only very minor additional errors in practical applications. This is due mainly to the length variation of available code combinations for the best approximation of the segment orientation.

- Deviation errors for circles or semi-circular arcs are constant, e.g. orientation independent, and curvature independent at infinitely high resolution. The error contribution of other circular arcs is length and orientation dependent.

These theoretical results are based on an infinitely high spatial resolution and do neither include quantization errors nor digitization errors. In practice, the limitation of spatial resolution introduces additional quantization errors which depend on the orientation and length of the segment. For example, in cases where the segment length is a multiple of the length of a chain code combination which approximates the segment orientation best, the deviation error will be exactly the same as the theoretical error. In other words, the higher the resolution of the digitization grid, the smaller the gap will be between practical and theoretical results. Experiments using common and new methods to reduce the deviation error are described in the following Chapter.
CHAPTER FIVE

5. SILHOUETTE APPROXIMATION BY CHAIN CODES

The previous chapters examined theoretical approximation errors caused by chain coded straight lines and circular arcs. Silhouettes consisting of these two shape primitives were examined in different pixel environments. In this chapter simple geometric silhouettes consisting of one circular arc and/or straight line segments are examined by experiment. Practical limitations are taken into account, e.g. limited spatial resolution and available pixel shapes and configurations. The experiments are carried out for shapes described by chain codes based on rectangular and square pixel shapes in a regular array configuration.

A critical review of traditional chain code methods for the estimation of line length is presented and a new algorithm based on the results of the error analysis is introduced. The new strategy features efficient processing with a high potential for parallel processing. The techniques considered are then applied to simple geometric shapes: circles, semi-circles and squares, with various sizes and different orientations for the latter two shapes. Comparison of the experimental results are then discussed.

5.1 Introduction

The traditional method used to describe arbitrary digitized curves has been published by Freeman [Freeman 1961]. His chain codes are still in use today. Since then a number of algorithms have been developed to overcome the limitations of the Freeman code. Three other methods are reviewed briefly and compared to the chain codes as well as to the new procedures presented in this chapter.

The codes considered are divided into two basic groups. The first group, commonly referred to as chain codes, provides typically 8 (or multiples of 8) vector directions from pixel centre to pixel centre including diagonal moves. The second group is called crack codes. These codes provide segment vectors only in the 4 main directions along the pixel borders: north, east, south and west. Naturally, in a hexagonal environment chain codes have 6 or multiples of 6 code directions and crack codes are limited to either 4 directions (triplet configuration) or 6 directions (hexagonal pixel shapes). In the following, both approaches (crack codes and chain codes) are considered. Due to practical limitations, only square and rectangular pixel shapes in regular arrays rather than in triplet configurations (Chapter 3) are examined.

A digitized straight line is described by a sequence of small vectors representing the chain code elements. Consider an arbitrarily curved digitized line. In order to estimate its length the curve can be approximated by short segments of straight lines. Approximation of digitized arbitrary curves (here silhouettes) leads generally to an over estimation of the outline length. The reason for that is that the original line is crossed over many times by the approximating code vectors due to the limited directions available in a chosen code set. Discrete object borders tend to produce ragged outlines. The length estimation of these ragged outlines tends therefore to over-estimate the actual arc length.
5.2 Silhouette Approximation

Perimeter estimation varies considerably with rotation and size of an object. Consider, for example, a square object. The calculation of the perimeter can only be correct for silhouettes orientated in one of the available vector directions in the chosen code set. For any other angle of rotation, the perimeter estimate is too high because individual line elements form a ragged border. This effect is quite obvious in the case of a square orientated at $\pi/4$ or 45 degrees approximated by a 4 direction crack code. The effect of over-estimation peaks for slope angles which are exactly half way between the available code directions e.g. for $m = \pi/4$ with $m = 1, 3, 5, 7$ for 4 direction codes (Chapter 4).

The amount of error depends, apart from the selected code set, on the noise content and on the mesh size of the sampling grid (digitization frequency) i.e. the resolution of the image in relation to the object size. However, principal estimation errors (Chapter 4) generally exceed quantization errors of this nature. Other contributing factors, such as errors due to noise and changing intensity of illumination, are not examined here.

The total length of the perimeter $p$ is commonly estimated to be equal to the number (n) of code elements of a given code sequence and is described by

$$p = n \quad \text{for square pixels } (r = d_x / d_y = 1) \text{ and}$$

$$p = n_x \cdot r + n_y \quad (d_y \text{ assumed as grid unit or grid constant}) \quad (5.1)$$

for pixels with an aspect ratio ($r$) other than 1:1. $n_x$ represents the number of east/west elements in the code sequence and $n_y$ is the number of north/south code elements.

Obviously, a more acceptable approximation of a digitized arc would be a mixture of various code elements with various lengths and slope angles (Figure 5-1) rather than elements of just one single code set as commonly applied.

![Figure 5-1: Arc approximation with various code sets](image)
Using chain code elements to describe digitized arbitrary curved lines is a special case of polygon approximation with a highly limited range of direction vectors, namely the chain code elements themselves. The principle of employing a mixture of chain code elements from a variety of code sets is used as a base for arc approximation. The goal of this approach aims at an economic improvement of arc length estimation of a digitized curve with a limited number of code vectors rather than an accurate curve approximation. However, a new algorithm for piecewise linear approximation of arbitrary silhouettes is presented in Chapter 6.

5.2 Description of Silhouette Coding Methods

5.2.1 Crack Code Methods

Horn Method

One of the most obvious correction methods in 4 direction coding is to correct the grid unit to an average factor of \( \pi/4 \) or 0.785 [e.g. Horn 1986]. This factor is precisely the average of the error function (Chapter 3) applied to crack codes and reflects the distribution of infinite small vector elements approximating a circle (Chapter 4). Figure 5-2 shows the error function for a square at various degrees of orientation. The zero line represents the average value with an error range of 41.4%.

This approach is far too general and implies that the best result will be achieved for circular and oval shapes. This method therefore works well for this category of shapes, but fails for most other shapes, in particular square, triangular and rectangular shapes (Chapter 4).

![Figure 5-2: Error function for a corrected square](image)
The expected perimeter deviation of the average corrected outline (Horn method) for a square at an arbitrary orientation is simply the difference between the average line and the corresponding error function values. Figure 5-2 shows the error function for average corrected 4-direction coded squares in a square pixel environment. These unacceptably high theoretical error values are verified by experimental results (section 5.3).

**Dunkelberger Method**

Another method [Dunkelberger 1985] takes advantage of a finer grid to approximate the object edge more closely. This approach basically introduces a chain code scheme superimposed on a crack code scheme (Figure 5-3). The method replaces direction changes in the 4 direction code with half length diagonal vectors essentially "cutting off sharp corners".

![Figure 5-3: Silhouette approximation by Dunkelberger](image)

The length approximation of the silhouette is improved by the availability of twice the number of direction vectors. These improvements rely basically on the usage of 8 instead of 4 direction vectors (compare the error function for chain and crack codes). However, this method has similar disadvantages to the original 8 direction chain code by Freeman.

Results by Dunkelberger may be advantageous for small shapes and short line segments as shown in his experiments with squares up to 20 pixel side length. However, this magnitude of side length is not realistic in practical applications where precision is required. With the above spatial resolution, a side length variation of ±1 pixel or ±5% is unavoidable due to the digitization scheme. For precise geometric measurements a close-up view is therefore necessary. Since today's machine vision systems provide usually 512 x 512 or at least 256 x 256 pixels per image frame, object sizes up to a size of 20 pixels in one dimension are considered as small objects.
Experiments with squares, triangles, semi-circles and circles with shape sizes up to 400 pixel side length/diameter (section 5.3) show no significant differences in perimeter deviations when Dunkelberger's method is compared to the Freeman chain codes.

Friedrich Method

This new approximation scheme replaces the length of each segment by an adjusted value depending on the pixel configuration. In a given code sequence three consecutive neighbours are considered: the previous (i), the current (j), and the next following (k) code vector. The length of the current element is altered according to the following code combinations: straight connections (a), corners (b), "stairs" (c), and turn-arounds (d).

a) **Straight connections**: The length of the current element will be one pixel width or one pixel height depending on the aspect ratio and the direction of the straight line.

b) **Corner elements** are approximated by a short-cut formed by 3/4 of a pixel width and 1/4 pixel height for the first part of the corner bit and vice versa for the second part.

c) **"Stair case" configuration**: The element length is exactly half the diagonal of a pixel and reproduces diagonal oriented straight line pieces exactly. This configuration leads to the same result as in Dunkelberger's method.

d) **Turn-arounds**: The length value will be chosen as either half a pixel width or half a pixel height according to the turn-around orientation.

**Figure 5-4** illustrates the approximated outline utilizing this replacement strategy and **Table 5-1** shows details in numeric form.
5.6 Silhouette Approximation

<table>
<thead>
<tr>
<th>VECTOR CONFIGURATION</th>
<th>LENGTH OF ELEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>Code Combinations*</td>
</tr>
<tr>
<td>i,j,k ∈ 0,1,2,3</td>
<td></td>
</tr>
<tr>
<td>(a) i=j=k even</td>
<td>east/west</td>
</tr>
<tr>
<td></td>
<td>odd = north/south</td>
</tr>
<tr>
<td>(b) i=j</td>
<td>i=0,2</td>
</tr>
<tr>
<td></td>
<td>i=1,3</td>
</tr>
<tr>
<td>(c) i=k</td>
<td></td>
</tr>
<tr>
<td>(d) i/=j/=k</td>
<td>i=0,2</td>
</tr>
<tr>
<td></td>
<td>i=1,3</td>
</tr>
</tbody>
</table>

*whereby 0 = east, 1 = north, 2 = west, 3 = south.

Px and Py = pixel dimensions, Py normalized to 1

Table 5-1: Code Combinations and their length replacement factors.

The major advantage of this method lies in its simplicity of implementation. Each code combination can be associated with the address (0...7) of a lookup table containing the pre-calculated length (Table 5-1). The number $n_i$ of each code combination $i$ is then simply multiplied by its associated length value $l_i$ and added up to yield the total perimeter length $p$ of the silhouette.

$$p = \sum_{i=1}^{7} n_i l_i$$

(5.2)

5.2.2 Chain Code Methods

The original Freeman code is used here as a standard reference to compare the effectiveness of other algorithms.

Dessimoz et al. [1983] suggest length adjustments for certain pixel configurations. These configurations include code elements with a direction change of $\pm \pi/4$ and corner elements with a direction change of $\pm \pi/2$ (Table 5-2).

The algorithm considers two adjacent code elements rather than only one immediate neighbouring element as applied in the 8 direction chain code and the crack code. However,
the principle is based on the 8 direction coding scheme and represents a conversion from 8 to 16 direction coding for some code combinations. The error range in estimating the outline length of a silhouette is hereby significantly reduced.

<table>
<thead>
<tr>
<th>GEOMETRICAL CONFIGURATIONS</th>
<th>LENGTH CONTRIBUTION</th>
<th>ELEMENTARY WEIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqrt{2}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/\sqrt{2}$</td>
</tr>
</tbody>
</table>

Table 5-2: Code approximation replacement table [Dessimoz]

Extending this principle of replacing suitable code combination with an adjusted length leads to a mixture of code elements from various code sets. The new method for chain codes by Friedrich examines three consecutive code elements for special code combinations to be length adjusted. The length of these specific code combinations are adjusted to the vector length of these 3 elements. This principle resembles a combination of an 8-direction code for large curvatures and corners, a 16 direction code for medium curved shapes, and a 24 direction code for smaller curvatures and straight lines. A significant increase in accuracy is achieved by eliminating a few code combinations. Table 5-3 gives details in numeric form.

<table>
<thead>
<tr>
<th>Code combination</th>
<th>Length value (square pixels)</th>
<th>8</th>
<th>16</th>
<th>24 direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1.41</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>2.41</td>
<td>2.23</td>
<td>-</td>
</tr>
<tr>
<td>112</td>
<td></td>
<td>3.82</td>
<td>3.65</td>
<td>3.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code combination</th>
<th>Length value (rectangular pixels)</th>
<th>8</th>
<th>16</th>
<th>24 direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1.67</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>2.67</td>
<td>2.40</td>
<td>-</td>
</tr>
<tr>
<td>112</td>
<td></td>
<td>4.33</td>
<td>4.07</td>
<td>4.01</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>4.67</td>
<td>4.51</td>
<td>4.47</td>
</tr>
</tbody>
</table>

Table 5-3: Organisation of a lookup table for length adjustments.
Similarly to the method described for the crack codes, a filter or mask can be thought to move over the serial chain code. Each time a valid combination is discovered, an associated counter is updated. Once the process is completed the corrected length value is then the sum of all code combination counters multiplied by their individual segment length. A total of 10 different segment lengths are used, so that equation 5.2 is updated to

\[ p = \sum_{i=1}^{10} n_i l_i \] (5.3)

using the replacement length values of Table 5-3.

The approximation and smoothing process can be best compared to the fish bone pattern shown in Figure 5-5. The pattern structure applies to each of the four quadrants for finding suitable code combinations which will be replaced by new appropriate segment length values shown in Table 5-3.

**Figure 5-5:** Fishbone pattern for code detection

5.3 Experiments with Simple Shapes.

5.3.1 Silhouette generation

The above methods have been applied to simple geometric shapes, viz. square, triangle, semi-circle and circle. A range of silhouettes of each of these shapes was generated by computer, with variable parameters such as side length, angular orientation and radius. For each of the above methods the circumferences are calculated step by step as the silhouette is established. The pixel locations \( X_n \) and \( Y_n \) for a circular silhouette are generated by using
\[ X_{nc} = r \cos \alpha_n + X_c \quad \text{for} \quad 0 < \alpha_n < 2\pi \]
\[ Y_{nc} = r \sin \alpha_n + Y_c \]  
(5.4)

with
\[ \alpha_n = n \arctan \left( \frac{v}{u} \right) \quad \text{and} \quad X_c, \ Y_c \quad \text{the centre coordinates.} \]

Subscript \( c \) stands for circle and \( n \) for the current pixel position. For each new pixel location, the appropriate code element is calculated by each of the methods. For the semi-circular shape the orientation is an additional parameter which is taken into account. Given the slope angle \( \alpha_s \) the above formula can be applied to the circular part of the shape in the range
\[ \alpha_s < \alpha_n < \alpha_s + \pi \]

The remaining part of the silhouette is a straight line generated by
\[ X_n = r \left[ \cos \alpha_s - l_n \sin \alpha_s \right] \]
\[ Y_n = r \left[ \sin \alpha_s + l_n \cos \alpha_s \right] \]  
(5.5)

where \( \alpha_s \) represents the orientation angle

with \( l_n = -1...+1 \)

Finally the square silhouette and the triangle are generated by using the side length 2a and the orientation \( \alpha_s \)
\[ X_{ns} = a \left[ \cos \alpha_m - l_n \sin \alpha_m \right] \]
\[ Y_{ns} = a \left[ \sin \alpha_m + l_n \cos \alpha_m \right] \]  
(5.6)

using
\[ \alpha_m = \alpha_s + (m-1) \frac{2\pi}{3} \quad \text{with} \quad m = 1...3 \quad \text{for the triangle and} \]
\[ \alpha_m = \alpha_s + (m-1) \frac{\pi}{2} \quad \text{with} \quad m = 1...4 \quad \text{for the rectangle and} \]
\( l_n \) as defined in (5.5).

5.3.2 Code Generation

5.3.2.1 Chain code generation

The positions of silhouette pixels are simulated and obtained by mathematical means rather than via a camera from an actual scene. Hence, the generation of an 8 direction chain code is straightforward. Chain code elements are obtained by the position comparison of two neighbouring pixel locations. Difference code vectors are assigned according to Table 5-4.
Table 5-4: 8 direction chain code from position differences

Most frame stores address individual pixels in the following manner: X locations increase from the left to the right (column) and Y addresses (rows) increase from the top to the bottom (inverse to conventional coordinate systems).

5.3.2.2 Crack code generation

In order to use estimation methods based on crack codes, the generated 8 direction code has to be converted to 4 direction code with the following simple procedure. The generating algorithm establishes the silhouette in the mathematical positive direction (anti-clockwise), hence the chain code also possesses a positive direction. The crack code can be established by comparing two neighbouring code elements providing the tracing direction is known. Based on the difference of these two code elements the 4 direction code is generated. The number of code elements which may be one, two, three or none, depend on the current pixel configuration.

The following Table 5-5 shows examples of the code conversion based on a clockwise tracing direction. The left hand column shows an example of each of the 6 pixel configurations and the appropriate chain code in the next column. The thick pixel sides represent the resulting 4 direction code listed on the left hand side.

The code conversion can be pre-calculated and stored in a look-up table for fast repetitive processing. The interesting fact about this algorithm is that it can operate in both directions from 8 to 4 direction code as well as from 4 to 8 direction code. However, the conversion table above is only valid for crack codes traced in negative direction (clockwise). In order to convert from crack codes traced in the positive direction a slight modification needs to be made. The conversion mechanism for 4 to 8 direction code is shown below.
5.3.3 Experiments with 4 chain coded shapes

In a series of experiments, silhouettes were generated and chain/crack codes were derived. Four basic shapes were examined: circle, semi-circle, triangle and square. Experiments with circles illustrated the influence of spatial resolution on the length estimation with no particular direction preference. Semi circles are suitable to examine the effect of a combination from two basic shape primitives: a straight line and a circular arc component on the perimeter estimation. Tests with squares and triangles basically verified the theory (Chapter 3) on length estimations of digitized straight lines at various orientations. The experiments are divided into two groups distinguished by the chain coding method: 4 direction and 8 direction coding. Examples of different aspect ratios (1:1 and 4:3) are also included.

5.3.3.1 Circles.

Circles were generated with diameters ranging from 40 to 400 pixels. Perimeter deviations of more than 10% were considered unsuitable for any recognition task. It was also found that the results were too unstable for circles below 40 pixels in diameter. (Dunkelberger’s experiments covered square shapes to a side length of only 20 pixels!). With an increasing size of circles, the estimation error reduced to a constant value characteristic for each method examined. As expected, the results for rectangular pixels have shown a slightly higher average compared to a square pixel environment.

<table>
<thead>
<tr>
<th>Pixel configuration</th>
<th>Code (8 dir)</th>
<th>Difference Code (4 dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Pixel configuration image" /></td>
<td>00 01</td>
<td>0 +1 0</td>
</tr>
<tr>
<td><img src="image2" alt="Pixel configuration image" /></td>
<td>02</td>
<td>+2</td>
</tr>
<tr>
<td><img src="image3" alt="Pixel configuration image" /></td>
<td>07 06</td>
<td>-1 -2</td>
</tr>
<tr>
<td><img src="image4" alt="Pixel configuration image" /></td>
<td>31 53</td>
<td>-2 -2</td>
</tr>
<tr>
<td><img src="image5" alt="Pixel configuration image" /></td>
<td>10 11</td>
<td>0 1</td>
</tr>
<tr>
<td><img src="image6" alt="Pixel configuration image" /></td>
<td>13 12</td>
<td>+2 +1</td>
</tr>
</tbody>
</table>

Table 5-5: Chain code conversion
Figure 5-6: Error comparison for crack coded circles

Figure 5-7: Error comparison for chain coded circles
Dinkelberger's method and Freeman's algorithm produced identical results with around 6% deviation error for square pixels and 6.5% for rectangular pixels (Figure 5-6 and 5-7). My own algorithm achieved a constant error value of ca. 3% for circle diameters larger than 100 pixels. Naturally, the average method by Horn gave the best results for circles with below 1% for square and rectangular pixel shapes (see also Appendix B).

The method by Dessimoz proved very reliable with deviation error values around the 2.5% mark. My own method achieved an average value of about 1.5% for chain coded circles (Figures 5-7).

5.3.3.2 Semi-circles.

Semi-circles are an interesting shape for the examination of outline length because the silhouette combines the two basic shape primitives: straight line and circular arc. The error contribution of these basic elements are quite distinct. While the error contribution of straight line segment is orientation dependent, the error contribution of the circular arc segment is both curvature and length dependent.

The combination of these two properties is shown by the fact that rotational variation results in a deviation in the error function and size/resolution variation manifests itself in a constant error contribution which can be regarded as an offset value for the error function.

In fact, both shape primitives, arc segment and straight line, are represented by their error contributions in proportion to their share of the total curve. For the averaging method of Horn, the offset is neglectable because the average error for a circle is less than 1% (see results above). However, the straight line content which amounts to 38.9% of the overall length of the outline is responsible for the huge overall error range. The error range for the crack code is scaled down from the theoretical maximum value of 41.4% to around 13% (-8% to +5%). The negative range of errors is due to the offset by the average value. The Dunkelberger method which is practically an 8 direction code shows the equivalent curve form which is again scaled down according to the length proportion of the straight line from the total outline. Similarly, the offset value is determined by the fraction of the curve from the overall shape. Naturally, the scaling and offset properties apply to all other methods as well.

Semi-circles were generated with a constant size at different orientations from 0 to 90 degrees in 0.5 degree steps. The results of this experiment verify, in fact, the above considerations (see also Chapter 4).

Figure 5-8 shows very clearly a distribution similar to that for the straight line approximation. The asymmetric distribution (here with a minimum at 36.9 degrees) is characteristic of a rectangular aspect ratio (here 4:3). This minimum is offset from zero by the proportional error value of the circular arc fraction which amounts to 6.1%. The average error value for
Figure 5-8: Error comparison for crack coded semicircle

Figure 5-9: Error comparison for chain coded semicircle
The accuracy of the crack coding method (Figure 5-8) does not change for a different aspect ratio because only the base vector directions are available. Again, note the similarity between Dunkelberger's method (Figure 5-8) and the Freeman codes (Figure 5-9).

Figure 5-9 shows the result of techniques which rely on an eight direction chain code. Clearly shown is the asymmetrical distribution for a 4:3 aspect ratio. Zero offsets and scaling effects are also distinct.

5.3.3.3 Squares and triangles

The perimeter estimation of simple polygons such as the square, the rectangle, and the triangle can be simplified by considering their straight line segments separately (Chapter 4). Since the error function in a square pixel environment is symmetrical to $\pi/4$ the error contribution of each side of a orthogonal polygon (square or rectangle) is equal. In other words the length estimation of the contour of a square/rectangular silhouette follows exactly the form of the error function.

![Triangular Shape Orientation Variation](image)

Figure 5-10: Error comparison for crack coded triangles
Figure 5-11: Error comparison for chain coded triangles

Figure 5-12: Error comparison for crack coded squares
Similar results are achieved for a triangle, except that the symmetry is $\pi/6$ or 30 degrees (Figure 3-10 and Figure 3-11). Figure 5-12 and Figure 5-13 show the results for squares using the methods examined for crack codes and chain codes. The magnitude of the error function is here on a 1:1 scale since no circular arc is part of the shape.

Note for the Horn method the shape of the function has not changed due to the $\pi/4$ symmetry of the error function for crack codes.
5.4 Conclusion

Summarizing results are shown in Figure 5-14 as error ranges taken from the maximum, minimum and average deviation value from a set of generated silhouettes.

![Perimeter Deviation for 4 basic shapes](image)

**Figure 5-14:** Results of methods examined

From Figure 5-14 it can be seen that the Friedrich codes are the most accurate in each of the two categories. The method described by Horn only works well for the circular shape, but the deviation for squares is unacceptably high. The results of the Dunkelberger method are very similar to the Freeman coding. The approach by Dessimoz shows slightly better results than the crack code corrections by Friedrich. However, the chain code correction method also by Friedrich is the most accurate with error averages below 2%.

In this chapter, the practical application of the error function for the length estimation of digitized outlines has been explored. The convolution theory (Chapter 4) was verified by using computer generated silhouettes. While this property mainly has academic value, the important result for a practical application is that simple code correction methods, as introduced here, are quite adequate to significantly improve deviation errors.
6.1 Introduction

The ultimate goal of recognizing partially visible shapes relies on a suitable shape description. The length of object sides and their enclosed angles are essential and sufficient for shape recognition [Davis 1977]. Boundary segments in the form of a polygon contain exactly these two parameters, namely segment length and orientation. A polygon description of a silhouette therefore provides a suitable representation for recognition. A silhouette polygon may best be described by a series of vectors, comprising length and angle values. Perkins [1978] presents such a procedure where objects are matched in a slope angle domain [Ballard 1982]. This approach has since been a popular technique for recognizing silhouette parts with a significant feature [Koch 1985, Knoll 1985, Turney 1985].

Many techniques for polygonal approximations of line drawings or plane curves exist [e.g. Ramer 1972, Pavlidis 1982]. However, the majority of line fitting algorithms are ill-suited for this purpose because they were designed for a different purpose. The resulting polygon is often contiguous, unless so-called end-point fitting methods are used. Commonly, a straight line, formed by two boundary pixels, is divided into new segments if the maximum distance between the straight line and the original outline exceeds a pre-selected threshold. The subsets are split and merged to achieve a better fit [Pavlidis 1974]. Split and merge methods produce good results but are time consuming since multiple passes are required. However, in machine vision applications, where cycle time is critical, very fast and hence simple algorithms are needed. Simplified end-point fitting methods are a suitable approach for the required fast polygon approximation. One such fast polygonal shape approximation is the arc operator.

The selection of break-points is the most critical part in terms of processing speed, consistency and reliability. Segmentation by tracing for example produces either an array of positions of boundary pixels or a series of relative direction codes such as chain-codes or crack codes, which can be grouped together to achieve a more economical description [Laing et al. 1977]. However, this approach is too restricted in that repetitive code sequences are combined into "long runs" which comprise only standard vector directions. A shape description based on the common 8 direction chain code is not directly suited to recognition purposes for two reasons. Firstly, the limitation to only eight different vector orientations does not give enough angular information. Secondly, a very long string in the order of several hundred code elements has to be compared with a number of reference code strings in order to recognize the boundary.
A small position offset of the object may change most of the code elements. Also a slight rotation of the object causes major code changes. Chain codes are therefore ill-suited for recognition due to the extreme sensitivity to boundary variation from quantization noise [e.g. Horn 1986], shadow effects and illumination variation.

The new algorithms presented in Chapter 5 combines various chain code sets to estimate the boundary length. This method has shown a significant improvement over existing methods. However, this method provides a very limited number of angles. Although the outcome is a reduced number of describing polygon vectors, these are still too numerous and noise sensitive and therefore not suitable for rapid shape recognition. A more precise description of angles and sides is required.

Chaudhuri [1984] combines chain code sequences in a new form of coding which produces a short form of the chain code description. For the purpose of recognition the major drawback of this method is the provision of too few angular parameters.

A different approach also designed for code compression [Landy 1985] combines chain code sequences into segments which are then coded with a variable length code. These new segments are in effect an acceptable approximation. However, this form of boundary description is only a short form of the chain code description with far too many vertices for rapid recognition.

An appropriate measure of efficiency is the compression factor, which relates the number of vertices to the number of chain code elements. This compression factor is used in this chapter to compare the arc operator with various other approximation methods [Teh 1988].

The error analysis has shown deficiencies of the length measurement of chain coded lines and silhouettes. Since segment length and angle are crucial for shape recognition, a versatile algorithm is needed. In this chapter a new algorithm named the arc operator is described. The arc operator features efficient processing and an acceptable accuracy. The segment length of the generated vector description may be calibrated to the actual segment length of the original boundary by using the technique described in Chapter 5.

6.2 A New Approach to Shape Analysis

The new algorithm is based on a straight line connection of two boundary pixels. The maximum perpendicular distance of this straight line to the original outline can be used as a measure of curvature. This property is successfully used to determine the most distant border pixel from this "short cut". The boundary location of this pixel presents a new "break point" for polygon approximation schemes if the distance value lies above the pre-set threshold [Pavlidis 1982].

Now consider a short straight line segment moving along the boundary where both end points are part of the boundary. This principle can be used to detect points of high curvature [Batchelor 1985]. The distance values obtained from such a procedure are a measure of local curvature as a function of the boundary location and may be used as an indicator of sharp corners by using an appropriate "corner" threshold.
6.2.1 Definition of the algorithm

For real time applications all calculations, in particular division and multiplications using floating point instructions, should be kept to a minimum. The arc operator is therefore a simplified procedure. To illustrate this best an analogue outline is assumed first. Simplified calculations can then be derived more easily.

Figure 6-1: Principle of the Arc operator

Figure 6-1 shows the geometric relations between the boundary points A, C, and B. The distance \( d \) is measured between the centre of the line AB and the boundary. Both other sides of the triangle \( ABC \), namely \( k \), are equal in length because the distance vector \( d \) is perpendicular to \( AB \) and located in the centre of the connection \( AB \). \( ABC \) is therefore an isosceles triangle with an orientation of \( \beta \). The angle \( \alpha \) in the triangle \( ADC \) corresponds to the curvature of the outline. An estimation of the curvature relies on an appropriate selection of \( k \), also called the region of support or the smoothing factor.

The distance \( d \) is clearly represented as

\[
d = k \sin \alpha \quad \text{where} \quad k = AC = CB \quad \text{or}
\]

\[
d = \sin \alpha \cdot \sqrt{(L_{1x}^2 + L_{1y}^2)} \quad \text{for direct computer implementation}
\]

\[
d = \{ \arctan \left( \frac{L_{1y}}{L_{1x}} \right) - \arctan \left[ \left( \frac{L_{1y} + L_{2y}}{L_{1x} + L_{2x}} \right) \right] \} \cdot \sqrt{(L_{1x}^2 + L_{1y}^2)} \quad (6.1)
\]

with the \( x \) and \( y \) components \( d = dx + dy \)
The length projections of $k$ onto the $y$-axis yield

$$L_{1y} = k \sin(\alpha + \beta) \quad \text{and} \quad L_{2y} = k \sin(\beta - \alpha) \quad (6.2)$$

$$c \sin \beta = L_{1y} + L_{2y} \quad \text{with} \quad c = AB \quad (6.3)$$

From triangle $BDF$

$$c/2 \sin \beta = d \cos \beta + k \sin(\beta - \alpha) \quad (6.4)$$

substituting (6.2) and (6.3) the $y$ component of the distance becomes

$$d_y = d \cos \beta = 1/2 (L_{1y} + L_{2y}) - k \sin(\beta - \alpha) \quad \text{or}$$

$$d \cos \beta = 1/2 (L_{1y} - L_{2y}) \quad (6.5)$$

The sign of the distance vector $d$ indicates whether the arc is convex or concave. For $d > 0$ ($\alpha > 0$) in equation 6.5 the arc is concave given the sequence $ACB$, otherwise the outline is convex ($d < 0$ and $\alpha < 0$) at this point.

Similarly, the projection of $d$ to the x axis yields

$$L_{1x} = k \cos(\alpha + \beta) \quad \text{and} \quad L_{2x} = k \cos(\beta - \alpha) \quad \text{where}$$

$$dx = d \sin \beta = 1/2 k \left[ \cos(\beta - \alpha) - \cos(\beta + \alpha) \right] \quad (6.6)$$

The exact distance $d$ is then

$$d = \sqrt{(dx^2 + dy^2)}$$

simplified by the following approximation

$$d = |dx + dy| \quad (6.7)$$

The error made by this approximation, namely $(\sin \alpha + \cos \beta)$ is identical to the error function for crack codes (Chapter 3). The approximation value overestimates pixel equivalent distances in diagonal directions. However, the maximum deviation of $+41.4\%$ at $\pi/4$ only results in additional break-points by triggering a preset distance threshold too early. These extra break-points are not significant and are grossly outweighed by a massive saving in processing time. It can be shown that a few extra break-points are quite acceptable in view of the processing time gained as discussed below.

In order to achieve an optimum number of break-points, more sophisticated and time consuming computation has to be done. Teh et al [1988] compare six methods with their own. Computing times measured on an Apple MacIntosh are in the order of 10 seconds which is not acceptable for such a small shape as presented in their publication.

6.2.2 Distance function

In digital image processing the coordinates of a particular point are generally known or can be obtained easily, so that, in fact, $dx$ and $dy$ are simple position differences. Let $A$, $C$, and $B$ be the sequential points on the object border at the location $i$ in a code sequence
describing the silhouette and let the coordinate values of point $C$ be $x(i)$ and $y(i)$. Points $A$ and $B$ are $k$ pixels away from $C$ in each direction so that $A$ is located -$k$ pixels from $C$ and $B$ is located +$k$ pixels from $C$. Then equations 6.5 and 6.6 become

$$2 L_{1x} = x(i) - x(i-k) \quad \text{and} \quad 2 L_{2x} = x(i + k) - x(i) \quad \text{thus}$$

$$2 d_x(i) = | x(i) - x(i-k) - [x(i+k) - x(i)] | =$$

$$= | 2x(i) - x(i-k) - x(i+k) | \quad (6.8a)$$

$$2 d_y(i) = | y(i) - y(i-k) - [y(i+k) - y(i)] | =$$

$$= | 2y(i) - y(i-k) - y(i+k) | \quad (6.8b)$$

and the approximated distance $d$ yields

$$2 d = 2 dx + 2 dy =$$

$$= 2x(i) + 2y(i) - x(i - k) - y(i - k) - x(i + k) - y(i + k) \quad (6.9)$$

By keeping twice the distance value and doubling the threshold instead, further simplification is achieved. Since the distance value at the boundary location $i-k$ has already been calculated as part of the iteration $k$ steps before, equation 6.9 becomes

$$d(i) = |x(i) + y(i) - x(i + k) - y(i + k)| - d(i - k) \quad (6.10)$$

Corner detection

Figure 6-2 shows first results of such an operator for a metal stamping with sharp corners. The distance or arc function can be directly used as an indicator of a corner. An appropriate threshold with a peak detector can accurately locate all corners of this object. Distance deviation errors of more than 40% (equation 6.7) are still acceptable because $d$ is used as a corner detector and since errors of $d$ are always positive the corner detector is triggered earlier, producing an additional break point. The reconstructed outline is shown in Figure 6-3. Although this simple method is adequate for corner detection, it fails when used on curves.

6.2.3 Running distance total

Low curvature parts or circular arcs cannot be detected by this method because the distance function is constant over such a segment. One solution to this problem is to integrate the distance function. When the sum exceeds a "curve" threshold a new break point is registered. Subsequent integration has to commence again. This principle was applied to a curved object, here a Kiwifruit.

The main reason for selecting this shape is that it is ideally suited for this particular case having a low curvature along the entire boundary. Furthermore, as a natural shape there are deviations built-in for testing the robustness of the arc operator. The second reason was that a compact description was sought for the Kiwifruit shape parameter. Figure 6-4 shows the result of this procedure, the distance function over the main part of the boundary length. Figure 6-5 shows the reconstructed outline.
Figure 6-2: Corner detection with the Arc operator

Figure 6-3: Reconstructed outline of a metal stamping
Figure 6-4: Curve detection using a Kiwi fruit silhouette.

Figure 6-5: Polygon approximation of a Kiwi fruit.
6.3 Parameter Selection

In order to achieve reliable results the parameters of the arc operator have to be carefully selected depending on object size and expected shape properties.

6.3.1 Segment length

Obviously, the segment length $k$ of the operator directly influences the resolution of the approximation. The part of the boundary considered is often referred to as region of support or smoothing factor. A large region of support will smooth out fine details. Accordingly, a small region of support produces a finer distribution of break-points. As shown in Figure 6-6 a too large region of support may miss important details, here on both sides of the handle and the centre pin.

![Large region of support](image)

Figure 6-6: Influence of a large region of support.

6.3.2 Perpendicular distance

Another important parameter is the distance $d_t$ threshold, the deviation allowance of the approximation. The smaller the distance "quota", the more vertices are generated. Too large a "quota" on the other hand results in too coarse an approximation. Generally this factor determines rapid directional changes, such as corners. An approximation of the perpendicular distance from the arc operator segment to the boundary is calculated using equation 6.10. Figure 6-7 illustrates the point, that distance "scanning" is best at high
curvature parts but very poor in low curvature parts of the boundary. This property is mainly due to the fixed region of support. A corner breakpoint is found if the current distance $d(i)$ exceeds the preset value $d_t$.

$$\text{Corner, if } d(i) > d_t$$

(6.11)

![Distance threshold](image)

**Figure 6-7:** Simple distance threshold.

### 6.3.3 Integrated distance

The accumulated distance $d_a$ is a measure of low curvature segments. The value selection therefore depends on the shape properties, for example whether shape segments are mainly curved or straight. Low threshold values are very sensitive to slight boundary changes and too large values tend to miss out the low end of curvatures. The important concept is that the integration procedure has to compensate for direction changes. In other words the sign of the distance value has to be taken into account as follows:

$$\text{Curve, if } \sum_{m=n}^{i} d(m) > d_a$$

(6.12)

with $n$ being the boundary position previously accepted as dominant point and $i$ the current position along the boundary. The most significant property here is the ability to discount slight variations of the outline and only define a curve point if the sign of the curvature does not change before the preset accumulated value is reached. Thus, a small "kink" in the outline will not necessarily result in a break-point. **Figure 6-8** illustrates this around the "neck" of the seat belt buckle near the lower part of the shape. It also illustrates that this parameter essentially covers the medium range of curvature.
6.3.4 Segment deviation

The third criteria, the segment deviation, is mainly intended for shapes with a variety of low and high curvature parts combined with sharp corners. In order to care for these circumstances an evaluating segment spanning from the last accepted dominant point to the current point is used to ensure the approximating segment will not deviate more than specified. However, this measure is only useful in combination with the other parameters above. Figure 6-9 shows that the deviation parameter used on its own provides reasonable approximation with the major drawback being a loss of accuracy. The breakpoints are chosen arbitrarily when the segment distance exceeds a threshold. Corner positioning is very poor if this parameter is used on its own.

The location of a corner depends on the "history" of the boundary curvature. For example, with a sharp corner located after a long straight line segment, the deviation threshold will be triggered past its actual position. The reason for this is that the distance from the centre of this growing segment to the digitized boundary will rise very slowly. However, good results can be achieved by combining the above criteria.

Figure 6-10 shows the successful polygon approximation with the arc operator applied to a seat belt buckle. The parameters settings are: operator length \( k = 8 \) pixels, the distance threshold: \( d_t = 15 \) (around 4 pixels) and the accumulated distance tolerance: \( d_a = 90 \) (around 12 pixels).
Segment distance

- Pixel size

- Original — Arc operator

Figure 6-9: Segment distance.

Combined parameters

- Pixel size

- Original — Arc operator

Figure 6-10: Polygon approximation for a seat belt buckle.
6.4 Implementation Considerations

Integer calculations favour enormous advantages in processing time. The implementation of the arc operator can be further simplified by splitting equation 6.10 into two terms, an x-part and a y-part, each with only one subtraction. Since the operator is moving along the object boundary, only the first part of equation 6.10 needs to be calculated at a time. The second term has been calculated already \( k \) steps before. Figure 6-11 shows the three arrays in columns. The first contains raw position data of boundary pixels, the second contains their position difference, and the third shows the distance values.

\[
\begin{align*}
\text{Position coordinates} & \quad \text{position differences} & \quad \text{Distance values} \\
x(i+k) & \rightarrow dx(i-k) & \rightarrow d(i) \\
x(i) & \rightarrow & \\
\end{align*}
\]

Figure 6-11: Simplified distance measure

Column 2 is the result of

\[
d_x(i) = x(i+k) - x(i) \quad \text{and} \quad d_y(i) = y(i+k) - y(i)
\]

(6.13)

and column 3 represents

\[
d_{2x}(i) = dx(i) - dx(i-k) \quad \text{and} \quad d_{2y}(i) = dy(i) - dy(i-k)
\]

(6.14)

Quite an important factor here is that only the previous \( k \) values have to be stored for this simplified procedure. In practice, it could be considered similar to a shift register \( k \) elements long. At the buffer exit an element delayed by \( k \) pulses is available which can be
added or subtracted to the current element from the buffer entrance. Repeating this technique twice for x and y values, equation 6.12 can be realized. Figure 6-12 shows complete details for both branches, x and y.

\[ \text{Arc Operator – Implementation} \]

\[
\begin{align*}
X_{pos} & \quad i \quad i+k \quad \Delta X \\
\text{Y}_{pos} & \quad i \quad i+k \\
\Delta Y & \quad i-k \\
\end{align*}
\]

Figure 6-12: Possible hardware implementation of the Arc operator.

Equation 6.14 replaces in effect equation 6.10. The key to this algorithm is a peak finder based on a derivative approach. Changes in the boundary positions caused by a rapid slope change show a peak at its fastest change in the second derivative (simplified by digitization using subtraction of discrete boundary elements). If an additional differentiation is calculated (third derivative), the peak transforms into a positive and a negative part, and the zero crossing between the two, pin-points the exact location of the peak. Figure 6-13 shows an extract of an ideal boundary with a sharp corner illustrating these properties.

In practice, the procedure is naturally very noise sensitive. Effects from the illumination and quantization noise of both the image digitization process and the integer calculations are major contributors to some instability. Furthermore, integer implementation introduces more vertex points due to the direction sensitivity of the simple mid-point distance approximation. Redundant points may easily be identified by either thresholding to a minimum segment length and/or to a minimum angular change between subsequent vectors provided a deviation check at the possible redundant break-point is calculated. These measures require a second pass on the boundary.
Figure 6-13: Arc operator parameters on a sharp corner sample.

An easier approach to produce an acceptable approximation, is to introduce priorities. Corner detection must have priority over curve detection. In other words, a break-point will be located preferably at locations with a peak distance value where a real corner is situated. If the "curve" threshold was exceeded before the distance peak occurred, then the curve point is ignored. In practice, exceeding the "curve" threshold will create a break-point, but this point may be dismissed if peak distance follows within a short distance. A sensible minimum boundary segment is half the length \( k \) of the arc operator. If the integrated distance has passed the critical value and the distance value has not passed its threshold, then a curve point is declared. This point may be withdrawn later by the distance value exceeding the "corner" threshold within a segment length of \( k \) pixels. If, on the other hand the curve threshold is above its limit value and the corner detection has also passed its threshold, then the distance function is examined on its maximum value.

This procedure ensures a preference of corner points over curve points and locates sharp corners as accurately as possible. Whereby a good synchronization of corners between the polygon description and the original outline is achieved. This means that the location of critical corner points is independent of the starting point along the boundary and also independent of the orientation of the object. Combining both methods yields a fast algorithm to generate silhouette vectors.

Figure 6-14 shows part of a boundary with a competitive break-point reporting procedure with the occurrence of such a point dismissal. The implementation of the above new feature yields a competitive reporting of break-points.
In cases where break-points are reported too closely to each other, a correction measure has to be applied. If a corner point is reported shortly after a curve or deviation point, the last corner position detection will be discounted.

![Arc Operator Curve / corner detection](image)

**Figure 6-14:** Competitive break-point reporting

### 6.5 Experimental Results

The break-points or dominant points on a digitized boundary or part thereof, can be used as a compact and effective representation of the object for shape analysis and recognition. The approximating polygon is formed by joining the vertices. The quality of fit may be measured quantitatively by a point-wise error between the digitized outline and the approximating polygon. Two error measurements are used here. Firstly, the maximum perpendicular deviation error $E_{\text{max}}$ between a boundary point and the closest polygon segment.

The maximum error is then defined as:

$$E_{\text{max}} = \max e_i \quad \text{with} \quad 1 \leq i \leq m \quad \text{(number of boundary points)} \quad (6.15)$$

The second measure is defined as the integral square error $E_{\text{int}}$ as follows:

$$E_{\text{int}} = \sum_{i=1}^{m} e_i^2$$

with $e_i$ being the perpendicular distance from the polygon segment to the boundary at point $i$. 

The arc operator and the split and merge algorithm are compared with six other algorithms from [Teh 1988]. In order to provide a straightforward comparison the same criteria were used. These comprise

a) the number of dominant points
b) the compression factor
c) the maximum deviation error
d) the integral square error
e) the total CPU processing time

\[ n \]
\[ m/n \]
\[ E_{\text{max}} \]
\[ E_{\text{int}} \]
\[ t \text{ in sec} \]

These measures are compared to six different methods reviewed briefly in chapter 2. The shape taken from Teh [1988] for a direct performance comparison consists of three semi-circles with different radii and a total of 102 boundary points. The shape represents a typical example of a curve possessing a feature of different sizes.

**Figure 6-15:** Split and merge method [Pavlidis 1982]

For comparison the arc operator was implemented in two versions, one calculating the deviation with floating point variables (equation 6.1) and the other with purely integer calculations (equation 6.10). **Figure 6-15** shows an approximation using the split and merge method. **Figure 6-16** illustrates the approximation with the arc operator with around a third of the processing time. Finally, **Figure 6-17** shows the result of an integer implementation of the arc operator with an enormous advantage in processing time. **Table 6-1** presents the experimental results in detailed form including the results from Teh [1988]. As can be seen from this comparison the integer implementation of the arc operator is by far the fastest method. The overall accuracy of other methods is only marginal better.
Figure 6-16: Arc operator with floating point calculations.

Figure 6-17: Arc operator with integer calculations.
The compression factor represents the number of boundary pixels $m$ over the number of vertices $n$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>no. points</th>
<th>compression factor</th>
<th>Max error (pixels)</th>
<th>Integral error (pixels $^2$)</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenfeld - Johnston</td>
<td>30</td>
<td>3.4</td>
<td>0.7</td>
<td>8.8</td>
<td>8.09</td>
</tr>
<tr>
<td>Rosenfeld - Weszka</td>
<td>34</td>
<td>3.0</td>
<td>1.0</td>
<td>15.4</td>
<td>10.13</td>
</tr>
<tr>
<td>Freeman - Davis</td>
<td>19</td>
<td>5.4</td>
<td>1.4</td>
<td>23.3</td>
<td>9.46</td>
</tr>
<tr>
<td>Sankar - Sharma</td>
<td>10</td>
<td>10.2</td>
<td>8.0</td>
<td>769.5</td>
<td>35.04</td>
</tr>
<tr>
<td>Split and merge</td>
<td>23</td>
<td>4.4</td>
<td>1.2</td>
<td>122.3</td>
<td>4.26*</td>
</tr>
<tr>
<td>Pavlidis</td>
<td>18</td>
<td>5.7</td>
<td>1.64</td>
<td>36.14</td>
<td>12.85</td>
</tr>
<tr>
<td>Anderson - Bezdek</td>
<td>22</td>
<td>4.6</td>
<td>1.00</td>
<td>20.61</td>
<td>9.40</td>
</tr>
<tr>
<td>Teh - Chin</td>
<td>15</td>
<td>6.8</td>
<td>1.8</td>
<td>238.2</td>
<td>1.80*</td>
</tr>
<tr>
<td>Arc operator - floating point</td>
<td>10</td>
<td>10.2</td>
<td>2.3</td>
<td>367.6</td>
<td>0.07*</td>
</tr>
<tr>
<td>Arc operator - integer calculation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* adjusted CPU times.

Table 6-1: Performance comparison of the arc operator with various other algorithms reviewed by [Teh 1988].

The processing times of Teh and Chin refer to an Apple Macintosh II (Motorola 68000 microprocessor) with a similar processing power of an IBM PC AT (Intel 80286 processor). The Pavlidis algorithm and the arc operator have been tested using a 80386 SX machine without a co-processor. The processing times are about 1.5 times faster compared to the AT or the apple Macintosh.

However, one has to bear in mind that the chosen shape is relatively small in a common 256 by 256 or even 512 by 512 pixel image. All shape examples in this work are based on an image with 256 by 256 rectangular pixels. Shapes not occupying a large portion of this size frame are enlarged which can be seen from the pixel size indicated in most figures. The boundary length of a frame filling object is typically in the order of 500 to 800 pixels compared to only 102 in the above semi-circular shape. Consequently, the processing times published by Teh would have to be multiplied by a factor of 5 to 8 to be able to compare processing times of other shapes used in this chapter.

The following set of figures shows more examples of shape approximation using a circular metal stamping with two opposite tags. Figure 6-18 illustrates the split and merge algorithm, Figure 6-19 a shape approximation using the (integer based) arc operator. Similarly, Figure 6-20 and Figure 6-21 illustrate the two approximation schemes applied to the silhouette of a screw terminal, and Figure 6-22 and Figure 6-23 show the algorithms applied to the chuck key.
Split and merge method

Figure 6-18: Split and merge approximation (circular stamping).

Arc operator (Integer)

Figure 6-19: Arc operator approximation (circular stamping).
Figure 6-20: Split and merge approximation (screw terminal).

Figure 6-21: Arc operator scheme applied to a screw terminal
Figure 6-22: Split and merge approximation (chuck key).

Figure 6-23: Arc operator approximation (chuck key).
Finally, in Table 6-2 detailed results are summarized for the six different shapes shown in previous figures.

<table>
<thead>
<tr>
<th>Shape</th>
<th>no. points</th>
<th>compression factor</th>
<th>Max error (pixels)</th>
<th>Integral error (pixels²)</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Switch cover plate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>18</td>
<td>32.3</td>
<td>2.9</td>
<td>395</td>
<td>3.96</td>
</tr>
<tr>
<td>Arc operator</td>
<td>23</td>
<td>25.3</td>
<td>1.9</td>
<td>253</td>
<td>0.05</td>
</tr>
<tr>
<td>2) Kiwifruit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>19</td>
<td>31.2</td>
<td>2.8</td>
<td>655</td>
<td>4.55</td>
</tr>
<tr>
<td>Arc operator</td>
<td>32</td>
<td>18.1</td>
<td>3.1</td>
<td>316</td>
<td>0.06</td>
</tr>
<tr>
<td>3) Seat belt buckle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>25</td>
<td>24.4</td>
<td>2.7</td>
<td>312</td>
<td>5.49</td>
</tr>
<tr>
<td>Arc operator</td>
<td>28</td>
<td>21.8</td>
<td>2.7</td>
<td>384</td>
<td>0.06</td>
</tr>
<tr>
<td>4) Metal stamping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>16</td>
<td>16.3</td>
<td>2.5</td>
<td>184.9</td>
<td>1.98</td>
</tr>
<tr>
<td>Arc operator</td>
<td>18</td>
<td>14.4</td>
<td>1.6</td>
<td>113</td>
<td>0.05</td>
</tr>
<tr>
<td>5) Screw terminal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>20</td>
<td>13.4</td>
<td>3.0</td>
<td>185</td>
<td>2.09</td>
</tr>
<tr>
<td>Arc operator</td>
<td>20</td>
<td>13.4</td>
<td>2.4</td>
<td>92</td>
<td>0.05</td>
</tr>
<tr>
<td>6) Chuck key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split and merge</td>
<td>24</td>
<td>15.4</td>
<td>2.6</td>
<td>179</td>
<td>3.29</td>
</tr>
<tr>
<td>Arc operator</td>
<td>23</td>
<td>16.0</td>
<td>3.6</td>
<td>170</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 6-2: Performance comparison of the arc operator and the split and merge algorithm [Pavlidis 1982].

Provided the parameters of the arc operator are selected carefully to meet the shape properties the implementation of the complete algorithm resulted in a fast and satisfactory polygon approximation.

The algorithm limitations are basically twofold. One, the segment resolution cannot be finer than the length $k$ of the leading and trailing segment of the arc operator. Two, the accuracy of the distance calculation is direction sensitive through the approximation scheme used. Particularly, diagonal directions are over-estimated and result in additional redundant break-points. These apparent shortcomings can be reduced by filtering the resulting vector for minimum segment length and angle. Recognizing objects can be further simplified if the shape matching is performed in the slope angle domain [Ballard 1982].

6.6 The Slope Angle Domain Application

The template and object boundaries may be represented in two domains, in the original cartesian space and in an arc-angle space also called "slope angle - arc length space". Matching of boundary segments is more efficient in the arc-angle space, essentially because of three major properties of this representation:
1) Angle changes and side length ratios are constant and independent of object size and orientation, the initial requirements for shape recognition.

2) Rotation in cartesian space translates simply into an angle offset in the arc-angle domain, since the angle of rotation is added to each individual segment or side angle.

3) Size in cartesian coordinates translates into a zooming factor for the length of segments, whereby angles are constant. In other words, the length of a boundary segment is proportional to the size and therefore to the distance of the object.

Recognition of an object is successful if the segment-to-side ratios of the boundary and the angular changes from one segment to the next are equal. A matching operation therefore produces an "angle of match", which is exactly the orientation angle of the object relative to the reference. Thus, the rotation of the object is a by-product of the recognition procedure. Another "by-product" of object recognition in the slope-angle domain is object size relative to the reference shape. For instance, from the actual segment length the object size can be calculated provided the distance to the camera is known and constant. Alternatively, an object can be recognized at various distances. Provided the size of the object is constant and known the relative distance to the camera can be extracted.

Once the visual sensor is calibrated, (for example by showing a reference object, at a given distance to the system) the actual object distance can be determined by the perimeter ratio or average ratio of segment length from both reference and current silhouette. Using the information of object type, location, orientation and distance, a handling device such as a robot can then be instructed to grip the object successfully.

6.6.1 Feature recognition

Straight line features are illustrated quite clearly in Figure 6-24 representing the switch cover from Figure 6-3. This shape consists only of straight line segments which correspond to horizontal lines in the arc-angle domain. In contrast to Figure 6-25 which represents the shape shown in Figure 6-17 consisting entirely of semi-circular arcs. These segments translate into straight lines at various degrees of angles according to their curvature. High curvature parts relate to rapid slope angle changes and steep segments, in contrast to low curvature parts which translate into horizontal straight lines and "flat" segments.

Combining these effects, Figure 6-26 represents the seat belt buckle from Figure 6-10. It can be seen that the rapid declining part of the curve around the 500 to 550 pixel mark corresponds to the high curvature part at the rounded end of the shape. The flat part between the 200 and 300 pixel mark corresponds to the low curvature part at the top of the shape. In other words sudden angular changes of the function indicate sharp corners, slow changes indicate circular arcs while horizontal lines correspond to straight line boundary segments.
Figure 6-24: Slope angle diagram of the switch cover with straight outlines.

Figure 6-25: Semi-circular object [Teh 1988].
6.6.2 Slope angle approximation

From these approximations it is obvious that the arc operator can also be applied in the slope angle domain producing straight line segments for further approximation of the boundary. These segments correspond not to the actual shape directly, but to circular arc segments and straight lines. The curvature of such a circular arc segment corresponds to the slope angle of the straight line in the slope angle domain.

Consider, for example, the semi-circular shape and its slope angle diagram illustrated in Figure 6-25. Instead of fitting straight lines to the actual object outline, circular arcs can be fitted. This can be achieved by fitting straight lines in the slope angle domain. The result is a more compact and accurate shape description. Figure 6-27 shows a successful approximation of such a procedure for the semi-circular shape. Figure 6-28 illustrates the results of applying the same strategy to the circular stamping. Note the sudden changes from the sloping part to a straight segment corresponding to the "tags" of the chosen shape. This approach results in a very economic shape description consisting of only two shape primitives - straight lines and circular arcs. The area of shape description using shape primitives which appears to be a useful and potential application area for the arc operator.

These significant object features can be detected and recognized locally independent of a complete boundary description. This mechanism provides therefore an ideal base for recognizing overlapping parts.

Figure 6-26: Slope angle representation of seat belt buckle.
Figure 6-27: Approximating the slope angle profile (semi-circular shape).

Figure 6-28: Approximating the slope angle profile (Circular stamping).
6.6.3 Overlapping objects

Local features may be extracted and objects recognized if significant parts are visible. **Figure 6-29** illustrates an example of such a configuration consisting of a screw-terminal for electrical connections, a chuck key and a circular stamping with tags on opposite sides. Characterizing features of the individual parts are visible for recognition and correspond to appropriate sections of their slope angle representation shown in **Figure 6-30**. The main part of the circular object including one of its tags is visible. In the slope angle diagram these features should be obvious: a constant sloping part for the circular arc and short sudden slope changes of about 90 degrees (compare to **Figure 6-28** and **Figure 6-31**). From the chuck key the characterizing long and thin handle is visible with its very short and sudden changes of 90 degrees and long horizontal parts in the slope angle domain (**Figure 6-32**). The silhouette of the screw terminal consists of a combination 45 and 90 degree change with different segment lengths (**Figure 6-33**).

**Figures 6-34** shows the results of a manual shape matching in the slope angle domain using a graphic correlation procedure. Note the similarity between the reference diagrams which have been approximated using the arc operator. Even with such a crude approximation the shape relation is shown quite clear. The individual slope angle diagrams are shifted to the matching position.

![Overlapping parts](image)

* Pixel size

---

**Figure 6-29**: Overlapping objects
6.28

Figure 6-30: Slope diagram of overlapping objects

Figure 6-31: Slope diagram of circular stamping (conventional approximation)
Figure 6-32: Approximated slope angle diagram of chuck key.

Figure 6-33: Approximated slope angle diagram of screw terminal.
6.7 Concluding Remarks

In this chapter a simple and effective operator is introduced capable of producing acceptable straight line approximations of given boundaries. With careful selection of parameters it can be applied to arbitrary curves. The operator may also be applied to the slope angle representation of the silhouette and produce circular arc segments as a shape representation, which can be useful for shape recognition. The major advantage of the arc operator is processing time particularly in its integer implementation. Performance comparisons made with six other algorithms has shown an enormous advantage in processing time in the order of one magnitude.

Despite some deficiencies in accuracy, the shape approximation can be successfully used in recognition tasks whereby the arc-angle domain approach provides the best potential. An example of overlapping parts with characteristic features of each part visible is illustrated and discussed. The area of local feature recognition shows the most promising application area of the arc operator. In fact, the operator is currently trialed as part of an expert system for production automation [Malone and Friedrich 1990].
CHAPTER SEVEN

7. CONCLUSION AND FUTURE WORK

This work investigates algorithms for fast image processing in the area of machine vision. A number of problems in the area of shape description are identified, analysed and in each case a solution is suggested. Machine vision is a relatively new field derived from established areas of image processing. Its requirements are different in terms of processing speed and therefore simplicity. A survey of shape description algorithms is presented in Chapter 2.

In order to provide the facilities required for this work, a complete image processing package was produced. The main features of this package are outlined in Chapter 2 and two applications are described. One is in the area of optical measurements of fruit shape parameters. Although the arc operator may be applied successfully here in order to compress the shape information, it was decided for compatibility reasons to describe the fruit silhouettes using the common 8 direction code. The second application of the package was in the area of robot vision, the ultimate target area of the development. A communication protocol between the vision sensor and a robot system has been developed and described. It features simplicity and programming flexibility despite severe limitations in the communication capabilities of the robot controller at the time of implementation.

Although an increasing number of commercial vision systems is available, the difficulty in recognizing parts from their shape still exists. Most of the commercial systems rely on global features of the presented object for shape recognition, but this approach fails to recognize a shape which is partly occluded or only partially visible. In order to solve this problem, it is necessary to find accurate boundary related parameters with little computational effort, and this is one of the basic areas of research in shape description. The first part of this work describes an error analysis for the length estimation of digitized straight lines as a function of the object orientation.

The conclusion from this investigation is that the accuracy of line length calculations depends basically on several factors. The pixel shape is important, whether square, rectangular or hexagonal. The pixel configuration can also be significant and the two basic ones, regular arrays and triplet configurations are examined. The error function developed expresses the length deviation in percent for each pixel shape and configuration by choosing the appropriate parameters. The length deviation of a digitized straight line is basically independent of its length. The only contributing factor is quantization noise. Circular arcs can also be included in this analysis.

It was found that the length deviation of a circle corresponds to the integration of the error function. A circle therefore has a constant error value depending only on the chain code set
employed and the pixel environment the chain code is based on. Since the error function of investigated pixel configurations and pixel shapes is periodic with \( \pi \), semi-circular arcs also possess a constant error value irrespective of their orientation. If the semi-circular shape is closed by a straight line, both effects are overlapped. In other words, the error contribution of the straight line as a function of its orientation is offset by a constant error value from the circular segment of the shape. Other simple shapes have also been investigated including squares and triangles. A new method is introduced to calculate specific error profiles for arbitrary shapes consisting of straight lines and circular arcs. A unique vector description of the shape, described in Chapter 4, has to be correlated with the appropriate error function. Appendix 2 displays a complete set of individual error profiles for these 4 basic shapes in various pixel environments.

Algorithms to reduce these basic errors are investigated in Chapter 5. Two new algorithms are discussed, one based on crack codes (4 direction) and the other based on the most common chain code (8 direction). Both methods compare favourably with existing techniques, the major advantage being simplicity and therefore computation speed. As a further step towards a suitable shape description for recognition purposes, a new algorithm named arc operator is developed.

The operator takes advantage of a simplified calculation of curvature. The algorithm is based on the deviation of the actual boundary from a chord or string connecting two pixels of the boundary. The arc operator can detect corners and successfully divides circular arcs into straight line components. The benefit of the arc operator system is that the segment length can be calibrated to the actual arc length using the algorithm for length approximation. Matching procedures in the slope angle domain are then simplified. However, choosing the initial parameter for the arc operator is critical due to quantization noise and can lead to reliability problems. Some of the shortcomings can be overcome by fitting circular arcs to the object silhouette rather than straight lines. In the slope angle domain this is achieved by fitting straight lines. As shown in Chapter 6, this approach can be very useful for recognizing local features and therefore provides the possibility of recognizing partially visible objects. The main research still needed on partial shape recognition is in the field of practical applications and reliable simplified approximation algorithms.

The arc operator is an excellent starting point here. However, parameter selection is dependent to some extent on the individual shape (for example whether it consists mainly of circular arcs or curved segments). The development of an automatic selection algorithm would benefit the robustness of the procedure and would extend application areas of this algorithm. Incorporating segment length adjustments by the application of the algorithms based on the error analysis would simplify the final matching procedure. Additional practical experiments in this area are needed to evaluate the algorithm for industrial applications.

To summarize, this thesis has contributed algorithms to reduce errors incorporated in shape description in two areas. Firstly, to improve accuracy in chain coding outlines and secondly to improve processing speed in shape approximation. These algorithms have been developed from their theoretical basis to the point where they are ready for evaluation in industrial applications.
CHAPTER EIGHT

8. REFERENCES

Agin,G.J. 1975:

Agin,G.J. and Duda,R.O. 1975:

Allan Bradley 1987:

Anderson, I.M. and Bezdek,J.C. 1984:

Ballard,D. and Brown,C. 1982:
Computer Vision. Prentice-Hall.

Batchelor,B.1979 :
Using Concavity Trees for Shape Description. IEE Journ. on Computers and Digital Techniques 2(4).

Batchelor,B. et al 1985:

Blum 1976:

Bolles,R.and Cain,A.1982:

Bowman,C.C. and Batchelor,B. 1985:

References

Braggins, D. 1984:

Bretschi, J. 1981:

Brune, W. and Bitter, K. 1983:

Burford, R.G. 1987:

Chaudhuri, B.B. and Kudu, M.K. 1984:
Digital Line Segment Coding: A New Efficient Contour Coding Scheme. IEE Proc., Part E, 131(4).

Clist, R.C. 1987:
The shape of things to come. New Electronics, New Zealand, (11):44-47.

Cunningham, R. 1981:

Davis, L.S. 1977:

Dessimoz, J.D. and Kamenos, P. 1983:

Doerr, A.R.J. 1983:

Dunkelberger, K.A. and Mitchel, R.O. 1985:
Contour Tracing for Precision Measurement. IEEE Int. Conf. on Robotics and Automation, p. 22-27, St. Louis, Missouri, USA.

Ellis, T.J. and Proffitt, D. 1979:

Video Signal Input. Robotics Age 3(3).
Foith, J.P. 1981:  

Foith, J.P. 1982:  
Intelligente Bildsensoren zum Sichten, Handhaben, Steuern und Regeln - Intelligent image sensors for inspection, handling and control, Springer, in German.

Freeman, H. 1961:  

Freeman, H. 1977:  
Shape Description via the Use of Critical Points. Proc. of the IEEE Conf. on Pattern Recognition and Image Processing, New York.

Freeman, H. and Davis L. 1977:  

Friedrich, W.E. 1985:  

Friedrich, W.E. and Arndt, G. 1986:  

Friedrich, W.E. and Arndt, G. 1988:  

Friedrich, W.E. 1989:  

Fu, K.S., Gonzalez, R.C. and Lee, C.S.G. 1987:  

Geissler, W. 1982:  
Vision System for Shape and Position Recognition of Industrial Parts. Proc. of the 2nd Int. Conf. on Robot Vision and Sensory Controls, Stuttgart, FRG.

Gonzalez, R. and Wintz 1977:  
Image Processing. Addison-Wesley.

Gonzalez, R. and Safabaksh, R. 1982:  
References


Malone,D.E. and Friedrich,W.E. 1990:  
Expert Production Automation System using Sensors. SPIE Conference on Applications of Artificial Intelligence VIII, Orlando, Florida, USA.

Marlow,B.E. and Batchelor,B.1980:  

Marlow,B.E. and Batchelor,B.1981:  

Marr,D. and Hildreth,E.1980:  

Nahim, P.J. 1974:  

Pavlidis,T. and Horowitz,S.L. 1974:  

Pavlidis,T. 1977:  

Pavlidis,T. 1982:  

Perkins,W.A. 1978:  

Pratt,W 1979.:  

Proffitt,D. and Rosen,D. 1979:  

Pugh,A. 1983:  

Pugh,A. 1985:  

Ramer,U. 1972:  

Rosenfeld, A. and Johnston, E. 1973:

Rosenfeld, A. and Weszka, J.S. 1975:

Rosenfeld, A. and Kak, A.C. 1982:

Sankar, P.V. and Sharma, C.V. 1978:

Scholten, D.K. 1983:

Slansky, J., Chazin, R.L. and Hansen, B.J. 1972:

Slansky, J. and Gonzales, V. 1980:
Fast Polygonal Approximation of digitized curves, Pattern Recognition, Vol. 12, 327-331.

Taylor, R. 1983:

Teh, C-H. and Chin, R.T. 1988:

Turney, J.L. et. al. 1985:

Weszka, J.S. 1978:

Vernon, D. 1978:

Zahn, C.T. and Roskies, R.Z. 1972:
APPENDICES

APPENDIX A
Error Function Coefficients
Matrix Form of Error Function

APPENDIX B
Catalogue of Error Profiles (Length Deviations) of Simple Geometric Shapes.

APPENDIX C
Software Library
Routine Descriptions as obtainable by the Robot Vision On-Line Help Facility.

APPENDIX D
Robot-Sensor Communication and Protocol

APPENDIX E
Published papers

Vision-Aided Flexible Component Handling
Proc. of the 1st Conf. on Robotics and Handling Automation
(ROBHANZ) p.45-53, Auckland, Nov 1986 also in Automation and Control (NZ),

A New Silhouette-related Algorithm for Improving
Vision-Aided Robot Applications.
Annals of the CIRP
Vol 37 (1): 481-484, 1988

submitted to SPIE Conference on Visual Communications
and Image Processing '89, Philadelphia, Pennsylvania, USA.
## APPENDIX A

### ERROR FUNCTION - COEFFICIENTS AND MATRICES

#### 4 direction code - coefficients-Square pixels

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 27.324%  maximum 41.421% at 45.00 deg  area: 2.000

Average error of 4 direction chain code: 27.324%

#### 8 direction code - coefficients-Square pixels

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.414</td>
</tr>
<tr>
<td>0.785</td>
<td>45.00</td>
<td>0.414</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 5.479%  maximum 8.239% at 22.50 deg  area: 0.828
Local average: 5.479%  maximum 8.239% at 67.50 deg  area: 0.828

Average error of 8 direction chain code: 5.479%

#### 16 direction code - coefficients-Square pixels

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.236</td>
</tr>
<tr>
<td>0.464</td>
<td>26.57</td>
<td>0.822</td>
<td>0.592</td>
</tr>
<tr>
<td>0.785</td>
<td>45.00</td>
<td>0.592</td>
<td>0.822</td>
</tr>
</tbody>
</table>
Local average: 1.831% maximum 2.749% at 13.28 deg area: 0.472
Local average: 0.872% maximum 1.308% at 35.78 deg area: 0.325
Local average: 0.872% maximum 1.308% at 54.22 deg area: 0.325
Local average: 1.831% maximum 2.749% at 76.72 deg area: 0.472

Average error of 16 direction chain code: 1.438%

24 direction code - coefficients-Square pixels

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.162</td>
</tr>
<tr>
<td>0.322</td>
<td>18.43</td>
<td>0.906</td>
<td>0.443</td>
</tr>
<tr>
<td>0.588</td>
<td>33.69</td>
<td>0.777</td>
<td>0.637</td>
</tr>
<tr>
<td>0.785</td>
<td>45.00</td>
<td>0.637</td>
<td>0.777</td>
</tr>
<tr>
<td>0.983</td>
<td>56.31</td>
<td>0.443</td>
<td>0.906</td>
</tr>
<tr>
<td>1.249</td>
<td>71.57</td>
<td>0.162</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 0.872% maximum 1.308% at 9.22 deg area: 0.325
Local average: 0.595% maximum 0.893% at 26.06 deg area: 0.268
Local average: 0.326% maximum 0.489% at 39.35 deg area: 0.198
Local average: 0.326% maximum 0.489% at 50.65 deg area: 0.198
Local average: 0.595% maximum 0.893% at 63.94 deg area: 0.268
Local average: 0.872% maximum 1.308% at 80.78 deg area: 0.325

Average error of 24 direction chain code: 0.641%
### 4 direction code - coefficients - Rectangular pixels

<table>
<thead>
<tr>
<th>Rad deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571 90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 27.324%  maximum 41.421% at 45.00 deg  area: 2.000

Average error of 4 direction chain code: 27.324%

### 8 direction code - coefficients - Rectangular pixels

<table>
<thead>
<tr>
<th>Rad deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1.000</td>
<td>0.333</td>
</tr>
<tr>
<td>0.644 36.87</td>
<td>0.500</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571 90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 3.600%  maximum 5.409% at 18.43 deg  area: 0.667

Local average: 7.841%  maximum 11.803% at 63.43 deg  area: 1.000

Average error of 8 direction chain code: 6.103%

### 16 direction code - coefficients - Rectangular pixels

<table>
<thead>
<tr>
<th>Rad deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1.000</td>
<td>0.181</td>
</tr>
<tr>
<td>0.359 20.56</td>
<td>0.886</td>
<td>0.485</td>
</tr>
<tr>
<td>0.644 36.87</td>
<td>0.697</td>
<td>0.737</td>
</tr>
<tr>
<td>0.983 56.31</td>
<td>0.303</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571 90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 5.464%  maximum 10.475% at 23.68 deg  area: 1.000

Local average: 7.841%  maximum 11.803% at 63.43 deg  area: 1.000

Average error of 16 direction chain code: 6.103%
Local average: 1.087% maximum 1.631% at 10.28 deg area: 0.363
Local average: 0.681% maximum 1.022% at 28.71 deg area: 0.287
Local average: 0.971% maximum 1.456% at 46.59 deg area: 0.343
Local average: 2.984% maximum 4.483% at 73.15 deg area: 0.606

Average error of 16 direction chain code: 1.698%

24 direction code - coefficients-Rectangular pixels

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.123</td>
</tr>
<tr>
<td>0.245</td>
<td>14.04</td>
<td>0.944</td>
<td>0.349</td>
</tr>
<tr>
<td>0.464</td>
<td>26.57</td>
<td>0.854</td>
<td>0.528</td>
</tr>
<tr>
<td>0.644</td>
<td>36.87</td>
<td>0.740</td>
<td>0.681</td>
</tr>
<tr>
<td>0.844</td>
<td>48.37</td>
<td>0.548</td>
<td>0.851</td>
</tr>
<tr>
<td>1.153</td>
<td>66.04</td>
<td>0.212</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 0.503% maximum 0.755% at 7.02 deg area: 0.246
Local average: 0.400% maximum 0.601% at 20.30 deg area: 0.220
Local average: 0.270% maximum 0.406% at 31.72 deg area: 0.180
Local average: 0.337% maximum 0.505% at 42.62 deg area: 0.201
Local average: 0.800% maximum 1.201% at 57.20 deg area: 0.311
Local average: 1.484% maximum 2.227% at 78.02 deg area: 0.424

Average error of 24 direction chain code: 0.760%
### 6 direction code - coefficients-Hexagonal pixels

<table>
<thead>
<tr>
<th>Rad (deg)</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.577</td>
</tr>
<tr>
<td>1.047</td>
<td>-0.000</td>
<td>1.155</td>
</tr>
<tr>
<td>2.094</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 10.266% maximum 15.470% at 30.00 deg area: 1.155
Local average: 10.266% maximum 15.470% at 90.00 deg area: 0.577

Average error of 6 direction chain code: 10.266%

### 12 direction code - coefficients-Hexagonal pixels

<table>
<thead>
<tr>
<th>Rad (deg)</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.268</td>
</tr>
<tr>
<td>0.524</td>
<td>0.732</td>
<td>0.732</td>
</tr>
<tr>
<td>1.047</td>
<td>0.268</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 2.349% maximum 3.528% at 15.00 deg area: 0.536
Local average: 2.349% maximum 3.528% at 45.00 deg area: 0.536
Local average: 2.349% maximum 3.528% at 75.00 deg area: 0.536

Average error of 12 direction chain code: 2.349%

### 18 direction code - coefficients-Hexagonal pixels

<table>
<thead>
<tr>
<th>Rad (deg)</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.168</td>
</tr>
<tr>
<td>0.333</td>
<td>0.882</td>
<td>0.509</td>
</tr>
<tr>
<td>Rad</td>
<td>deg</td>
<td>A</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
</tr>
<tr>
<td>0.243</td>
<td>13.90</td>
<td>0.937</td>
</tr>
<tr>
<td>0.524</td>
<td>30.00</td>
<td>0.795</td>
</tr>
<tr>
<td>0.805</td>
<td>46.10</td>
<td>0.606</td>
</tr>
<tr>
<td>1.047</td>
<td>60.00</td>
<td>0.394</td>
</tr>
<tr>
<td>1.290</td>
<td>73.90</td>
<td>0.141</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
</tr>
</tbody>
</table>

Local average: 0.493% maximum 0.740% at 6.95 deg area: 0.244
Local average: 0.663% maximum 0.995% at 21.95 deg area: 0.283
Local average: 0.663% maximum 0.995% at 38.05 deg area: 0.283
Local average: 0.493% maximum 0.740% at 53.05 deg area: 0.244
Local average: 0.493% maximum 0.740% at 66.95 deg area: 0.244
Local average: 0.663% maximum 0.995% at 81.95 deg area: 0.283

Average error of 24 direction chain code: 0.585%
### 6 direction code - coefficients-Triplets (square)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.618</td>
</tr>
<tr>
<td>1.107</td>
<td>63.43</td>
<td>-0.000</td>
<td>1.118</td>
</tr>
<tr>
<td>2.034</td>
<td>116.57</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 11.644\% maximum 17.557\% at 31.72 deg area: 1.236
Local average: 7.841\% maximum 11.803\% at 90.00 deg area: 0.500

Average error of 6 direction chain code: 10.522\%

### 12 direction code - coefficients-Triplets (square)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.303</td>
</tr>
<tr>
<td>0.588</td>
<td>33.69</td>
<td>0.685</td>
<td>0.776</td>
</tr>
<tr>
<td>1.107</td>
<td>63.43</td>
<td>0.236</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 2.984\% maximum 4.483\% at 16.85 deg area: 0.606
Local average: 2.308\% maximum 3.466\% at 48.56 deg area: 0.531
Local average: 1.831\% maximum 2.749\% at 76.72 deg area: 0.472

Average error of 12 direction chain code: 2.420\%

### 18 direction code - coefficients-Triplets (square)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.193</td>
</tr>
<tr>
<td>0.381</td>
<td>21.80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average error of 18 direction chain code: 2.420\%
<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.785</td>
<td>45.00</td>
<td>0.852</td>
<td>0.562</td>
</tr>
<tr>
<td>1.107</td>
<td>63.43</td>
<td>0.592</td>
<td>0.822</td>
</tr>
<tr>
<td>1.406</td>
<td>80.54</td>
<td>0.313</td>
<td>0.962</td>
</tr>
<tr>
<td>1.736</td>
<td>99.46</td>
<td>-0.000</td>
<td>1.014</td>
</tr>
</tbody>
</table>

Local average:
- 0.785: 1.224% maximum 1.838% at 10.90 deg area: 0.385
- 1.107: 1.389% maximum 2.085% at 33.40 deg area: 0.411
- 1.406: 0.872% maximum 1.308% at 54.22 deg area: 0.325
- 1.736: 0.749% maximum 1.124% at 71.99 deg area: 0.301
- Average error of 18 direction chain code: 1.072%

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.140</td>
</tr>
<tr>
<td>0.278</td>
<td>15.95</td>
<td>0.919</td>
<td>0.425</td>
</tr>
<tr>
<td>0.588</td>
<td>33.69</td>
<td>0.752</td>
<td>0.675</td>
</tr>
<tr>
<td>0.876</td>
<td>50.19</td>
<td>0.551</td>
<td>0.843</td>
</tr>
<tr>
<td>1.107</td>
<td>63.43</td>
<td>0.349</td>
<td>0.944</td>
</tr>
<tr>
<td>1.326</td>
<td>75.96</td>
<td>0.123</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average:
- 0.278: 0.650% maximum 0.976% at 7.97 deg area: 0.280
- 0.588: 0.807% maximum 1.211% at 24.82 deg area: 0.312
- 0.876: 0.697% maximum 1.046% at 41.94 deg area: 0.290
- 1.107: 0.447% maximum 0.671% at 56.81 deg area: 0.232
- 1.326: 0.400% maximum 0.601% at 69.70 deg area: 0.220
- 1.571: 0.503% maximum 0.755% at 82.98 deg area: 0.246

Average error of 24 direction chain code: 0.602%
### 6 direction code - coefficients-Triplets (rectangular)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1.000</td>
<td>0.535</td>
</tr>
<tr>
<td>0.983</td>
<td>56.31</td>
<td>-0.000</td>
<td>1.202</td>
</tr>
<tr>
<td>2.159</td>
<td>123.69</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 8.911% maximum 13.421% at 28.15 deg area: 1.070  
Local average: 13.378% maximum 20.185% at 90.00 deg area: 0.667

**Average error of 6 direction chain code:** 10.583%

### 12 direction code - coefficients-Triplets (rectangular)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1.000</td>
<td>0.236</td>
</tr>
<tr>
<td>0.464</td>
<td>26.57</td>
<td>0.776</td>
<td>0.685</td>
</tr>
<tr>
<td>0.983</td>
<td>56.31</td>
<td>0.303</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 1.831% maximum 2.749% at 13.28 deg area: 0.472  
Local average: 2.308% maximum 3.466% at 41.44 deg area: 0.531  
Local average: 2.984% maximum 4.483% at 73.15 deg area: 0.606

**Average error of 12 direction chain code:** 2.420%

### 18 direction code - coefficients-Triplets (rectangular)

<table>
<thead>
<tr>
<th>Rad</th>
<th>deg</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1.000</td>
<td>0.147</td>
</tr>
<tr>
<td>0.291</td>
<td>16.70</td>
<td>0.907</td>
<td>0.458</td>
</tr>
<tr>
<td>Rad</td>
<td>deg</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>0.106</td>
</tr>
<tr>
<td>0.211</td>
<td>12.09</td>
<td>0.951</td>
<td>0.334</td>
</tr>
<tr>
<td>0.464</td>
<td>26.57</td>
<td>0.834</td>
<td>0.568</td>
</tr>
<tr>
<td>0.733</td>
<td>41.99</td>
<td>0.659</td>
<td>0.762</td>
</tr>
<tr>
<td>0.983</td>
<td>56.31</td>
<td>0.443</td>
<td>0.906</td>
</tr>
<tr>
<td>1.249</td>
<td>71.57</td>
<td>0.162</td>
<td>1.000</td>
</tr>
<tr>
<td>1.571</td>
<td>90.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local average: 0.373% maximum 0.560% at 6.05 deg area: 0.212
Local average: 0.535% maximum 0.803% at 19.33 deg area: 0.254
Local average: 0.608% maximum 0.913% at 34.28 deg area: 0.271
Local average: 0.524% maximum 0.786% at 49.15 deg area: 0.251
Local average: 0.595% maximum 0.893% at 63.94 deg area: 0.268
Local average: 0.872% maximum 1.308% at 80.78 deg area: 0.325

Average error of 24 direction chain code: 0.603%
Chain code error matrix for digital straight lines

Estimation error of 4 code in % - square pixels

<table>
<thead>
<tr>
<th>Y\X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41.42</td>
<td>34.16</td>
<td>26.49</td>
<td>21.27</td>
<td>17.67</td>
<td>15.08</td>
<td>13.14</td>
<td>11.63</td>
<td>10.43</td>
<td>9.45</td>
</tr>
<tr>
<td>2</td>
<td>34.16</td>
<td>41.42</td>
<td>38.68</td>
<td>34.16</td>
<td>29.99</td>
<td>26.49</td>
<td>23.62</td>
<td>21.27</td>
<td>19.31</td>
<td>17.67</td>
</tr>
<tr>
<td>3</td>
<td>26.49</td>
<td>38.68</td>
<td>41.42</td>
<td>40.0</td>
<td>37.2</td>
<td>34.16</td>
<td>31.31</td>
<td>28.75</td>
<td>26.49</td>
<td>24.52</td>
</tr>
<tr>
<td>4</td>
<td>21.27</td>
<td>34.16</td>
<td>40.0</td>
<td>41.42</td>
<td>40.56</td>
<td>38.68</td>
<td>36.44</td>
<td>34.16</td>
<td>31.99</td>
<td>29.99</td>
</tr>
<tr>
<td>5</td>
<td>17.67</td>
<td>29.99</td>
<td>37.2</td>
<td>40.56</td>
<td>41.42</td>
<td>40.84</td>
<td>39.5</td>
<td>37.8</td>
<td>35.98</td>
<td>34.16</td>
</tr>
<tr>
<td>6</td>
<td>15.08</td>
<td>26.49</td>
<td>34.16</td>
<td>38.68</td>
<td>40.84</td>
<td>41.42</td>
<td>41.0</td>
<td>40.0</td>
<td>38.68</td>
<td>37.2</td>
</tr>
<tr>
<td>7</td>
<td>13.14</td>
<td>23.62</td>
<td>31.31</td>
<td>36.44</td>
<td>39.5</td>
<td>41.0</td>
<td>41.42</td>
<td>41.11</td>
<td>40.33</td>
<td>39.27</td>
</tr>
<tr>
<td>8</td>
<td>11.63</td>
<td>21.27</td>
<td>28.75</td>
<td>34.16</td>
<td>37.8</td>
<td>40.0</td>
<td>41.11</td>
<td>41.42</td>
<td>41.18</td>
<td>40.56</td>
</tr>
<tr>
<td>9</td>
<td>10.43</td>
<td>19.31</td>
<td>26.49</td>
<td>31.99</td>
<td>35.98</td>
<td>38.68</td>
<td>40.33</td>
<td>41.18</td>
<td>41.42</td>
<td>41.23</td>
</tr>
<tr>
<td>10</td>
<td>9.45</td>
<td>17.67</td>
<td>24.52</td>
<td>29.99</td>
<td>34.16</td>
<td>37.2</td>
<td>39.27</td>
<td>40.56</td>
<td>41.23</td>
<td>41.42</td>
</tr>
</tbody>
</table>

Estimation error of 4 code in % - rectangular pixels

<table>
<thead>
<tr>
<th>Y\X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40.00</td>
<td>28.75</td>
<td>21.27</td>
<td>16.72</td>
<td>13.73</td>
<td>11.63</td>
<td>10.08</td>
<td>8.90</td>
<td>7.96</td>
<td>7.20</td>
</tr>
<tr>
<td>2</td>
<td>38.68</td>
<td>40.00</td>
<td>34.16</td>
<td>28.75</td>
<td>24.52</td>
<td>21.27</td>
<td>18.73</td>
<td>16.72</td>
<td>15.08</td>
<td>13.73</td>
</tr>
<tr>
<td>3</td>
<td>32.00</td>
<td>41.18</td>
<td>40.00</td>
<td>36.18</td>
<td>32.23</td>
<td>28.75</td>
<td>25.80</td>
<td>23.34</td>
<td>21.27</td>
<td>19.51</td>
</tr>
<tr>
<td>4</td>
<td>26.49</td>
<td>38.68</td>
<td>41.42</td>
<td>40.00</td>
<td>37.20</td>
<td>34.16</td>
<td>31.31</td>
<td>28.75</td>
<td>26.49</td>
<td>24.52</td>
</tr>
<tr>
<td>5</td>
<td>22.39</td>
<td>35.29</td>
<td>40.56</td>
<td>41.35</td>
<td>40.00</td>
<td>37.80</td>
<td>35.37</td>
<td>32.99</td>
<td>30.77</td>
<td>28.75</td>
</tr>
<tr>
<td>6</td>
<td>19.31</td>
<td>32.00</td>
<td>38.68</td>
<td>41.18</td>
<td>41.23</td>
<td>40.00</td>
<td>38.19</td>
<td>36.18</td>
<td>34.16</td>
<td>32.23</td>
</tr>
<tr>
<td>7</td>
<td>16.95</td>
<td>29.05</td>
<td>36.44</td>
<td>40.15</td>
<td>41.38</td>
<td>41.11</td>
<td>40.00</td>
<td>38.47</td>
<td>36.76</td>
<td>36.02</td>
</tr>
<tr>
<td>8</td>
<td>15.08</td>
<td>26.49</td>
<td>34.16</td>
<td>38.68</td>
<td>40.84</td>
<td>41.42</td>
<td>41.00</td>
<td>40.00</td>
<td>38.68</td>
<td>37.20</td>
</tr>
<tr>
<td>9</td>
<td>13.58</td>
<td>24.29</td>
<td>31.99</td>
<td>37.01</td>
<td>39.88</td>
<td>41.18</td>
<td>41.40</td>
<td>40.92</td>
<td>40.00</td>
<td>38.83</td>
</tr>
<tr>
<td>10</td>
<td>12.34</td>
<td>22.39</td>
<td>29.99</td>
<td>35.29</td>
<td>38.68</td>
<td>40.56</td>
<td>41.34</td>
<td>41.35</td>
<td>40.84</td>
<td>40.00</td>
</tr>
</tbody>
</table>
Estimation error of 8 code in % - square pixels

<table>
<thead>
<tr>
<th>Y \ X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>7.97</td>
<td>7.97</td>
<td>7.06</td>
<td>6.18</td>
<td>5.45</td>
<td>4.85</td>
<td>4.37</td>
<td>3.96</td>
<td>3.63</td>
</tr>
<tr>
<td>2</td>
<td>7.97</td>
<td>0</td>
<td>6.18</td>
<td>7.97</td>
<td>8.23</td>
<td>7.97</td>
<td>7.53</td>
<td>7.06</td>
<td>6.60</td>
<td>6.18</td>
</tr>
<tr>
<td>3</td>
<td>7.97</td>
<td>6.18</td>
<td>0</td>
<td>4.85</td>
<td>7.06</td>
<td>7.97</td>
<td>8.23</td>
<td>8.18</td>
<td>7.97</td>
<td>7.68</td>
</tr>
<tr>
<td>4</td>
<td>7.06</td>
<td>7.97</td>
<td>4.85</td>
<td>0</td>
<td>3.96</td>
<td>6.18</td>
<td>7.38</td>
<td>7.97</td>
<td>8.20</td>
<td>8.23</td>
</tr>
<tr>
<td>5</td>
<td>6.18</td>
<td>8.23</td>
<td>7.06</td>
<td>3.96</td>
<td>0</td>
<td>3.34</td>
<td>5.45</td>
<td>6.75</td>
<td>7.53</td>
<td>7.97</td>
</tr>
<tr>
<td>6</td>
<td>5.45</td>
<td>7.97</td>
<td>7.97</td>
<td>6.18</td>
<td>3.34</td>
<td>0</td>
<td>2.88</td>
<td>4.85</td>
<td>6.18</td>
<td>7.06</td>
</tr>
<tr>
<td>7</td>
<td>4.85</td>
<td>7.53</td>
<td>8.23</td>
<td>7.38</td>
<td>5.45</td>
<td>2.88</td>
<td>0</td>
<td>2.53</td>
<td>4.37</td>
<td>5.68</td>
</tr>
<tr>
<td>8</td>
<td>4.37</td>
<td>7.06</td>
<td>8.18</td>
<td>7.97</td>
<td>6.75</td>
<td>4.85</td>
<td>2.53</td>
<td>0</td>
<td>2.26</td>
<td>3.96</td>
</tr>
<tr>
<td>9</td>
<td>3.96</td>
<td>6.60</td>
<td>7.97</td>
<td>8.20</td>
<td>7.53</td>
<td>6.18</td>
<td>4.37</td>
<td>2.26</td>
<td>0</td>
<td>2.04</td>
</tr>
<tr>
<td>10</td>
<td>3.63</td>
<td>6.18</td>
<td>7.68</td>
<td>8.23</td>
<td>7.97</td>
<td>7.06</td>
<td>5.68</td>
<td>3.96</td>
<td>2.04</td>
<td>0</td>
</tr>
</tbody>
</table>

Estimation error of 8 code in % - rectangular pixels

<table>
<thead>
<tr>
<th>Y \ X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5.34</td>
<td>5.10</td>
<td>4.43</td>
<td>3.48</td>
<td>3.36</td>
<td>2.98</td>
<td>2.67</td>
<td>2.42</td>
<td>2.21</td>
</tr>
<tr>
<td>2</td>
<td>10.94</td>
<td>0</td>
<td>4.35</td>
<td>5.34</td>
<td>5.36</td>
<td>5.10</td>
<td>4.76</td>
<td>4.43</td>
<td>4.12</td>
<td>3.84</td>
</tr>
<tr>
<td>3</td>
<td>11.69</td>
<td>7.96</td>
<td>0</td>
<td>3.50</td>
<td>4.87</td>
<td>5.34</td>
<td>5.40</td>
<td>5.29</td>
<td>5.10</td>
<td>4.88</td>
</tr>
<tr>
<td>4</td>
<td>10.68</td>
<td>10.94</td>
<td>6.07</td>
<td>0</td>
<td>2.90</td>
<td>4.35</td>
<td>5.05</td>
<td>5.34</td>
<td>5.41</td>
<td>5.36</td>
</tr>
<tr>
<td>5</td>
<td>9.51</td>
<td>11.76</td>
<td>9.32</td>
<td>4.87</td>
<td>0</td>
<td>2.47</td>
<td>3.89</td>
<td>4.69</td>
<td>5.13</td>
<td>5.34</td>
</tr>
<tr>
<td>6</td>
<td>8.47</td>
<td>11.69</td>
<td>10.94</td>
<td>7.96</td>
<td>4.06</td>
<td>0</td>
<td>2.14</td>
<td>3.5</td>
<td>4.35</td>
<td>4.87</td>
</tr>
<tr>
<td>7</td>
<td>7.59</td>
<td>11.25</td>
<td>11.63</td>
<td>9.85</td>
<td>6.90</td>
<td>3.48</td>
<td>0</td>
<td>1.89</td>
<td>3.17</td>
<td>4.03</td>
</tr>
<tr>
<td>8</td>
<td>6.86</td>
<td>10.68</td>
<td>11.80</td>
<td>10.94</td>
<td>8.83</td>
<td>6.07</td>
<td>3.04</td>
<td>0</td>
<td>1.70</td>
<td>2.90</td>
</tr>
<tr>
<td>9</td>
<td>6.25</td>
<td>10.08</td>
<td>11.69</td>
<td>11.52</td>
<td>10.12</td>
<td>7.96</td>
<td>5.41</td>
<td>2.70</td>
<td>0</td>
<td>1.53</td>
</tr>
<tr>
<td>10</td>
<td>5.73</td>
<td>9.51</td>
<td>11.42</td>
<td>11.76</td>
<td>10.94</td>
<td>9.32</td>
<td>7.22</td>
<td>4.87</td>
<td>2.43</td>
<td>0</td>
</tr>
</tbody>
</table>
### Estimation error of 16 code in % - rectangular pixels

<table>
<thead>
<tr>
<th>Y\X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.33</td>
<td>2.74</td>
<td>2.69</td>
<td>2.52</td>
<td>2.33</td>
<td>2.16</td>
<td>2.00</td>
<td>1.85</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1.24</td>
<td>0</td>
<td>1.62</td>
<td>2.33</td>
<td>2.64</td>
<td>2.74</td>
<td>2.74</td>
<td>2.69</td>
</tr>
<tr>
<td>3</td>
<td>2.33</td>
<td>1.24</td>
<td>0</td>
<td>1.29</td>
<td>0.95</td>
<td>0</td>
<td>1.21</td>
<td>1.92</td>
<td>2.33</td>
<td>2.57</td>
</tr>
<tr>
<td>4</td>
<td>2.74</td>
<td>0</td>
<td>1.29</td>
<td>0</td>
<td>1.18</td>
<td>1.24</td>
<td>0.75</td>
<td>0</td>
<td>0.97</td>
<td>1.62</td>
</tr>
<tr>
<td>5</td>
<td>2.69</td>
<td>1.62</td>
<td>0.95</td>
<td>1.18</td>
<td>0</td>
<td>1.06</td>
<td>1.31</td>
<td>1.09</td>
<td>0.61</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2.52</td>
<td>2.33</td>
<td>0</td>
<td>1.24</td>
<td>1.06</td>
<td>0</td>
<td>0.95</td>
<td>1.29</td>
<td>1.24</td>
<td>0.95</td>
</tr>
<tr>
<td>7</td>
<td>2.33</td>
<td>2.64</td>
<td>0.75</td>
<td>1.31</td>
<td>0.95</td>
<td>0</td>
<td>0.86</td>
<td>1.24</td>
<td>1.30</td>
<td>1.18</td>
</tr>
<tr>
<td>8</td>
<td>2.16</td>
<td>2.74</td>
<td>1.92</td>
<td>0</td>
<td>1.09</td>
<td>1.29</td>
<td>0.86</td>
<td>0</td>
<td>0.78</td>
<td>0.71</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
<td>2.74</td>
<td>2.33</td>
<td>0.97</td>
<td>0.61</td>
<td>1.24</td>
<td>1.24</td>
<td>0.78</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1.85</td>
<td>2.69</td>
<td>2.57</td>
<td>1.62</td>
<td>0</td>
<td>0.95</td>
<td>1.30</td>
<td>1.18</td>
<td>0.71</td>
<td>0</td>
</tr>
</tbody>
</table>

### Estimation error of 16 code in % - square pixels

<table>
<thead>
<tr>
<th>Y\X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.33</td>
<td>2.74</td>
<td>2.69</td>
<td>2.52</td>
<td>2.33</td>
<td>2.16</td>
<td>2.00</td>
<td>1.85</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1.24</td>
<td>0</td>
<td>1.62</td>
<td>2.33</td>
<td>2.64</td>
<td>2.74</td>
<td>2.74</td>
<td>2.69</td>
</tr>
<tr>
<td>3</td>
<td>2.33</td>
<td>1.24</td>
<td>0</td>
<td>1.29</td>
<td>0.95</td>
<td>0</td>
<td>1.21</td>
<td>1.92</td>
<td>2.33</td>
<td>2.57</td>
</tr>
<tr>
<td>4</td>
<td>2.74</td>
<td>0</td>
<td>1.29</td>
<td>0</td>
<td>1.18</td>
<td>1.24</td>
<td>0.75</td>
<td>0</td>
<td>0.97</td>
<td>1.62</td>
</tr>
<tr>
<td>5</td>
<td>2.69</td>
<td>1.62</td>
<td>0.95</td>
<td>1.18</td>
<td>0</td>
<td>1.06</td>
<td>1.31</td>
<td>1.09</td>
<td>0.61</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2.52</td>
<td>2.33</td>
<td>0</td>
<td>1.24</td>
<td>1.06</td>
<td>0</td>
<td>0.95</td>
<td>1.29</td>
<td>1.24</td>
<td>0.95</td>
</tr>
<tr>
<td>7</td>
<td>2.33</td>
<td>2.64</td>
<td>0.75</td>
<td>1.31</td>
<td>0.95</td>
<td>0</td>
<td>0.86</td>
<td>1.24</td>
<td>1.30</td>
<td>1.18</td>
</tr>
<tr>
<td>8</td>
<td>2.16</td>
<td>2.74</td>
<td>1.92</td>
<td>0</td>
<td>1.09</td>
<td>1.29</td>
<td>0.86</td>
<td>0</td>
<td>0.78</td>
<td>0.71</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
<td>2.74</td>
<td>2.33</td>
<td>0.97</td>
<td>0.61</td>
<td>1.24</td>
<td>1.24</td>
<td>0.78</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1.85</td>
<td>2.69</td>
<td>2.57</td>
<td>1.62</td>
<td>0</td>
<td>0.95</td>
<td>1.30</td>
<td>1.18</td>
<td>0.71</td>
<td>0</td>
</tr>
</tbody>
</table>
### Estimation error of 24 code in % - square pixels

<table>
<thead>
<tr>
<th>Y \ X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.95</td>
<td>1.24</td>
<td>1.31</td>
<td>1.29</td>
<td>1.24</td>
<td>1.18</td>
<td>1.12</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.24</td>
<td>0</td>
<td>0.61</td>
<td>0.95</td>
<td>1.14</td>
<td>1.24</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.40</td>
<td>0.18</td>
<td>0</td>
<td>0.24</td>
<td>0.19</td>
<td>0</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>0.95</td>
<td>0</td>
<td>0.40</td>
<td>0</td>
<td>0.48</td>
<td>0</td>
<td>0.19</td>
<td>0</td>
<td>0.22</td>
<td>0.24</td>
</tr>
<tr>
<td>5</td>
<td>1.24</td>
<td>0.24</td>
<td>0.18</td>
<td>0.48</td>
<td>0</td>
<td>0.49</td>
<td>0.27</td>
<td>0.14</td>
<td>0.18</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1.31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.49</td>
<td>0</td>
<td>0.46</td>
<td>0.40</td>
<td>0</td>
<td>0.18</td>
</tr>
<tr>
<td>7</td>
<td>1.29</td>
<td>0.61</td>
<td>0.24</td>
<td>0.19</td>
<td>0.27</td>
<td>0.46</td>
<td>0</td>
<td>0.44</td>
<td>0.46</td>
<td>0.20</td>
</tr>
<tr>
<td>8</td>
<td>1.24</td>
<td>0.95</td>
<td>0.19</td>
<td>0</td>
<td>0.14</td>
<td>0.40</td>
<td>0.44</td>
<td>0</td>
<td>0.41</td>
<td>0.48</td>
</tr>
<tr>
<td>9</td>
<td>1.18</td>
<td>1.14</td>
<td>0</td>
<td>0.22</td>
<td>0.18</td>
<td>0</td>
<td>0.46</td>
<td>0.41</td>
<td>0</td>
<td>0.38</td>
</tr>
<tr>
<td>10</td>
<td>1.12</td>
<td>1.24</td>
<td>0.45</td>
<td>0.24</td>
<td>0</td>
<td>0.18</td>
<td>0.20</td>
<td>0.48</td>
<td>0.38</td>
<td>0</td>
</tr>
</tbody>
</table>

### Estimation error of 24 code in % - rectangular pixels

<table>
<thead>
<tr>
<th>Y \ X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.56</td>
<td>0.72</td>
<td>0.75</td>
<td>0.74</td>
<td>0.71</td>
<td>0.68</td>
<td>0.64</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.16</td>
<td>0</td>
<td>0.36</td>
<td>0.56</td>
<td>0.66</td>
<td>0.72</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.32</td>
<td>0.13</td>
<td>0</td>
<td>0.16</td>
<td>0.12</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1.58</td>
<td>0</td>
<td>0.42</td>
<td>0</td>
<td>0.40</td>
<td>0</td>
<td>0.14</td>
<td>0</td>
<td>0.14</td>
<td>0.16</td>
</tr>
<tr>
<td>5</td>
<td>2.09</td>
<td>0.35</td>
<td>0.23</td>
<td>0.50</td>
<td>0</td>
<td>0.40</td>
<td>0.21</td>
<td>0.10</td>
<td>0.12</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2.22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.50</td>
<td>0</td>
<td>0.39</td>
<td>0.32</td>
<td>0</td>
<td>0.13</td>
</tr>
<tr>
<td>7</td>
<td>2.20</td>
<td>1.00</td>
<td>0.35</td>
<td>0.24</td>
<td>0.29</td>
<td>0.47</td>
<td>0</td>
<td>0.37</td>
<td>0.37</td>
<td>0.16</td>
</tr>
<tr>
<td>8</td>
<td>2.13</td>
<td>1.58</td>
<td>0.28</td>
<td>0</td>
<td>0.17</td>
<td>0.42</td>
<td>0.44</td>
<td>0</td>
<td>0.35</td>
<td>0.40</td>
</tr>
<tr>
<td>9</td>
<td>2.03</td>
<td>1.91</td>
<td>0</td>
<td>0.31</td>
<td>0.22</td>
<td>0</td>
<td>0.48</td>
<td>0.41</td>
<td>0</td>
<td>0.32</td>
</tr>
<tr>
<td>10</td>
<td>1.93</td>
<td>2.09</td>
<td>0.73</td>
<td>0.35</td>
<td>0</td>
<td>0.23</td>
<td>0.21</td>
<td>0.5</td>
<td>0.38</td>
<td>0</td>
</tr>
</tbody>
</table>
APPENDIX B

CATALOGUE OF ERROR PROFILES

Perimeter Length Deviations of Simple Geometric Shapes in Relation to Their Orientation Using Different Chain Codes, Pixel Shapes, and Configurations

Straight lines, Full Circle, Semi-Circle, Triangle, Square and Rectangle
are considered based on square and rectangular pixels which are configured in a regular array or organized as triplets as well as hexagonal pixels coded with 4, 8, 16 and 24 direction code.

Practical results for square and rectangular pixels are also included using coding schemes described in Chapter 5.
**Straight Line**

*Square Pixels — Triplet Configuration*

Perimeter Deviation

<table>
<thead>
<tr>
<th>Orientation</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

6 dir code (top)  12 dir code (bottom)

**Straight Line**

*Square Pixels — Triplet Configuration*

Perimeter Deviation

<table>
<thead>
<tr>
<th>Orientation</th>
<th>0%</th>
<th>0.5%</th>
<th>1%</th>
<th>1.5%</th>
<th>2%</th>
<th>2.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

18 dir code (top)  24 dir code (bottom)
**Straight Line**
Rectangular Pixels – Triplet Configured

Perimeter Deviation

Orientation

- 6 dir code (top)  
- 12 dir code (bottom)

---

**Straight Line**
Rectangular Pixels – Triplet Configured

Perimeter Deviation

Orientation

- 18 dir code (top)  
- 24 dir code (bottom)
Straight Line
Hexagonal Pixels

Perimeter Deviation

Orientation

6 dir code (top)  12 dir code (bottom)

Straight Line
Hexagonal Pixels

Perimeter Deviation

Orientation

18 dir code (top)  24 dir code (bottom)
The remainder of this section shows perimeter deviation profiles of simple geometric shapes consisting of circular arcs and straight line segments. The presented shapes are circles and semi-circles, triangles, squares, and rectangles coded with 4, 8, 16 and 24 direction code in a regular pixel array of square or rectangular pixels.

For comparison the theoretical results are shown on the top of the page using the equations presented in Chapter 4, simulated results are presented on the lower half of the page using the approximation methods described in Chapter 5. For hexagonal pixelshapes and related configurations only the theoretical error analysis is presented.
Full Circle  
Square Pixels – Array Configuration

Perimeter Deviation

0%  5%  10%  15%  20%  25%  30%

Orientation

0  15  30  45  60  75  90  105  120  135  150  165  180

4 dir code (top)  8 dir code (bottom)

Circular Shape
Variation of Spatial Resolution

Perimeter Deviation

0%  3%  6%  9%

Spatial Resolution in Pixel (Diameter)

0  50  100  150  200  250  300  350  400

4 Direction Code (Aspect Ratio 1:1)

Dunkelberger  Friedrich  Horn
Error Profiles

Full Circle
Square Pixels – Array Configuration

Perimeter Deviation

Orientation

0 15 30 45 60 75 90 105 120 135 150 165 180

1.6% 1.4% 1.2% 1% 0.8% 0.6% 0.4% 0.2% 0%

16 dir code (top) 24 dir code (top)

Circular Shape
Variation of Spacial Resolution

Perimeter Deviation

Freeman
Dessimoz
Friedrich

Spacial Resolution in Pixel (Diameter)

8 Direction Code (Aspect Ratio 1:1)
Full Circle
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

16 dir code (top) — 24 dir code (bottom)

Circular Shape
Variation of Spatial Resolution

Perimeter Deviation

Spatial Resolution in Pixel (Diameter)

8 Direction Code (Aspect Ratio 4:3)
Full Circle
Rectangular Pixels – Array Configuration

<table>
<thead>
<tr>
<th>Orientation</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
</tr>
</thead>
</table>

Perimeter Deviation

- 4 dir code (top)
- 8 dir code (top)

Circular Shape
Variation of Spacial Resolution

<table>
<thead>
<tr>
<th>Spacial Resolution in Pixel (Diameter)</th>
<th>0%</th>
<th>3%</th>
<th>6%</th>
<th>9%</th>
</tr>
</thead>
</table>

4 Direction Code (Aspect Ratio 4:3)
Semi-Circle
Square Pixels – Array Configuration

Perimeter Deviation

Orientation

4 dir code (top)  8 dir code (bottom)

Semi-Circular Shape
Orientation variation

Length Deviation

Orientation

4 Direction Code (Aspect Ratio 1:1)
Resolution: 200 pixel diameter
Semi-Circle
Square Pixels – Array Configuration

Perimeter Deviation

Orientation

16 dir code (top)  24 dir code (bottom)

Semi-Circular Shape
Orientation variation

Length Deviation

Orientation

8 Direction Code (Aspect Ratio 1:1)
Resolution: 200 pixel diameter
Semi-Circle
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

Semi-Circular Shape
Orientation variation

Length Deviation

Orientation

4 Direction Code (Aspect Ratio 4:3)
Resolution : 200 pixel diameter
Semi Circle
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

- 16 dir code (top) — 24 dir code (bottom)

Semi Circular Shape
Orientation Variation

Perimeter Deviation

Orientation

8 Direction Code (Aspect Ratio 4:3)
Resolution: 200 pixel diameter
**Triangle**

**Square Pixels - Array Configuration**

**Perimeter Deviation**

Orientation

---
4 dir code (top)  
8 dir code (top)

**Triangular Shape**

**Orientation Variation**

**Perimeter Deviation**

Orientation

---
Dunkelberger  
Friedrich  
Horn

**4 Direction Code (Aspect Ratio 1:1)**

Resolution: 200 pixel side length
Triangle
Square Pixels – Array Configuration

Perimeter Deviation

Orientation

16 dir code  24 dir code

Triangular Shape
Orientation Variation

Perimeter Deviation

Orientation

8 Direction Code (Aspect Ratio 1:1)
Resolution: 200 pixel side length

Error Profiles
Triangle
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

4 dir code (top)  8 dir code (bottom)

Triangular Shape
Orientation Variation

Perimeter Deviation

Orientation

4 Direction Code (Aspect Ratio 4:3)
Resolution : 200 pixel side length
Triangle
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

16 dir code (top)  24 dir code (bottom)

Triangular Shape
Orientation variation

Perimeter Deviation

Orientation

8 Direction Code (Aspect Ratio 4:3)
Resolution: 200 pixel side length
Square / Rectangle
Square Pixels – Array Configuration

Perimeter Deviation

<table>
<thead>
<tr>
<th>Orientation</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>15</td>
<td>30</td>
<td>45</td>
<td>60</td>
</tr>
</tbody>
</table>

- 4 dir code (top)
- 8 dir code (bottom)

Square Shape
Orientation Variation

Perimeter Deviation

<table>
<thead>
<tr>
<th>Orientation</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>45</td>
<td>90</td>
</tr>
</tbody>
</table>

4 Direction Code (Aspect Ratio 1:1)
Resolution: 200 pixel side length
Square / Rectangle
Square Pixels – Array Configuration

Perimeter Deviation

Orientation

16 direction code
24 direction code

Square Shape
Orientation variation

Perimeter Deviation

Orientation

8 direction code (Aspect Ratio 1:1)
Resolution: 200 pixel side length
Square
Rectangular Pixels – Array Configuration

Perimeter Deviation

Orientation

- 4 dir code (top)  - 8 dir code (bottom)

Square Shape
Orientation Variation

Perimeter Deviation

Orientation

4 Direction Code (Aspect Ratio 4:3)
Resolution: 200 pixel side length
Square
Rectangular Pixels - Array Configuration

Perimeter Deviation

Orientation

- 16 dir code (top)
- 24 dir code (bottom)

Square Shape
Orientation variation

Perimeter Deviation

Orientation

8 Direction Code (Aspect Ratio 4:3)
Resolution : 200 pixel side length
Rectangle
Rectangular Pixels – Array Configuration

Perimeter Deviation

4 dir code (top)  8 dir code (bottom)

Rectangular Shape
Orientation Variation

Perimeter Deviation

Dunkelberger
Friedrich
Horn

4 Direction Code (Aspect Ratio 4:3)
Resolution: 200 pixel side length
**Rectangle**

*Rectangular Pixels – Array Configuration*

**Perimeter Deviation**

- 16 dir code (top)  
- 24 dir code (bottom)

**Orientation**

**Rectangular Shape**

*Orientation variation*

**Perimeter Deviation**

- Freeman  
- Dessimoz  
- Friedrich

**8 Direction Code (Aspect Ratio 4:3)**

Resolution: 200 pixel side length
Circle
Square Pixels - Triplet Configuration

Perimeter Deviation

Orientation

- 6 dir code (top)  - 12 dir code (bottom)

Circle
Square Pixels - Triplet Configuration

Perimeter Deviation

Orientation

- 18 dir code (top)  - 24 dir code (bottom)
Semi Circle
Square Pixels – Triplet Configuration

Perimeter Deviation

Orientation

6 dir code (top)  12 dir code (bottom)

Semi Circle
Square Pixels – Triplet Configuration

Perimeter Deviation

Orientation

18 dir code  24 dir code
**Square**

*Square Pixels – Triplet Configuration*

**Perimeter Deviation**

- **Orientation**
  - 0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180

- **6 dir code (top)**
- **12 dir code (bottom)**

**Square**

*Square Pixels – Triplet Configuration*

**Perimeter Deviation**

- **Orientation**
  - 0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180

- **18 dir code (top)**
- **24 dir code (bottom)**
Full Circle
Rectangular Pixels — Triplet Configured

Perimeter Deviation

Orientation

- 6 dir code (top)  —  12 dir code (bottom)

Full Circle
Rectangular Pixels — Triplet Configured

Perimeter Deviation

Orientation

- 18 dir code (top)  —  24 dir code (bottom)
Semi Circle
Rectangular Pixels – Triplet Configured

Perimeter Deviation

0%  15  30  45  60  75  90  105  120  135  150  165  180
Orientation

6 dir code (top)  12 dir code (bottom)

Semi Circle
Rectangular Pixels – Triplet Configured

Perimeter Deviation

0%  15  30  45  60  75  90  105  120  135  150  165  180
Orientation

18 dir code (top)  24 dir code (bottom)
Triangle
Rectangular Pixels – Triplet Configured

Perimeter Deviation

Orientation

6 dir code (top) — 12 dir code (bottom)

Triangle
Rectangular Pixels – Triplet Configured

Perimeter Deviation

Orientation

18 dir code (top) — 24 dir code (bottom)
**Rectangle**
Rectangular Pixels – Triplet Configured

**Perimeter Deviation**

- 6 dir code (top)
- 12 dir code (bottom)

**Orientation**

0 15 30 45 60 75 90 105 120 135 150 165 180

---

**Rectangle**
Rectangular Pixels – Triplet Configured

**Perimeter Deviation**

- 18 direction code
- 24 direction code

**Orientation**

0 15 30 45 60 75 90 105 120 135 150 165 180
**Semi Circle Hexagonal Pixels**

*Perimeter Deviation*

0% 15 30 45 60 75 90 105 120 135 150 165 180

- 6 dir code (top)
- 12 dir code (bottom)

**Semi Circle Hexagonal Pixels**

*Perimeter Deviation*

0% 15 30 45 60 75 90 105 120 135 150 165 180

- 16 dir code (top)
- 24 dir code (bottom)
**Square Hexagonal Pixels**

**Perimeter Deviation**

- 6 dir code (top)
- 12 dir code (bottom)

**Orientation**

0 15 30 45 60 75 90 105 120 135 150 165 180

---

**Square Hexagonal Pixels**

**Perimeter Deviation**

- 18 dir code (top)
- 24 dir code (bottom)

**Orientation**

0 15 30 45 60 75 90 105 120 135 150 165 180
Error Profiles

Rectangle
Hexagonal Pixels

Perimeter Deviation

Orientation

--- 6 dir code (top)  --- 12 dir code (bottom)

Rectangular Hexagonal Pixels

Perimeter Deviation

Orientation

--- 18 dir code (top)  --- 24 dir code (bottom)
APPENDIX C

SOFTWARE LIBRARY

Description of routines as available from the
Robot Vision on-line Help Facility for
image processing routines, robot vision programs
and communication procedures.

W E Friedrich
April, 1987.

All commands (bold printing) may be abbreviated; the minimal command is printed in capital letters, the optional rest in small letters.

RVHELP *
for a summary of available information, image processing routines
and robot vision programs

RVHELP HELP
for syntax information on topics and subtopics and

RVHELP "topic/subtopic"
for detailed information under each topic and/or subtopic
HELP Lists helpful information

Syntax:
RVHELP[/option] [ topic[/item1/item2... ]] or RVHELP *

Semantics:
RVHELP * lists the items for which help is available.
RVHELP HELP lists the HELP text (of which this is a part).
RVHELP topic lists information on the specific topic only.
RVHELP topic/item lists the text associated with the specific item under the subtopic OPTIONS.

Items are specific command options.

PRINTER
Prints the RVHELP text on the line printer

TERMINAL (default)
Types the RVHELP text on the terminal

EXAMPLES
RVHELP ERASE Lists information about the ERASE command
RVHELP/PRINTER IMAPRI Prints information about the IMAPRI command
RVHELP VISION/W Describes the W (window) option for VISION
RVHELP *
ADD       Adds the content of an image file to the frame store content
ARCOP     Arc operator - a new polygon approximation technique
ASEA      Lists helpful information on the ASEA - DEC connection.
CAMERA    Select new camera input channel
CCHAIN    Converts chain code into crack code and creates a data file.
CCONV     Converts chain code into crack code or vice versa
COMAND    Communicates commands to the robot (speech controlled robot)
ERASE     Clears the frame store to a predefined grey-level value
ERFUNC    Generates length approximation error function
ERFCON    Error function correlation - for simple geometric shapes
FREEZE    Stores an image into the frame store from the activated input
GRAB      Writes images continuously to the frame store
HARDWARE  Lists information on the MATROX image processing boards
HELP      Lists helpful information on how to use this facility
IMAPRI    Prints an image from the frame store on a DEC LA100 printer
INROVI    INtelligent RObot VIision Program
INTVIS    Intelligent Vision Program
IOTEST    Performs an input/output test on the communication link
MATROX    Lists helpful information on the MATROX image processing board
MULTIO    Identification of different objects in the field of view
NEGATE    Negates the contents of the image buffer (black to white etc.)
ESTPER    Generates simple geometric shapes and simulates outline
READ      Loads the specified (packed) image file into the frame buffer
READN     Reads specified (one pixel value per byte) image file into the frame buffer
READQ     Reads the specified image file into one quadrant of the buffer
RECALL    Reads the specified image file into one the frame buffer
REDUCE    Reduces the image size to one quarter
ROBVIS    Robot Vision Program - recognizes objects at certain positions
SHIFT     Shifts brightness levels of the image by specified value
SLOANG    Draws polygon and slope angle diagram from a vector list
SUB       Subtracts specified image data from content of image buffer
VISION    Interactive Image processing package
WRITE     Writes content of image buffer to disk in compressed form
WRITEN    Writes content of image buffer to disk in non-condensed form
ADD

Adds the content of an image file to the frame store content

SYNTAX: ADD filespecs or RUN [device:]ADD

SEMANTICS

Performs an addition of two images namely the specified image file plus the current content of the frame store.

EXAMPLE: ADD BELT.DAT

ARCOP

Arc operator - a new polygon approximation technique

SYNTAX: ARCOP or RUN [device:]ARCOP

SEMANTICS

ARCOP reads a file from disk containing object outline information in form of a 4 or 8 direction chain-code, converts it to 8 direction code if applicable, performs a polygon approximation using a new method called arc operator, and writes polygon vector coordinates onto disk. ARC - Operator Analysis is carried out with an optional display of resulting curves on a device interpreting DEC's REGIS commands (e.g. Gigi, VT240 etc.). Results comprise:

- 3 segment angles (base for curvature calculation)
- Curvature
- Difference of Curvature (1st derivative curvature)
- Distance function (approximated curvature)
- Difference of distance function (1st derivative thereof)
- 2nd derivative of distance function
- Direction change
- Difference of direction change

Input chain code files can be generated by VISION, CCHAIN or CCONV. Output polygon files can be used by SLOANG to display the polygon and generate a slope angle diagram (see SLOANG description).

OPTIONS

Increment size along the object silhouette
Arc operator segment length
Deviation tolerance
Threshold setting for accumulated deviation
Aspect ratio of frame store
Output control - short or detailed results on screen with optional printout
EXAMPLES

ARCP

Chain code file: BELTCC

(I) Increment size: 1 ! incl. all pixels of the outline
(S) Segment length: 8 ! default arc segment length
(D) Distance tolerance: 15 ! max deviation allowed
(A) Accumulated distance: 90 ! threshold for accumulated deviation
(T) Deviation tolerance: 16 ! max deviation
(X) Aspect ratio (X): 3 ! standard TV ratio 4:3
(Y) Aspect ratio (Y): 4 !

(O) TT/LP/List (0/2/+1): 0 ! 0 = min. output on screen
! 1 = detailed output on screen
! 2 = min. output on screen + printer
! 3 = max. output on screen + printer

Output file [1/0] ? 1! writes vector addresses to
Output file: BELVEC! a file called BELVEC.DAT

ASEA

Lists helpful information on the ASEA - DEC connection.

SEMANTICS

Technical details comprising program structure, input / output connections and information code are listed in OPTIONS below.

OPTIONS

Register addresses (Micro PDP-11 : parallel port)

Output from LSI to Robot 176772 (LSIROB)
Input from Robot to LSI 176774 (ROBLSI)
CAMERA Select new camera input channel

SYNTAX: CAMERA or RUN [device:]CAMERA

SEMANTICS
Accepts video input including sync information from the alternative input channel. Useful for capturing images at a different sync rate than displaying.

EXAMPLES:
CAMERA ! Selects camera input 1, if channel 0 was in use.
CAMERA ! Selects camera input 0, if channel 1 was in use.

CCHAIN Converts chain code into crack code and creates a data file.

SYNTAX: CCHAIN or RUN [device:]CCHAIN
Chain code file: filespecs

SEMANTICS
Reads a file of a 4 direction code (crack code) and converts it to an 8 direction chain code. Input files must be of a given format. VISION generates chain code files in a suitable format. The generated output file has the extension *.CHC

CCONV Converts chain code into crack code or vice versa

SYNTAX: CCONV or RUN [device:]CCONV
Chain code file: filespecs

SEMANTICS
Reads a file of boundary codes and converts to either 4 direction chaincode or 8 direction chain code. Input files must be of a certain format.
VISION and CCHAIN generate chain code files in a suitable format.

ERASE Clears the frame store to a predefined grey-level value

SYNTAX: ERASE Grey-level-value

SEMANTICS
Sets all pixel values to the specified grey-level on the entire frame store [default = 0].

OPTION: Grey-level specification in the range of 0 to 15
EXAMPLES:
ERASE  Clears frame store to an grey level of 0 (black)
ERASE 7  Clears frame store to an grey level of 7 (grey)

ERFUNC  Generates length approximation error list for digitized straight
        lines with variable orientation (slope angle), different pixel shapes,
        pixel configurations, and aspect ratio.
SYNTAX:  ERFUNC or RUN ERFUNC

SEMANTICS
Generates an error function for estimating the length of a straight line for square,
rectangular or hexagonal pixels in a regular array configuration or a triplet configuration.
Available code directions can range from 8 to 800. The program also generates a detailed
listing if desired. The listing contains average values for each section of the error function
as well as function coefficients, minima and maximum error values in percent. Additional
graph plotting is available for a REGIS compatible monitor such as a DEC GIGI or a
VT240. The plot data can be logged optional onto a plot file.

OPTIONS
- Pixelshape, pixel configuration
- Code set (ranging from 8 to 800 directions)
- Detailed file output
- Plotting of the function with a VT240 or a GIGI computer.
- Plot data logging.

EXAMPLE:
ERFUNC
Pixel shape :  square (S)
              rectangular (R)
              hexagonal (H)
              square triplets (U)
              rec. triplets (T) ? R
Min nr of code directions [8...800] ? 8
Max nr of code directions [8...800] ? 8
Detailed Listing [Y/N] ? N
File output [Yes / No] ? N
Plot graph [Y/N] ? Y  
File plot data [Y/N] ? N

This selection generates deviation data for 8 direction chain code and plots the result on the graphic output device.

---

**ERFCON**

Error function correlation - in order to generate error profiles for simple geometric shapes (circle, semi-circle, triangle, square and rectangle)

---

**FREEZE**

Stores an image into the frame store from the activated input channel/camera

**SYNTAX:**  
FREEZE

**SEMANTICS**

The image input signal from the selected channel (CAMERA) is digitized and stored in the image buffer. It over-writes any previously stored information.

---

**GRAB**

Writes images continuously to the frame store

**SYNTAX:**  
GRAB

**SEMANTICS**

This hardware function sets the digitizer into continuous mode. Images are continuously digitized in real-time from the selected input channel (CAMERA) and writes to the image buffer. Naturally, each scan over-writes the previous image information.

---

**HARDWARE**

Lists information on the MATROX image processing boards

**SEMANTICS**

The two MATROX image processing boards are functionally combined. The smaller board contains a real-time digitizer with a resolution of 4 bits per pixel. The A/D converter chip can be replaced by 6-bit or 8-bit flash converters (sockets are already installed). The second board contains all the circuitry for all necessary image signals and is also responsible for synchronization. The image buffer or frame store is also installed on this board. The memory size is 256 by 256 by 4bit which amounts to 32kB frame buffer memory. More details on hardware functions including information on programmable registers such as functions and locations can be found in the MATROX manual.
### OPTIONS

Hardware registers include

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image data register (CDR)</td>
<td>170000</td>
</tr>
<tr>
<td>Command and status register (CMD)</td>
<td>170001</td>
</tr>
<tr>
<td>Column address register (PAD)</td>
<td>170002</td>
</tr>
<tr>
<td>Line address register (LAD)</td>
<td>170003</td>
</tr>
<tr>
<td>Scroll register (SCR)</td>
<td>170004</td>
</tr>
</tbody>
</table>

#### Commands

- Write Bit 15: Clear
- Write Bit 14: Freeze image
- Write Bit 11: Selection of input channel
- Write Bit 10: Selection of input channel

#### Status

- Read Bit 15: Ready
- Read Bit 14: Blanking between image fields
- Read Bit 11: Input channel no active
- Read Bit 10: Input channel no active

**EXAMPLES** For details on how to address and use hardware registers, see supplied Fortran, Pascal or Macro sources AND refer to the DEC software manual (red folders).

---

**RVHELP**

Lists helpful information on how to use this facility

**SYNTAX:**

RVHELP[/option] [topic[/item1/item2...]] or RVHELP *

**SEMANTICS**

- RVHELP * lists the items for which help is available.
- RVHELP HELP lists the HELP text (of which this is a part).
- RVHELP topic lists information on the specific topic only.
- RVHELP topic/item lists the text associated with the specific item under the subtopic OPTIONS.

Items are specific command options.
PRINTER  Prints the RVHELP text on the line printer
TERMINAL (default)  Types the RVHELP text on the terminal

EXAMPLES:
RVHELP ERASE  ! Lists information about the ERASE command
RVHELP/PRINTER IMAPRI  ! Prints information about the IMAPRI command
RVHELP VISION/W  ! Describes the W (window) option for VISION

IMAPRI  Prints an image from the frame store on a DEC LA100 printer

SYNTAX: IMAPRI

SEMANTICS  Prints the image contained in the image buffer to a printer with programmable printer pins for example LA100 printer from DEC.

INROVI  INteractive RObot VIision Program

SYNTAX: INROVI

SEMANTICS
Object recognition for component handling by the ASEA industrial robot. The user selects the object the vision system is searching for. Both object identification and position code are transmitted to the robot. A control unit is used to start the process. It also displays all interactions between robot and vision sensor, including busy/ready signals and object/position codes. Don’t forget to press the red start button after setting up the robot system.

OPTIONS
Object selection by user (function keys 17-19)

INTVIS Interactive Vision Program

SYNTAX: INTVIS

SEMANTICS
Demonstrates the capabilities of remote requested object handling.

OPTIONS
Object selection by user (function keys 17-19)
IOTEST  Performs an input/output test on the communication link between the micro 11 sensor and the robot.

SYNTAX:    IOTEST or    RUN IOTEST or FRUN IOTEST! foreground operation

SEMANTICS: Checks the output channels for the parallel interface

EXAMPLES

IOTEST! runs the iotest as background job - micro 11 busy FRUN IOTEST! runs the iotest as foreground job - micro 11 free the user can run other programs in background -one for the foreground/background monitor RT11FB and up to six for the extended memory monitor RT11XM.

MATROX  Lists helpful information on the MATROX image processing boards.

SEMANTICS

The two Matrox image processing boards contain one frame memory for of 4 bits per pixel. The A/D converter chip can be replaced by 6-bit or 8-bit flash converters (sockets are already installed). The second and larger board contains all the circuitry for all necessary image signals and is also responsible for synchronization. The image buffer or frame store is also installed on this board. The memory size is 256 by 256 by 4bit which amounts to 32kB frame buffer memory.

The pixel data can be addressed via registers. The base address is 170000 (octal). Commands and status of the MATROX boards can be sent to/read from the CMD register (see list below).

OPTIONS

Hardware registers include

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image data register</td>
<td>(CDR) 170000</td>
</tr>
<tr>
<td>Command and status register</td>
<td>(CMD) 170001</td>
</tr>
<tr>
<td>Column address register</td>
<td>(PAD) 170002</td>
</tr>
<tr>
<td>Line address register</td>
<td>(LAD) 170003</td>
</tr>
<tr>
<td>Scroll register</td>
<td>(SCR) 170004</td>
</tr>
</tbody>
</table>

Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Bit 15 : Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Bit 14 : Freeze image</td>
</tr>
<tr>
<td>Command</td>
<td>Bit 11 : Selection of input channel</td>
</tr>
<tr>
<td>Command</td>
<td>Bit 10 : Selection of input channel</td>
</tr>
</tbody>
</table>
Status

- Read Bit 15: Ready
- Read Bit 14: Blanking between image fields
- Read Bit 11: Input channel no active
- Read Bit 10: Input channel no active

More details on hardware functions including information on programmable registers such as functions and locations can be found in the MATROX manual.

MULTIO Interpretation of up to three objects in the field of view

SYNTAX: MULTIO or RUN MULTIO

SEMANTICS

Demonstrate the capabilities of multiple object recognition. Similar to the InRoVi program but the processing sequence is fixed rather than left to the operator. Objects are picked up from the left to the right (from the furthest to the nearest object).

NEGATE Negates the contents of the image buffer (black to white etc.)

SYNTAX: NEGATE

SEMANTICS: Inverts the content of the entire frame buffer.

ESTPER Generates simple geometric shapes and simulates outline approximation errors using various methods based on chain codes and crack codes.

SYNTAX: ESTPER or RUN [device:]ESTPER ! details see example

SEMANTICS

Perimeter of basic geometric shapes are compared to the results of 6 different methods of estimating the perimeter length of a digitized silhouette. Deviations are displayed on the screen in percent from the theoretical results. The shape can be optionally be drawn into the frame store.

OPTIONS

Aspect ratio Range of shape sizes Circular, semi-circular or square shape Orientation range Centre coordinates Plotting on/off Detailed listing
EXAMPLE

ESTPER
Aspect ratio : 0.75  ! typical aspect ratio (4:3) for a TV camera
Minimum radius : 50  ! initial shape size (50)
Maximum radius : 60  ! up to a specified size of 60 pixels
Radius step size : 5  ! at 5 pixels intervals
Circle (C), Semi-circle (H), Square (S), Rectangle (R)
or Triangle (T) : S  ! selects square shape
MinAngle [0...90] : 0  ! produces squares at orientations ranging
MaxAngle [0...90] : 90 ! from 0 to 90 degrees
Angle step size : 15  ! at 15 degrees intervals
Display [Y/N] : Y    ! generates squares at the different
                    ! orientations on the screen
List details [Y/N] : N ! prints only end results

READ Loads the specified image file into the frame buffer
SYNTAX: READ filespecs  ! default extension .IMG
SEMANTICS
Reads an image file and stores the information in the frame buffer. Image file information must be presented in a compressed form: two pixel values per byte.
EXAMPLE
READ NUTS  ! reads the image file NUTS.IMG from default device

READN Reads the specified image file into the frame buffer
SYNTAX: READN file  ! default extension .IMG
SEMANTICS
Reads image information from the specified file and displays the image.
Image file information must be presented in a non-compressed form: one pixel values per byte. Image file are therefore twice as long (128 blocks or 64 kByte).
EXAMPLE: READN LONG  ! reads image file LONG into the frame buffer
READQ  Reads the specified image file into one quadrant of the buffer
SYNTAX: READN file n ! default extension .IMG

SEMANTICS
Reads image information from the specified file and displays the image in one quadrant specified in the option.
OPTION: n quadrant number 0 to 3 - starting from top left to bottom right.
EXAMPLE:

    READQ BELT 2 ! reads image BELT into the bottom left quadrant of the image buffer

RECALL  Reads the specified image file into one the frame buffer
SYNTAX: RECALL filespecs ! no default extension

SEMANTICS
Reads an image file and stores the information in the frame buffer. Image file information must be presented in a compressed form that is two pixel values per byte.
Same as READ - the only difference is that with RECALL the extension always have to be specified.
EXAMPLE:

    RECALL SY:BELT.DAT ! reads image file BELT.DAT from system device

REDUCE  Reduces the image size to one quarter
SYNTAX: REDUCE n

SEMANTICS
Squeezes the entire image into the specified quadrant (default is top left = quadrant 0).
OPTIONS: n quadrant number 0 to 3 - starting from top left to bottom right.
EXAMPLE:

    REDUCE 1 squeezes the image into quadrant no 1 - top right
ROBVIS  
Robot Vison Program - recognizes objects at certain positions

SYNTAX:  
ROBVIS or RUN [device:]ROBVIS

SEMANTICS
Demonstrates the capabilities of multiple object recognition. Similar to the INROVI program but the processing sequence is fixed rather than left to the operator. Objects are picked up from the left to the right (from the furthest to the nearest object).

SHIFT  
Shifts brightness levels of the image by specified value

SYNTAX:  
SHIFT or RUN [device:]SHIFT

SEMANTICS
Adds the specified constant to each individual pixel grey level with a wrap around effect. Useful for detecting small grey-level differences and for edge detection.

OPTION:  
Constant grey-level.

EXAMPLE:  
SHIFT
Brightness level: 5  
! adds 5 to each pixel value so that pixels with a value of 10 appear bright and all pixel values above 10 are wrapped around and appear black to dark grey.

SLOANG  
Draws polygon and slope angle diagram from a vector list.
Vector data are read from a specified disk file.

SYNTAX:  
SLOANG

SEMANTICS
Produces a slope-angle diagram of a polygon silhouette for object shape recognition. Vector data are read from a file. Suitable input vector files can be generated by AR COP.

OPTIONS
Input and output data file control  
Output plot control incl. colour choice

EXAMPLE:  
SLOANG
Vector input file ? BELVEC  
! reads vector address from BELVEC.DAT
Draw silhouette polygon [T/F] ? T  
! draws object silhouette
Angle adjusted [T/F] ? T ! calibrated angle variation
REGIS commands to GIGI (LP) [T/F] ?! sends plot commands to GIGI
Choose graph colour code [1...7] : 2 ! plots graph in red

---

**SUB**
Subtracts specified image data from content of image buffer

**SYNTAX:**
SUB filespecs

**SEMANTICS**
Performs an image subtraction between the current content of the frame buffer and the specified image data file.

**EXAMPLE:**
SUB BELT.DAT ! subtracts the content of BELT.IMG from the current frame store content

---

**VISION**
Interactive Image processing package

**SYNTAX:**
VISION or RUN VISION

**SEMANTICS**
Physical-device-name is SYstem device for all image file input and output. However, a logical ASsign redirects input and output for images and chain code files.

**OPTIONS:**
- * denotes hardware functions
- # denotes command is executed within a region of the image called window
- ? List of commands available
- @ Indirect command facility
- A[n] Auto mode (repeats n times the commands of an indirect command file)
  - A Execute existing command file
  - L Learn mode (creates a command file and writes commands until end of learn mode )
  - E End of learn / auto mode
- B[n] Brightness-shift # (adds/subtracts n to/from each pixel in other words shifts the grey level up/down by n)
C[n] Camera input line *
    (choose input source line 1 to 4 e.g. camera/recorder/sync. etc.)

D[n] Differences in grey-levels #
    gradient operation (displays vertical and horizontal grey-level changes
    of each pixel above a threshold of n)

E[n] Erase image buffer *
    (clears / over-writes the frame buffer with a constant grey-level n)

F Freeze an image *
    (stores it in the frame buffer)

G Continuous grabbing mode *
    (over-writes frame buffer with every input scan)

H Histogram function #
    (displays the frequency of grey-levels within the whole image or
    within a specified region of the image called window)

H[+n] Line Histogram -
    (displays the frequency of grey-levels out of the n-th line - if 0 ≤ n ≤ 256)

H[-n] Column Histogram -
    (displays the frequency of grey-levels out of the n-th line, providing
    n is negative otherwise it is treated as line-histogram

I[n] Image comparison
    compares the image stored in the frame buffer with a specified one
    from disc and displays different grey-levels between them providing
    the difference is more than the threshold n)

J[m,n] Line measurement
    (alters pixel values to white if grey-level is above threshold n
    or else to black - valid in line m only.

J[m,-n] Column measurement
    alters pixel values to white if grey-level is above threshold n and
    to black in all other cases - executed in column m only.
K  Kiwifruit profile measure mode
    repeats processing commands of an indirect command file for each fruit
    (3 views each) and generates automatically a data-file containing
    size (area), perimeter, weight (keyed in), length and width of the fruit
    and the shape as a 4-direction chain-code (variable length).

L[n]  Level reduction #
    clears the [4-n] least significant bits of grey level

M  Centre (Middle) of area
    displays the centre of an object with a cross after thresholding only

N  Negate image # (inverts all grey levels)

O  Mirrors image #
    S  Exchanges sides (vertical mirror)
    U  Swabs bottom and top (horizontal mirror)
    X  Image turn around (45 degree mirror) - whole image only -

P[n]  Truncates lower grey-levels #
    (writes "black" to pixels with a grey-level below n)

Q[n]  Truncates upper grey-levels #
    (writes "white" to pixels with a grey-level above n)

R  Recall image from disk

S  Store image on disk

T[n]  Threshold the image by level n #

U  Pause (only valid for indirect command file)

V[n]  Scroll image n lines * (hardware scroll)

W  Window mode with options as follows:
    C  show current window limits (draws a frame)
    E  erases a window frame drawn
    L  set limits via line and column number as follows
    M  back to main menu
    N  set new window limits using cursor and arrow keys
    (upper left corner first, followed by Return and then lower right corner)
P  show previous window limits (draws a frame)
R  reset window limits back to 0,255,0,255
S  swap window limits using one out two window buffers
     (erases the old and draws a new frame)
?  prints option list again

X  Traces object outline -
     (executes a window setting and a local thresholding which is displayed
      with level 8 (default) after tracing and displays perimeter, centre of area,
      size, length and width of area)
X[n] same as above - but with threshold level n and no display of the
     threshold operation (faster)
X[-n] same as above - but with medium threshold
     no area calculation -- very fast!

Y  Write chain-code file incl. parameters calculated with the X command
    to disc

Z  Read chain-code from disc and display outline

^Z End of program (exit to monitor)

notes:
* hardware functions   # apply only within a region of the image called window if set

IMAGE DATA file structure
Fixed length non-ASCII file containing brightness information of each pixel of an 256 x 256
image (65,536 values) in a packed form (2 values per byte). File length : 64 blocks = 32kB.
CHAIN-CODE DATA file structure

Variable length ASCII file
1.) Filename   (up to 10 ASCII characters)
2.) Object area (4:3 aspect ratio corrected)
3.) Perimeter  (number of array code elements)
4.) Perimeter  (4:3 aspect ratio corrected)
5.) Weight     (optional for kiwifruit application)
6.) Length (4:3 aspect ratio corrected)
7.) Width (maximum x dimension)
8.) Line number of start location of chain code
9.) Column number of start location of chain code
10 to ... code elements (ASCII Characters)

COMMAND FILE structure

All interactive commands described above can be used in an indirect command file with the same command syntax. A command is an ASCII character followed by one or two Integer numbers (optional - according to appropriate commands) command file from column 20 onwards.

EXAMPLES of VISION Commands in an indirect command file:

W       enter window mode
L       pre-set window limits to
100,200,80,120 lines from 100 to 200 - columns from 80 to 120
S       swap window limits to previous
L       and set new window limits to
50,150,80,120 lines from 50 to 150 - columns from 80 to 120
M       back to main menu again
F       freeze an image
T7      local threshold level 7
M       display center of area (within the current window)
X-9     quick tracing at level 9
Y       write chain-code of outline onto disk
SHAPE1 in file SHAPE1.DAT
W       enter window mode again
S       swap window limits to previous
M       back to main menu again
X9      trace object within window frame at local level 9
Y       write chain-code of outline onto disk
SHAPE2 in file SHAPE2.DAT
U       pause - and place new objects

The indirect command file @INDIR.COM demonstrates some of the capabilities in more detail.
WRITE          Writes content of image buffer to disk in compressed form
SYNTAX:       WRITE filespecs = ! default extension .IMG

SEMANTICS
Writes the entire contents of the frame store into the specified disk file in condensed form:
  2 pixel values per byte.
EXAMPLE:
    WRITE new ! writes frame buffer content into disk file NEW.IMG

WRITEN         Writes content of image buffer to disk in non-condensed form
SYNTAX:       WRITE filespecs = ! default extension .IMG

SEMANTICS
Writes the entire contents of the frame store into the specified disk file in non-compressed
  form: 1 pixel value per byte. Output file are twice as long (128 blocks or 64 kBytes) as image
  files created by the WRITE command.
EXAMPLE:
    WRITEN LONG ! writes frame buffer content into file LONG.IMG
APPENDIX D

ROBOT - SENSOR COMMUNICATION AND PROTOCOL

Vision-Aided Flexible Component Handling System

(as demonstrated at ROBHANZ Conference, University of Auckland, 20/21 Nov 1986)

Robot Program Hierarchy

List of Sensor - Robot Commands and Information Codes

Robot Command Codes to the vision sensor & multi-function unit
Robot Communication

<table>
<thead>
<tr>
<th>Robot-output</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 4 3 2 1</td>
<td>Description</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>Robot control not available / not working</td>
</tr>
</tbody>
</table>

A) Mode Indication

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0</td>
<td>16 Demonstration mode - robot not communicating</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>3 Sensor demonstration request</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>1 Position calibration mode - calibration request</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>2 Robot vision mode</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>4 RV with interactive object request (user/sensor/machine)</td>
</tr>
</tbody>
</table>

^ calibration mode
^ robot vision
^ interactive robot vision mode

B) Data Request to Sensor

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 0</td>
<td>2 Position request in interactive robot vision mode</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>4 Command request in remote control mode</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>8 Object request in both robot vision modes</td>
</tr>
</tbody>
</table>

^ object no. request
^ position request
^ remote command request

Synchronization between robot and sensor

By entering the selection mode (menu) the robot controller transmits a sequence of two code words to the visual sensor (15 followed by 0). The sensor is then ready to take "orders". The next code word (0) sent to the sensor is interpreted as a mode selection. The sensor replies with a busy signal to allow time for loading of the requested program. The busy signal is switched off again as soon as the program is ready. The following request e.g. "analyse image", invokes the image analysis. The result is available and valid, after the busy flag is off again.

The positive going edge of input signals (i.e. processing requests) activates the sensor which indicates its activity by a busy flag. The negative going edge of the busy signal indicates that results from the sensor are available and valid. The negative going edge of any data request (sensor input) indicates that result data are read and no longer needed. This causes the sensor to clear its output lines, and resume program execution.
Sensor -- Robot & Multi-function unit

Sensor- Numeric output  Value

4 3 2 1  Description

A) Object Code - Robot Vision Mode (RobVis / Multio)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>no object recognized - try new object request</td>
</tr>
<tr>
<td>0001</td>
<td>Object 1 recognized</td>
</tr>
<tr>
<td>0010</td>
<td>Object 2 recognized</td>
</tr>
<tr>
<td>0011</td>
<td>Object 3 recognized</td>
</tr>
<tr>
<td>0100</td>
<td>Object 4 recognized</td>
</tr>
<tr>
<td>0101</td>
<td>Object 5 recognized</td>
</tr>
<tr>
<td>0110</td>
<td>Object 6 recognized</td>
</tr>
<tr>
<td>0111</td>
<td>Object 7 recognized</td>
</tr>
</tbody>
</table>

^ small object
^ medium object
^ large object

B) Position Code - Interactive Robot Vision Mode (INROVI)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Requested object not found</td>
</tr>
<tr>
<td>0001</td>
<td>Requested object at location 1 (left)</td>
</tr>
<tr>
<td>0010</td>
<td>Requested object at location 2 (centre)</td>
</tr>
<tr>
<td>0100</td>
<td>Requested object at location 3 (right)</td>
</tr>
</tbody>
</table>

^ location 1
^ location 2
^ location 3

General

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Object not found / invalid position request / Sensor not working</td>
</tr>
<tr>
<td>1000</td>
<td>Sensor busy</td>
</tr>
<tr>
<td>1111</td>
<td>Sensor not available</td>
</tr>
</tbody>
</table>
Control unit -- Robot

<table>
<thead>
<tr>
<th>Robot-</th>
<th>Numeric input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1</td>
<td>1</td>
<td>Calibration mode</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>2</td>
<td>Robot vision mode</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>4</td>
<td>Robot vision mode with user/machine request</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>8</td>
<td>Demonstration mode</td>
</tr>
</tbody>
</table>

Control unit -- Sensor

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Numeric input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1</td>
<td>1</td>
<td>Request for small object (blue)</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>2</td>
<td>Request for medium object (yellow)</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>4</td>
<td>Request for large object (Green)</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>8</td>
<td>Manual recognition request</td>
</tr>
</tbody>
</table>

Registers and position locations used

R1 Counter for object 1
R2 Counter for object 2
R3 Counter for object 3
Indirect location addresses
R9 Position code
Pick-up position at light table (11 = far, 12 = centre, 13 = close)

R8 Position code
Screwing position at plate (31 = bottom (small diameter),
32 = centre (medium diameter)
33 = top (large diameter)

POSITION LOCATION 0 Reference location

Pick-up positions

POSITION LOCATION 10 Intermediate position close to table
POSITION LOCATION 11 Pre-pick-up position (far end - light table)
POSITION LOCATION 12 Pre-pick-up position (centre light table)
POSITION LOCATION 13 Pre-pick-up position (close light table)

Storage positions

POSITION LOCATION 20 Intermediate position close to storage
POSITION LOCATION 21 Pre-place position (left storage)
POSITION LOCATION 22 Pre-place position (centre storage)
POSITION LOCATION 23 Pre-place position (right storage)
Screwing positions

POSITION LOCATION 30 Intermediate position close to screw-plate
POSITION LOCATION 31 Pre-screw position (left screw location)
POSITION LOCATION 32 Pre-screw position (centre screw location)
POSITION LOCATION 33 Pre-screw position (right screw location)

Process Options and Sequence

A) Menu Identification
   Wait for 0 all input lines are set (negative)
   Wait for 15 all input lines are reset (negative)
   Ready for selection by control unit

B) Identification of Selected Mode (receive negative logic)
   1110 "-2" : Calibration request
       - send busy signal
       - start image analysis routine
       - object identification request
       - send busy signal
       - analyse scene
       - send object code

   1110 "-3" : Robot Vision request
       - send busy signal
       - start image analysis routine
       - object identification request
       - send busy signal
       - analyse scene
       - send object code

   1011 "-5" : Interactive Robot Vision request
       - send busy signal
       - start interactive image analysis

   1101 "-3" : object location request
       - send busy signal
       - wait for interactive object request
       - analyse scene
       - send location code

   0111 "-9"
       - object identification request
       - send object code

   1100 "-4" : Demonstration request
       - send busy signal
       - start image processing demonstration

       0000 / 1111 ready for Menu identification
Vision-aided flexible component handling

In many manufacturing situations where automation is applied to boost productivity, flexible component handling plays an essential role in reducing "non-productive" times. Robotic devices can be seen as one of the key devices towards flexible manufacturing. The main advantage of industrial robots over traditional means of automation is programmability, hence flexibility. Robots can perform arbitrary sequences of pre-stored motions, motions guided by sensors, or motions resulting from sensory input. In this, vision and force are the two most important "senses", as they provide necessary information of external states.

This paper, prepared by W. FRIEDRICH, scientist, AIDD, DSIR, Auckland (formerly research fellow, Department of Mechanical Engineering, University of Auckland) and G. ARNDT, associate professor, Department of Mechanical Engineering, University of Auckland, presents methods which enable a robot system to "see" objects using a sensor system which identifies and locates parts. The principles of part recognition applied to robot vision are discussed briefly. A new method of object recognition is presented. Emphasis is laid on experimental results and robot-sensor communication.

Introduction

This paper summarises research work into "Flexibly-integrated component handling systems" (FICHS), performed over the past three years. It is a logical follow-on from a previous survey and study on "component handling in New Zealand manufacturing industry", which among other things, showed the need for a demonstration unit for evaluation of "appropriate" component handling techniques, including a sensor-assisted industrial robot. Recent developments in sensor technology, in particular video-based sensors, promise a much wider application area. New Zealand industry, where medium to small batch sizes or one-off production is most common, offers a large potential for sensor-aidsed systems.

As part of the above project, a survey on image processing activities in New Zealand and on image processing systems available was performed, supported by an industry questionnaire.

The main findings of this survey emphasised the need for information exchange and awareness for modern handling devices, rather than the need for advanced sensor assistance. Today, two years later, the survey results are perhaps not as relevant as then (only 6 robots were operating in New Zealand when, compared to over 30 robots installed now) but the above still applies. Predictions of the robot population in 1983 and the number of installed robots today indicate a similar trend to that seen overseas. With the increasing number of sensor-assisted robot installations, the need arises to gain practical experience with such systems, in order to translate properties and features into the specific needs for New Zealand industry.

The vast majority of current industrial robot applications are using internal control only, without significant external sensing. Instead, the environment is designed to eliminate and compensate all possible uncertainties. This approach requires large investments in design time and special-purpose equipment for each new application. Much of the potential versatility of robots is wasted under these conditions.

In contrast, sensing enables robots and other machines to deal with environmental uncertainties — hence contributing to increased flexibility and relaxed requirements for tooling, e.g., much simpler fixtures or none at all may be necessary. Ultimately, sensors could help to reduce the programming effort necessary, in that coarse positional accuracy can be corrected by sensors during operation. Thus, automatic program generation or off-line programming becomes more feasible.

Sensors may, for example, be used to identify the position of objects, or to inspect parts. However, the most common use of sensory data in existing systems is to initiate or terminate motions in order to either synchronise robots with peripherals or to search ("stop on signal") for an edge of an inaccurately located part. This binary sensor data can be used to choose alternative actions in the robot program. Even this kind of simple sensing is very useful e.g. unloading pallets, where search operations can be performed to identify the pallet location. Individual searches for each part supply offset positions mapped into the pallet space for subsequent gripping motions, thus allowing partially filled pallets to be processed.

Advanced sensors can provide information on object location or identity, on distance, or other measurements such as force. This kind of information enables the robot system to respond in an intelligent way on the basis of external states or conditions.

In many applications, particularly component handling and assembly operations, touch and sight provide the most valuable information. Gripping may for example be upgraded by implementing strategies such as "stop on force". The most versatile information, however, can be obtained using vision for path planning. Measurements can be taken prior to starting the motion and a decision to examine the next object can be made immediately after the results are obtained from the image sensor. Thus, handling time can be substantially reduced — provided the robot system is capable of interfacing, and of processing symbolic data at this level.

The potential for the application of robots can be extended enormously by sensor assistance, in particular in situations where batch lots are small and parts are often changed. A large body of literature on robotic sensors exists.

Processing of visual information

Again, extensive literature exists on this topic. Hence, the general objectives and principles of vision sensing and object recognition will only be briefly summarised here, before dealing with a new approach to object shape analysis.

In robotic applications, visual image processing is intended primarily to inspect, locate, and recognise objects. These images can be divided basically into three categories of increasing complexity:

1. The recognition of multiple, but complete and isolated objects is commercially well established, providing objects are easily separable by intensity, requiring excellent illumination.

2. Images containing parts of an object or overlapping parts require careful exami-
ination. Very few systems are as yet capable of dealing with this category. 3. Scenes with shadows and changing or poor lighting are the most difficult ones to deal with, in particular with the tight time constraints in an industrial environment.

The basic components of a robot vision system are a sensor, a digitiser, a complete processor, and a robot system including peripherals. The sensor, most commonly a TV camera (solid state type), captures an image of the object. Digitisation takes place during the image acquisition scan.

Image information is stored in the computer memory for further processing, either as a “pixel” array, whose values are proportional to the light intensity for the individual picture elements (pixels), or as a position table of objects/background transitions — so-called binary run-length coding. This process is already a segmentation discriminating the object(s) from the background, by so-called global thresholding. A suitable threshold can be chosen automatically, either by measuring the light intensity and adjusting the threshold accordingly, or by histogram evaluation.

In some cases, however, it is sufficient to examine object edges. Two common approaches are applied to find edges: contour tracing and edge detection. Outline following is applied commercially since it is a quick way of determining the silhouette of an object after an initial edge pixel is detected. The next generation of industrial vision systems will probably employ edge detection applied to grey-level images, since classical edge detectors lack robustness.

A feature extractor is commonly applied for identifying significant object properties, such as statistical measures or shape features. The derived feature set is then compared and matched against the corresponding values for a finite set of reference models. The final decision is mostly based on the “nearest neighbour” principle, in which the option with the smallest “Euclidean distance” — within a tolerance — to the reference feature set is chosen.

The spatial sensor is usually activated at the request of the robot system. Object identity, object position, and orientation are then transmitted to the robot control, where a final decision is made.

There are generally two approaches to the recognition of objects as they rotate in three dimensions. Three-dimensional recognition of objects is still in an early stage of research, mainly because of difficulties in 3-dimensional segmentation procedures, which have to deal with 3D overlapping, size and distance estimation, and shadows, to name but a few. Although 3-dimensional recognition is required in robot vision, 2-dimensional recognition, to detect the stable state of the object is sufficient in most applications. By knowing its orientation, a part may then be handled to suit the task requirement.

A New Approach

A new method for quick and low-cost shape analysis has been investigated as part of the FICHS project. It is based on the fact that, basically, sides and their enclosed angles are essential to recognise an object. Boundary segments in the form of a polygon contain exactly these two parameters, and therefore provide a suitable presentation of the silhouette. A silhouette polygon may best be described by a series of vectors, containing their length and angle.

Many techniques of generating polygonal approximation to “line drawings” or plane curves exist. The resulting polygon is often not contiguous — unless so-called end-point fitting methods are used, in which a straight line, formed by two boundary pixels, is divided into new segments if the maximum distance between the straight line and the original outline exceeds a pre-selected threshold.

In machine vision applications, where time is a prime consideration, very fast (hence simple) algorithms are needed. One such fast polygonal shape approximation system will now be presented.

Segmentation by tracing produces either a relative direction code (chain-code), or a position array of boundary pixels, which can be grouped together to achieve a more economical description. Given such a “raw” outline, an algorithm, called the Arc Operator, reduces the shape information to a minimum of vectors through the Distance Functions and the Running Distance Total.

For the former, consider a straight line connection of two boundary pixels as a basis to determine the rate of curvature. The distance from the centre of this straight line

Table 1: Sensor data and type at various levels of complexity.

<table>
<thead>
<tr>
<th>Control Level</th>
<th>Application</th>
<th>Typical Location of sensor</th>
<th>Type of data</th>
<th>Time Intervals</th>
<th>Control action</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>adaptive control e.g, welding, deburring, jet cutting</td>
<td>Sensor usually built into the gripper — moving —</td>
<td>Distance, deviation, force to correct 3-dim. motion</td>
<td>10 ... 100 per sec.</td>
<td>continuous correction of velocity direction of motion</td>
<td>closed loop</td>
</tr>
<tr>
<td>Sensor</td>
<td>robot vision e.g. handling, assembly, etc.</td>
<td>Sensor position stationary — also distributed with the process</td>
<td>object — identity, position, orientation stable state</td>
<td>0.1 ... 1 per sec</td>
<td>relative/absolute position/orientation update</td>
<td>open loop</td>
</tr>
<tr>
<td>Sensor</td>
<td>robot vision or robot control e.g. assembly, handling</td>
<td>Sensor position stationary — close to the process</td>
<td>new task no. assembly part 2, use screwdrive</td>
<td>0.1 ... 10 per hour</td>
<td>e.g., tool/gripper selection</td>
<td>open loop or no feedback</td>
</tr>
<tr>
<td>Overall Control</td>
<td>Batch change, new set-up</td>
<td>Sensor position, remote from process</td>
<td>e.g., complete “raw” program</td>
<td>once/day a month</td>
<td></td>
<td>open loop or no feedback</td>
</tr>
</tbody>
</table>
Continued from page 17.

to the original outline is a measure of curvature. Low curvature parts or circular arcs cannot be detected by this method because the distance function would be constant over such a segment. Therefore, a running total (the latter measure) is introduced which will generate an "artificial corner" by exceeding a certain threshold. Combining both methods yields a fast algorithm for the generation of silhouette vectors. Figure 1 shows a polygon approximation of a metallic workpiece and a kiwi-fruit to illustrate this.

The template and object boundaries are represented in two domains: the original cartesian space, and an arc-angle space also called "slope angle-arc length space". Matching of boundary segments is more efficient in the arc-angle space, essentially because of two major properties of this representation:

1. The angle-arc rotation is added to each individual segment or side angle rotation in cartesian space translates simply into an angle offset in the arc-angle domain.

2. Size in cartesian coordinates translates into a zooming factor for the length of segments, whereby angles are constant. In other words, a length of a boundary segment is proportional to the size and therefore to the distance of the object. Figure 2 shows the rotational properties of the arc-angle domain.

Recognition of an object is successful if the segment/size ratios of the boundary and the angular changes from one segment to the next are equal. A matching operation therefore provides an "angle of match", which is exactly the angle the object is rotated by (refer to Figure 2). Thus, the orientation of the object is a "byproduct" of the recognition procedure. Another "byproduct" of recognition is distance. Once the visual sensor is calibrated (e.g. by showing a reference object, at a given distance to the system), the actual object distance can be determined by the perimeter ratio or average ratio of segment length from both reference and current silhouette.

With the information about object type, location, orientation and distance, the robot may then be instructed to grip the object successfully. Small positional errors can be handled by other sensors built into the hand of the robot, such as proximity and force sensors.

Information flow

Sensors are used to identify the position of parts, to recognise and inspect parts, as well as to detect errors during manufacturing or assembly operations. Sensor data is required at various time intervals and different levels of complexity. Table 1 shows a classification of sensor levels.

Two key requirements are placed on robot programming systems. First, the robot system has to provide general input and output mechanisms for acquiring sensor data. Second, to provide a versatile control mechanism (e.g. force control), for using sensory data to determine robot motions. This need to specify parameters for sensor-based motions and to specify alternative actions on sensory conditions is the prime motivation for using advanced robot programming languages.

Processing simple binary sensor signals differs significantly in computing complex-
up location) and a screwplate and bin as end positions.

The TV camera monitors the pick-up area. Image analysis is activated on request by the robot control. Subsequently, objects are analysed using shape matching. Location and object identity are transmitted to the robot. If the object is then picked up from the light bench and either screwed onto a fitting bolt, or sorted.

The 5-axis robot is equipped with a versatile pneumatic gripper featuring interchangeable and adjustable fingers. For simplicity, the robot's wrist movement is chosen to perform the gripping operation, although in industrial applications a screwing head would be used.

The TV camera delivers the scene image (which is stored in a frame store) for further processing. The object silhouette is extracted by a contour follower. Corners and breakpoints are produced using the Arc Operator principle. The outline, based on a polygon, is matched against a set of reference models. Results are displayed on a graphic monitor and sent to the robot upon request.

The operation mode of, and some interactions with, the process can be directed from a small control box by the operator. As an operator interface, the control box allows one to select various system options, and displays the system state and robot/sensor data.

Alternative pre-programmed movements can be performed by the robot, based on input states/information. The current hardware/software option allows communication to external devices only on a single bit level, but optional firmware will be installed shortly to allow full data exchange with a host computer. Despite the present limitations in communication, commands to the sensor, such as calibration or image analysis, are given by the robot control, and information such as object identity and pick-up position are received and processed. Since positional information cannot be processed directly, pickup locations are indexed.

Assisted by the vision sensor, the industrial robot is the heart of the system and controls the overall process. Input and output lines are divided into sensor I/O and I/O for manual control. Operator requests, e.g. mode selection via control box, are processed by the robot control system, and commands are generated to activate the sensor system. A request/busy line synchronises the robot/sensor data transfer. The sensor information is used to decide on both the gripping position and the handling procedure.

The process is divided into three different operation modes, viz. calibration, automatic robot vision and interactive robot vision.

In the calibration mode all handling positions are verified by the operator to allow a new setup to be mapped into the program space. At the same time, the vision sensor is also in the calibration mode. A reference pattern is shown to the system to identify size (distance) and orientation.

In the automatic robot vision mode, objects are analysed and handled according to a predefined position sequence. In other words, only object identity is passed between sensor and robot control, because the pick-up sequence is fixed. A missing object causes the system to skip the position and issue a new request for object analysis.

![Figure 4. Rapid object position test.](image)

The interactive robot vision mode is a direct simulation of a manufacturing situation where several objects are required by various machines at different times due for example to different cycle times. In this set-up the robot can request an object via keyboard input (softkeys) or control box input. The request is handled by the image processing system and the results of the analysis (identity and position) are transmitted to the robot for further actions. In cases where the object required is not recognised, an error message is issued and a new input request is made.

Optical presence checks are simplified because fixed pick-up positions are required. Processing of picture data is concentrated in each case on only a small area surrounding the area of interest. Analyzing just one centre in the line of each area of interest already yields a number of results (Figure 4).

**Conclusion**

Flexible component handling requires sensing capabilities, with vision being perhaps the most important sense. The application potential for sensors arises in particular in situations where automatic systems have to address a variety of components. The Arc Operator principle has the potential to overcome some limitations of commercial systems. Identification of objects is achieved by matching silhouette segments in the angular domain. The shape segments are generated by an arc-operator applied to a 'raw' outline description. This approach allows identification of position, orientation and relative distance of the object.

Despite severe limitations in communication, the experimental system is capable of instructing the vision sensor, and of processing symbolic data. Within the framework of an artificial level of automation, the flexible approach for dealing with sensory data, as described, can lead to very efficient and successful implementations.

**Acknowledgement**

The financial assistance towards this work given by the DSIR and the NRAC is gratefully acknowledged.

**References**

A New Silhouette-Related Algorithm for Improving Vision-Aided Robot Applications

W. E. Friedrich — Submitted by G. Arndt (1)
Received on January 15, 1988

A new algorithm for robot vision applications is presented which relies on an error analysis of chain code approximations. Silhouette information of objects presented is obtained from simple calculations with a high potential for hardware implementation.

Experiments with simple geometric shapes are discussed. The experimental setup includes an industrial robot and a basic robot vision system developed by the authors, and demonstrates flexible component handling aided by vision sensing using the new algorithm.

KEYWORDS: Robot vision, chain codes, silhouette approximation

1. INTRODUCTION

Whilst vision-based recognition systems have been developed to highly sophisticated levels, the need for simplified and fast algorithms for real-time industrial applications remains. Although investigations relating to this topic have been reported widely [e.g. 1, 2], the majority of relevant papers has been published elsewhere (cf. section 6: references). This paper presents a contribution towards fast object recognition developed at the University of Auckland.

Most current industrial robot applications use internal control only, without significant external sensing. Instead, the environment is designed to eliminate and/or compensate for all possible uncertainties. Much of the potential versatility of robots is wasted under these conditions. In contrast, sensing enables robots and other machines to deal with environmental uncertainties, hence to contribute to increase flexibility and relax requirements for tooling.

The most common use of sensory data in existing systems is to initiate or terminate motions in order to either synchronize the motion of the robot with peripherals, or to search ("stop on signal") for an edge of an inaccurately located part. Even this kind of simple sensing is very useful in a number of applications. Touch and sight provide the most valuable information, particularly in component handling and assembly operations. Gripping may for example be upgraded by implementing strategies such as "stop on force". The most versatile information, however, can be obtained using a vision sensor. Advanced sensors can provide information on object location or identity, and on distance or geometric measurements. In setups with a stationary camera measurements on the objects are taken prior to starting the motion, in order to take advantage of the relatively long time of a robot movement can be optimized and handling time can be substantially reduced. However, the robot system has to provide the type of interfacing, and the power of processing symbolic data.

1.1 Processing of Visual Information

In robotic applications, visual image processing is intended primarily to inspect, locate, and recognize objects. Those kind of images can be divided basically into three categories of increasing complexity:

1. The recognition of multiple, but complete and isolated objects is commercially well established [3,4]. This approach is easily separable by intensity. This approach requires consistent excellent illumination which is often difficult to achieve in factory environments.

2. Images containing parts of an object or overlapping parts require careful examination. Very few systems are as yet capable of dealing with this category [5].

3. Scenes with shadows and changing or poor lighting are the most difficult ones to deal with, in particular with the tight time constraints in an industrial environment. Processing of images in this category are still in the experimental stage [5].

Generally two approaches to object recognition are common: Objects are distinguished by statistical and global measures, or by local features, i.e. shape properties of the object outline. Recognition based on statistical features utilises for example area, perimeter, compactness, centroid, radius vectors, "fuzzy" numbers for industrial applications. The SRI vision module [6], developed as early as 1975, is the classical example of such a system. Several commercial systems have adopted this method, whereby parameter extraction is realised in both hardware and software. Global features result from properties of an object considered as a whole, whereas local features describe only small parts of an object. Hence, local features can contribute to overcome some of the current limitations of commercially available systems, such as the successful recognition of parts with similar statistics, and the recognition of touching or overlapping parts.

A new approach to a rapid recognition procedure is suggested below, which has the potential for hardware implementation, and hence further speed improvements.

Most machine vision systems applied to recognition, inspection or measurement tasks rely heavily on silhouette information. Many commercial vision systems, often referred to an early binary vision systems, utilise the SRI feature set [6], including boundary-related parameters such as perimeter, compactness, and excursions from the centre of area. Other shape recognition systems require local geometric properties, for example the curvature as a function of arc length. Boundary related parameters are extremely important where object recognition or feature measurement is not limited to isolated fully visible components, but is aimed at recognising overlapping, touching or partially visible objects. The prime consideration here is an on-line estimation of the arc length. Current methods of estimating the arc length or perimeter in digital images are either not very accurate (e.g. chain codes [7]) or more complicated (such as polygon approximations and spline approximation schemes), and are therefore too slow for industrial real-time applications.

2. REVIEW OF ERROR ANALYSIS

This section reviews the results of a new method of analysing errors [8] introduced by the estimation of arc length of digital curves and outlines. The new approach applies to various code sets such as the most common 8 direction chain code and the 4 direction crack code. Moreover, the analysis includes other code sets as well as considering more than one neighbour pixel in each direction. Furthermore, the analysis is not restricted to digitization schemes based on square rasters. In contrast to others [9,10], this new method takes any kind of aspect ratio (rectangular pixel shapes) as well as different pixel forms (e.g. hexagonally-shaped pixels) and various pixel configurations, such as triplets, into account.

The new approach to analyse deviation errors in the estimation of chain coded lines is based on the fact that a digital line is approximated by the two nearest available chain code elements. The error function then expresses the length deviation, introduced by chain coding, as a function of slope angle and pixel configuration such as square, rectangular and hexagonal pixels.
The general form of the error analysis for approximating straight lines with a sloping angle of $\theta_x$ is

$$\text{Err} = A \cos \theta_x \cdot B \sin \theta_x = 1$$  \hspace{1cm} (1)

whereby factor $A = B = 1$ for crack the crack code $A = 1, B = 1/3$ for chain codes with rectangular pixels and $A = 1, B = 1/\sqrt{3}$ for hexagonal pixels.

Most other methods for estimating the length of digital lines rely on 8-direction chain codes or the 4-direction code, where pixels are treated as tiny tiles and the object is traced along the "cracks" rather than from pixel centre to pixel centre. In both cases the outline is divided generally into small line segments reflecting the move from one pixel to the next, along the borderline of the object of interest. The length of a digitized outline is estimated simply by adding the individual lengths of these small segments. It has been shown [10] that the average error in estimating the length of a digital straight line amounts to 27.3% using 4 direction codes and 5.4% using 8 direction chain codes. However, this result only applies to square pixels. The recently established error analysis shows that the pixel shape plays a significant role in estimating the length of digital lines. The average deviation error for 8 direction chain codes rises from 5.4% to 6.1% considering rectangular pixels with an aspect ratio of 4:3.

Figure 1 shows the error function for length estimation of straight lines as a function of its slope angle, considering one nearest element in each direction for square, rectangular (8 direction code) and hexagonal pixels (6 direction code).

Large deviation values arise because of the large angle differences between the direction vectors available, for example in crack coding. These errors are reduced significantly by the introduction of additional direction vectors. The number of available direction vectors is obviously a direct measure for the accuracy of digital line approximation. To quantify, Table 1 summarizes maximum and average percentage errors introduced by applying chain code sets for the length estimation of digital straight lines, for 4 to 24 code directions.

The results for square pixels are identical to those found by Proffitt [10].

![Error function for square, rectangular and hexagonal pixels](image)

Figure 1: Error function for digital length estimation of digital straight lines versus slope angle.

| Table 1: Length estimation errors for chain codes |
|-----------------|-----------------|-----------------|-----------------|--|
|                  | No. of directions | Deviation       |                |                |
| Square pixels    | 4                | 8.2%            | 2.8%           | 1.3%           | Maximum |
|                  | 16               | 5.5%            | 1.4%           | 0.64%          | Average  |
| Rect. pixels     | 41.4%            | 11.8%           | 4.5%           | 2.2%           | Maximum |
|                  | 58.7%            | 6.1%            | 1.7%           | 0.76%          | Average  |

| Pixel shape      | 6                | 12               | 18              | 24             | Deviation |
|------------------|------------------|------------------|-----------------|----------------|
| Hexagon pixels   | 15.5%            | 3.5%             | 1.8%            | 1.0%           | Maximum   |
|                  | 10.3%            | 2.3%             | 1.0%            | 0.59%          | Average   |

Summarizing, it is seen that the average error decreases considerably with an increasing number of directions available to match the slope of the digital line. In fact, the error value reduces exponentially with increasing numbers of available code elements. Regular pixel shapes such as square or hexagonal shapes are beneficial because equally distributed angles result in a lower average deviation.

3. THEORY AND APPLICATION OF APPROXIMATION ALGORITHM.

In the following four well known methods - two for each code category (crack codes and chain codes) - to reduce the approximation errors are briefly described and compared with two newly developed methods [8].

3.1 Methods for Reducing Estimation Errors

One of the most obvious methods [11] in 4 direction coding is to correct the grid unit to an average factor of $5/4$ (0.785). This method works well for circular shaped objects, but fails for most other shapes. The reason is that only a circle provides an even distribution of all possible segment angles to justify the use of an average factor. Another method [12] takes advantage of a finer grid to approximate the object edge more closely. This approach introduces basically a chain code scheme superimposed on the crack code scheme. The disadvantages are similar to the original Freeman code [7]. The original Freeman code is used here as a standard reference to compare the effectiveness of other algorithms. Dempsey et al. [13] suggest a replacement of two neighbouring code elements with a direction change of ±1/4 and assigning these elements new length values. This principle represents for some code combinations a conversion from 4 to 16 direction coding. The error range is reduced significantly.

3.2 The New Algorithm

A new method [8] to correct the crack coded length information is summarized in the following (k) code vector. The length of the current element is adjusted according to the code combinations for straight connections (a), corners (b), steps (c) and turn-arounds (d). Table 2 shows details in numeric form and an example for the improved outline approximation.

<table>
<thead>
<tr>
<th>Table 2: Code combinations and their replacement values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Configuration</td>
</tr>
<tr>
<td>Symbol</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>[a]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>[b]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>[c]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>[d]</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Figure 2: Crack code approximation by Friedrich.**
Figure 2 shows an application of this approximation scheme.

The new method [8] for chain codes works in a similar way. Three neighbouring code elements are considered. By replacing the total length value of these 3 code elements by its “short cut”, this method resembles a combination of 16 and 24 direction codes, with a significant increase in accuracy.

8 directions
16 directions
24 directions

Figure 3: Fishbone pattern for detecting suitable code combinations for an improved length estimation. (One of the four quadrants shown only).

Similar to the method described for the crack codes a moving window or filter can be thought to move over the serial chain code. Each time a valid combination is discovered the replacement length can be looked up in a prepared table adjusted to the pixel shape in use. Or better still: counters for each code combination can be used and only one correction needs to be made once the process is completed.

The approximation and smoothing process can be best compared to the fish bone pattern shown in Figure 2. The code structure applies to each of the four quadrants, so that the segment length results in the values shown in Table 3.

Table 3: Segment length of various code elements compared to the equivalent code combinations of other code sets.

<table>
<thead>
<tr>
<th>Code combination</th>
<th>Length value (square pixels)</th>
<th>8</th>
<th>16</th>
<th>24 directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1.41</td>
<td>1.41</td>
<td>2.23</td>
<td>-</td>
</tr>
<tr>
<td>76</td>
<td>2.41</td>
<td>2.41</td>
<td>3.65</td>
<td>3.61</td>
</tr>
<tr>
<td>776</td>
<td>3.82</td>
<td>3.82</td>
<td>5.65</td>
<td>5.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Length value (rectangular pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1.67</td>
</tr>
<tr>
<td>76</td>
<td>2.67</td>
</tr>
<tr>
<td>776</td>
<td>4.03</td>
</tr>
<tr>
<td>770</td>
<td>4.67</td>
</tr>
</tbody>
</table>

3.3 Experiments with Simple Shapes.

The methods above have been applied to simple geometric shapes, viz. square, semi-circle and circle. A range of silhouettes of each of these shapes was generated by computer, with various parameters such as side length, angular orientation and radius. For each of the above methods the circumstances are calculated step by step as the silhouette is established. The pixel locations Xn and Yn for a circular silhouette are generated by using

\[ X_n = r \cos \varphi_n + X_c \quad \text{for} \quad 0 < \varphi_n < 2\pi \]
\[ Y_n = r \sin \varphi_n + Y_c \]

with \( \varphi_n = n \arctan (l/r) \) and \( X_c, Y_c \) centre coordinates.

Subscript c stands for circle and n for the current pixel position. For each new pixel location the appropriate code element is calculated by each of the methods. For the semi-circular shape, the orientation is an additional parameter taken into account. Given the slope angle \( \psi_c \), the above formula can be applied for the circular part of the shape for the range of

\[ n \psi < \varphi_n < n \psi + \pi \]

The remaining part of the silhouette is a straight line given with

\[ X_n = r \cos \varphi_n + X_c \quad \text{for} \quad n \psi + \pi < \varphi_n < n \psi + 2\pi \]

\[ Y_n = r \sin \varphi_n + Y_c \]

with \( \varphi_n \) being the orientation angle \( \psi_c \) the steplength with \( l_n = 1 \ldots 4 \)

Finally the square silhouette is generated by using the side length 2a and the orientation \( \varphi_n \)

\[ X_n = a \cos \varphi_n + X_c \quad \text{for} \quad 0 < \varphi_n < \pi/2 \]
\[ Y_n = a \sin \varphi_n + Y_c \]

with \( \varphi_n = \varphi_n + (n-1)\pi \) with \( a \ldots 4 \)

\[ l_n = a \]

Results are shown in Figure 4 in form of error ranges taken from the maximum, minimum and average deviation value from a set of generated silhouettes.

![Figure 4: Results of the applied methods as error ranges in percent.](image)

From Figure 4 it can be seen that the Friedrich codes are most accurate in each of the two categories. The method described by Horn only works well for the circular shape, but the deviation for squares is unacceptably high. The results of the Dunkelberger method are very similar to the Freeman coding. The approach by Dessimios shows slightly better results than the crack code corrections by Friedrich. However, the chain code correction method also by Friedrich is the most accurate with error averages below 2%.

4. SYSTEM IMPLEMENTATION

A system was implemented in which the visual sensor is activated at the request of the robot system. Object identity, object position, and orientation are then transmitted to the robot control, where a final decision is made. Because visual sensing is processing-intensive, vision sensors are often implemented in separate processors. Therefore, the robot system must be able to interface with the external system at the level of symbolic data, rather than at the physical level only. Data and parameters have to “flow” in both directions, from and to the controller. Commands or requests may be issued by the control unit and results or data are returned to it. Similar conditions apply in an overall control structure in sending and receiving data. Variable data exchange plays an essential role in a flexible system, whereby programming of sensor devices is a prime factor for flexibility.

4.1 System Integration

For demonstration purposes it was decided to use a selective screwing operation in which up to 7 different nut sizes are handled by the system. 3 of them are screwed onto the appropriate bolt, all others are sorted in pre-allocated positions.

The system comprises the following components as shown in Figure 5 [also c.f. 14]

- An industrial robot (IRB 6/2 - ASEAL)
- Visual sensor equipment including a DEC micro 11 (LSI 11/23+) with special-purpose interface boards and additional graphic hardware to display results of image analysis.
A solid state TV-camera with 256 by 256 pixels to capture scene images.

- A control box to manually interact with the process.
- Peripherals such as a light bench (pick-up location), a screwplate, and boxes as end positions.

ROBOT VISION

Figure 5: The robot vision setup at the University of Auckland.

The TV camera monitors the pick-up area. Image analysis is activated on request by the robot control. Subsequently, objects are analysed using shape matching. Later, the robot's wrist movement was chosen to perform the screwing operation, although in industrial applications a screwing head would be used. The TV camera detects the screwing box, which is stored in a frame store for further processing. The object silhouette is extracted by a contour follower. The chain-coded outline is reduced to a polygon description [14] and matched against a set of reference models. Results are displayed on a graphic monitor and sent to the robot upon request. The operation mode of, and some interactions with, the process can be directed from a small control box by the operator. As an operator interface, the control box allows one to select various system options and displays the system state and robot/sensor data.

4.2 Overall Control Structure

Assisted by the vision sensor, the industrial robot is the heart of the system and controls the overall process. Input and output lines are divided into sensor I/O and control I/O for manual control. Operator requests, e.g., mode selection via control box, are processed by the robot control system, and commands are generated to activate the sensor system. A request/busy line synchronizes the robot/sensor data transfer. The sensor information is used to decide on both the gripping position and the handling procedure. The process is divided into three different operation modes, viz. calibration, automatic robot vision and interactive robot vision.

In the calibration mode all handling positions are verified by the operator to allow a new setup to be mapped into the program space. At the same time the vision sensor is also in the calibration mode. A reference pattern is shown to the system to identify size (distance) and orientation. In the automatic robot vision mode, objects are analyzed and handled according to a predefined position sequence. In other words, only object identity is passed between sensor and robot control, because the pick-up sequence is fixed. A missing object causes the system to skip one position and issue a new request for object analysis. The interactive robot vision mode is a direct simulation of a manufacturing situation, where several objects are required by various machines at different times, due, for example, to different cycle times. In this setup the operator can request an object via keyboard input (software) or control box input. The request is handled by the image processing system and results of the analysis, identity and position, are transmitted to the robot for further actions. In cases where the object required is not recognized, an error message is issued and a new input requested.

5. CONCLUSION

Automatic flexible component handling equipment necessitates sensing capabilities. Vision sensors in many cases satisfy this requirement. Application potential for such sensor applications arises particularly in situations where automatic systems have to address a variety of components.

The paper presents an overall concept towards flexible component handling, with main emphasis on a new and fast algorithm for shape analysis. This algorithm has the potential to overcome some limitations of commercial systems, as has been shown by an in-depth analysis of chain code errors. This error analysis describes the behaviour of estimation errors for chain coded straightlines. The algorithm is described and compared with other common algorithms using simple geometric shapes, with favourable results.

An experimental setup featuring such a vision system is described. Despite severe limitations in communication, the system is capable of instructing the vision sensor, and of processing symbolic data in real time. Within the framework of an appropriate level of automation, the flexible approach of dealing with sensory data as described can lead to very efficient and successful implementations.

6. REFERENCES


Integrated Robot Vision System for Flexible Component Handling

Werner E Friedrich

Auckland Industrial Development Division (AIDD)
Department of Scientific and Industrial Research (DSIR),
P.O. Box 2225, Auckland, New Zealand.

ABSTRACT

This paper presents an application of an experimental robot vision system for flexible component handling. Operational details of system components are described. Emphasis is laid on information processing, system integration and overall control. The functions of the image processing part are described briefly including a summary of a new algorithm for outline approximations of component silhouettes. Advantages of this algorithm include its potential for recognizing overlapping, touching or partially visible objects. The overall system implementation is addressed in more detail, covering synchronization between sensor and robot, operating modes, and control hierarchy.

1. INTRODUCTION

In many manufacturing situations where computerized automation is applied to increase productivity, robotic devices play an essential role in reducing "non-productive" times. This applies particularly to sensor-guided robots. In this, vision is perhaps the most significant sensory capability in flexible automation.

The vast majority of current industrial robot applications use internal control only, without significant external sensing. Instead, the environment is designed to eliminate and compensate for all possible contingencies. This approach requires large investments in design time and special-purpose equipment for each new application. Much of the potential versatility of robots is wasted under these conditions. In contrast, sensing enables robots and other machines to deal with environmental uncertainties, and hence to increase flexibility and relax requirements for tooling, e.g. much simpler fixtures may be necessary. The potential for the application of robots can be extended enormously by sensor-assistance, in particular in situations where batch lots are small and parts are often changed.

Sensors may for example be used to identify the position of objects, or to inspect parts. This kind of information enables the robot system to respond in an intelligent way on the basis of external states or conditions. The most versatile information can be obtained using a vision sensor. In handling applications, where the camera is stationary, image analysis and robot movement can be implemented in parallel. Measurements can be taken prior to starting the robot motion, image analysis can take place during motion, and the next object can be examined immediately after the results are transferred to the robot controller while the robot may still be in motion. Thus, handling time may be substantially reduced without the need for extremely high processing speed on the image analysis side.

2. THE EXPERIMENTAL ROBOT VISION SYSTEM

The requirements for an experimental vision system for robotic applications comprise two categories of processing capabilities. These fundamental requirements are firstly digitization and storage capabilities for an image, and secondly processing routines to extract the required information.

Vision sensors are often implemented in separate processors because visual sensing is processing intensive. Therefore, an additional requirement involves communication to the robotic device. The robot system must be
able to interface with the external system at the physical level as well as at the level of symbolic data. Parameters and data have to passed in both directions. For example, commands or requests may be issued by the control unit and results or data returned to the control unit. Moreover, in many applications sensors need to be programmed.

Programming can be achieved either manually (which may be required in the setup stage only) - or automatically, e.g. via downloading of program information from either the host controller or the robot. Variable data exchange plays an essential role in a flexible system, whereby programming of sensor devices is the prime factor for flexibility, particularly where rapid batch changes are required.

A robot vision system has been implemented by the author in which the visual sensor is activated at the request of the robot controller. The results of an image analysis, part number and position, are then transmitted to the robot controller. For demonstration purposes it was decided to use a selective screwing operation in which up to 7 different nut sizes are handled by the system, 3 of them are screwed onto the appropriate bolt, all others are sorted in a pre-allocated store.

2.1 Hardware Components

The system comprises the following components:

- An industrial robot system (IRb 6/2 by ASEA)
- Visual sensor equipment based on DEC micro 11 with special-purpose interface boards and additional graphic hardware to display the results of the image analysis.
- A video camera to capture scene images.
- A control pendant to manually interact with the process. The pendant is connected to both the robot control and the vision sensor via a parallel interface.
- Peripherals such as an illuminated light bench (pick-up location), a screw-plate, and a component storage.

The image processing part of the system includes two special-purpose interface boards: a real-time frame grabber with a 4bit A/D converter and a frame buffer for storing a 256 by 256 image. The aspect ratio of the frame store corresponds to that of the camera in use (generally 4:3 for standard TV cameras). The shape of each individual pixel becomes rectangular and measurements taken from the image have to be calibrated appropriately.

2.2 Software Components

A universal software package has been developed featuring two kinds of command modes: an interactive mode and an automatic mode. The interactive mode is designed for camera calibration and setup procedures. The automatic mode simulates the integration of robot vision with production machinery.

In the interactive command mode, three different categories of routines are available: pre-processing functions, feature extraction programs and classification routines. These are combined in a software library. Development work has concentrated on implementing essential processing functions within a suitable software structure. Image processing routines are written in assembler for reasons of processing speed. The software frame is mainly implemented in Pascal.

The software package is capable of handling three different types of file I/O: image I/O, chain code I/O and command I/O. These features are only available in local mode, i.e. when the sensor system is not
communicating to the robot system. In order to assist the setup of command files a record facility is provided. Instructions are simultaneously executed and stored in a command file in a readable form (ASCII format) to allow alteration by editing.

The described functions are also available at the DEC operating system level. Moreover, all image processing functions embedded in the vision module are available to user programs in form of a subroutine library. A comprehensive on-line help facility is implemented.

3. OVERALL FUNCTION

The TV camera delivers the scene image, which is stored in a frame store for further processing. Image analysis is activated on request by the robot control. Subsequently, the object silhouette is extracted by a contour follower. A set of simple geometric features is extracted and matched against a library of reference models. Results are displayed on a graphic monitor. Upon request, location and identity of the object are transmitted to the robot. The object is then picked up from the presented location and either screwed onto a fitting bolt, or placed in the appropriate storage.

The 5-axis robot is equipped with a versatile pneumatic gripper, constructed at the University of Auckland, featuring interchangeable and adjustable fingers. For simplicity, the robot's wrist movement is chosen to perform the screwing operation, although in industrial applications a special screwing head would be used instead.

3.1 Shape Recognition

Generally two approaches to shape recognition are common: Objects are distinguished either by global (statistical) measures or by local features, i.e. shape properties of the object outline.

Recognition based on statistical features utilizes for example area, perimeter, compactness, centroid, radius vectors, Euler number and hole parameter. The SRI vision module \(^1\), developed as early as 1975, is the classical example of such a system. Several commercial systems have adopted this method \(^2,3\), whereby parameter extraction is realized in both hardware and software.

Global features result from properties of an object considered as a whole, whereas local features describe only small parts of an object. Hence local feature extraction can help to overcome some of the current limitations of commercially available systems, such as successful distinction between parts having similar statistics and the recognition of touching or overlapping parts.

The ultimate goal of recognizing partially visible shapes relies on a suitable shape description. It has been demonstrated that the length of object sides and their enclosed angles are essential and sufficient for shape recognition \(^4\). Boundary segments in the form of a polygon contain exactly these two parameters, namely segment length and orientation. Such a polygon description of a silhouette contains a list of these two parameters. Perkins \(^5\) presents such a procedure where objects are matched in a slope-angle domain. This approach has since been a popular technique for recognizing silhouette parts with significant identifying features, such as small high curvature parts or long straight lines with a pointed corner at the end \(^6,7,8\).

In machine vision applications, where cycle time is critical, very fast and hence simple algorithms are needed. One such fast polygonal shape approximation is the arc-operator presented here. The arc-operator is a simplified end-point fitting method suitable for the required fast polygon approximation.
3.2 The arc-operator

The new algorithm developed by the author is based on a straight line connection of two boundary pixels. The maximum perpendicular distance of this straight line to the original outline is a measure of curvature. Now consider the straight line segment moving along the boundary where both end points are part of the boundary. The distance values obtained from such a procedure are proportional to the curvature as a function of the boundary location. The distance function can be directly used as an indicator of sharp corners by using an appropriate "corner" threshold with a peak detector.

Low curvature parts or circular arcs cannot be detected by this method, because the distance function would be constant over such a segment. Therefore an integration of the distance function is introduced which basically adds the distance values until a carefully selected "curve" threshold is exceeded and a new breakpoint is defined.

In order to produce an acceptable approximation, corner detection must have priority over curve detection. In other words a breakpoint will be located as accurately as possible at locations with a sharp corner, so that a good synchronization of corners between the polygon description and the original object can be achieved.

![Figure 1: Principle of arc-operator.](image-url)
For real-time applications, all calculations, in particular divisions, should be kept to a minimum. The arc operator is therefore simplified as follows. Let $A$, $B$, and $C$ be the sequential points on the object border at the location $i$ in a code sequence describing the silhouette and let the coordinate values of point $C$ be $x(i)$ and $y(i)$. Points $A$ and $B$ are $l$ pixels away from $C$ in each direction so that $A$ is located $-l$ pixels from $C$ and $B$ is located $+l$ pixels from $C$. The appropriate equations are

\begin{align*}
dx(i) &= x(i) - x(i-l) - \{x(i+1) - x(i)\} = 2x(i) - x(i-l) - x(i+1) \\
dy(i) &= y(i) - y(i-l) - \{y(i+1) - y(i)\} = 2y(i) - y(i-l) - y(i+1)
\end{align*}

where $x(i)$ and $y(i)$ are the boundary locations, $dx(i)$ and $dy(i)$ the $x$ and $y$ components of the distance $d$ which equals

\begin{align*}
d &= dx + dy = 2x(i) + 2y(i) - x(i-l) - y(i-l) - x(i+l) - y(i+l)
\end{align*}

or simplified using iteration

\begin{align*}
d(i) &= x(i) - x(i+l) + y(i) - y(i+l) - d(i-l)
\end{align*}

A typical sample of a processed boundary section is shown in Figure 1.

3.3 Implementation Considerations

The procedure is naturally very noise sensitive. Quantization noise is a major contributor to instability. Therefore, an additional precaution has to be introduced, a deviation point. This is achieved by additionally monitoring the distance measure of a chord from the last reported breakpoint to the current location. In cases where breakpoints are reported too closely to each other, i.e. shorter than the segment length of the operator, a correction measure has to be applied. If a corner point is detected shortly after a curve or deviation point, the last curve position detection will be discounted.

The implementation of this competitive technique of reporting breakpoints presents a satisfactory solution to generate reliable silhouette vectors.

3.3.1 Parameter Selection

In order to achieve reliable results the parameters for the arc operator have to be selected carefully. Obviously, the length $l$ of the operator directly influences the resolution of the approximation. Similarly, a small distance threshold produces a finer distribution of breakpoints. The accumulated distance is a measure of low-curvature segments. The value selection is therefore dependent on the shape properties, e.g. whether shape segments are mainly curved or straight. The third criteria, the deviation tolerance, is mainly intended for shapes with a combination of low-curvature parts and sharp corners. The parameters used in Figure 2 are operator length $l=8$ pixels, distance threshold: 15 pixels, accumulated tolerance: 90 pixels.
Figure 2: Successful polygon approximation applied to a curved metal stamping (seat belt buckle).

3.3.2 Slope angle domain

The template and object boundaries are represented in two domains, in the original cartesian space and in a slope angle space. Matching of boundary segments is more efficient in the slope angle domain (Figure 3), essentially because of two major properties of this representation:

1) Rotation in cartesian space translates simply into an angle offset in the slope angle domain, since the angle of rotation is added to each individual segment or side angle. A correlation procedure quickly reveals the orientation of the object.

2) The object size in cartesian coordinates translates into a zooming factor for the length of segments, whereby angles are constant. In other words, the length of a boundary segment is proportional to the object size. A scaling factor can therefore provide a measure of distance to the object.
4. OVERALL CONTROL STRUCTURE

Assisted by the vision sensor, the industrial robot controller is the heart of the system and is in charge of the overall function. Input and output lines to the controller include sensor I/O and manual control I/O (Figure 3). Operator requests, e.g. mode selection via control box, are processed by the robot control system, and commands are generated to activate the sensor system. A request/busy line synchronizes the robot/sensor data transfer. The sensor information is used to decide on both the gripping position and the handling procedure.

4.1 Modes of Operation

The system operation is divided into three different operation modes, viz. calibration, automatic robot vision and interactive robot vision.
In the calibration mode all handling positions are verified by the operator to allow a new setup to be mapped into the the program space. At the same time the vision sensor is also in the calibration mode. A reference pattern is shown to the system to identify size (distance) and orientation.

In the automatic robot vision mode, objects are analyzed and handled according to a predefined position sequence. In other words, only object identity is passed between sensor and robot control, because the pick-up sequence is fixed. A missing object causes the system to skip one position and issue a new request for object analysis.

The interactive robot vision mode directly simulates a manufacturing situation, where different objects are required by a number of production machines operating at various cycle times. In this setup the operator can request an object via keyboard input (softkeys). The request is handled by the image processing system and results of the analysis - identity and position, are transmitted to the robot for further processing. In cases where the object required is not recognized, an error message is issued and a new input requested.

The operation mode of and some interactions to the handling system can be directed from a control pendant which also displays the system state and robot/sensor data.

4.2 Sensor Communication

Various pre-programmed motions controlled by sensor information can be achieved. The current hardware/software option allows communication to external devices only on a single bit level, but optional firmware will be installed shortly to allow full data exchange with a host computer. Since positional information cannot be processed directly at present, pickup locations are indexed. Despite these limitations in communication, commands to the sensor such as calibrate or analyze image can be issued by the robot control, and the results from the sensor including object identity and pick-up position can be received and processed via the control pendant in a simple binary form.

Additionally, the control pendant is employed to signal the information exchange between robot and sensor
device. It is always possible to recognize the process status by assessing the signal pattern shown. Since all push buttons are illuminated, input and output information are directly related to each other. For example, if the request for image analysis is issued by the robot controller the appropriate push button is illuminated. The operator can issue the same command by pressing this push button. The end result is identical: the vision sensor receives the command code for image analysis.

5. CONCLUSION

Flexible component handling requires sensing capabilities, with vision being perhaps the most important sense. Application potential for sensor applications arise in particular in situations where automatic systems have to address a variety of components.

An overall concept of flexible component handling has been presented with main emphasis on a new approach to shape analysis. This "arc-operator" principle has the potential to overcome some limitations of commercial systems. Identification of objects is achieved by matching silhouette segments in the arc-angle domain. The shape segments are generated by an arc-operator applied to a raw outline description. This approach allows rapid identification of position and orientation of the object.

An experimental setup featuring such a vision system is described. Despite severe limitations in communication, the system is capable of instructing the vision sensor and of processing symbolic data. Within the framework of an appropriate level of automation, the flexible approach to dealing with sensory data as described can lead to very efficient and successful implementations.

6. REFERENCES