Automatic Segmentation Techniques of Profile Measurement for Circle-Line Splining

Diploma Thesis

Christoph-Peter Hinterleitner

Institute for Automation
Department Product Engineering
University of Leoben
Leoben, Austria

June, 2010

Supervisor:

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary
University of Leoben, Austria

In remembrance to my best friend, Thomas Lettner.

I hereby declare that this thesis and the work reported herein was composed and originated entirely by myself, unless stated otherwise.

Leoben, June 2010 Christoph-Peter Hinterleitner

Acknowledgements

First of all, I would like to thank my supervisor Prof. Paul O'Leary for his support and the plenty of useful hints and ideas.

It is a pleasure to thank the team of vatron gmbh who made this thesis possible. Especially I want to thank my workmates Peter Schalk, Ewald Fauster, Ingo Reindl and Ronald Ofner for their mental and technical support as well as for the valuable contributions and the great discussions in the office. It is a pleasure for me to work in this great atmosphere.

Moreover I would like to thank the team of the Institute for Automation, especially Doris Widek and Gerold Probst for their great support on technical things and paperwork.

I am deeply grateful to my parents Ernestine and Walter for all the love, confidence and support in all my life. My present and future success will always be based on my parent's success.

Finally, I want to thank my friends and my roommates for their encouragement and emotional support. Especially, I want to thank Thomas Lettner for the great discussions on the technical issues, as well as Markus Erlachner for the support on any problems and his useful contributions. I also want to thank Simone for her great mental support and motivating words.

Abstract

This thesis addresses techniques for automatic registration and segmentation of 2D geometry measurement data associated with profiled steel tubing. The CAD model consists of a concatenation of many circular arcs and straight line segments, which form the closed profile. The registration and segmentation of the measurement data is a prerequisite to performing a least square fitting of the coupled geometric objects. Prior to fitting the segments of data are associated with specific geometric objects.

The cross-section of the geometric objects is measured by a set of light-sectioning sensing heads arranged around the profile.

In order to derive quantitative information from the measured data, the data points have to be registered and segmented prior to fitting. In the present work, techniques addressing the following tasks are presented: (1) template matching; (2) data segment assignment; (3) data sorting and (4) determination of transition points between the particular segments. Thereby, CAD data representing the ideal geometry is incorporated. The above process yields segmented and sorted data, ready for least square fitting.

All methods are tested and verified with real measurement data, recorded from an automatic measuring system using five light-sectioning heads to capture the cross-section.

Kurzfassung

Diese Arbeit behandelt Techniken für die automatische Segmentierung von 2D-Querschnittsmessdaten von Profilierwalzteilen und deren Zuordnung zu Modelldaten. Das CAD-Modell besteht aus einer Aneinanderreihung von mehreren Kreisbogen- und Geradensegmenten, welche das geschlossene Profil darstellen. Die Segmentierung der Messdaten ist erforderlich, um eine Anpassung der Querschnitte mit der Methode der kleinsten Quadrate durchzuführen. Vor der Anpassung müssen die Semgente den geometrischen Objekten zugeordnet werden.

Die 2D-Querschnittsmessdaten werden mit mehreren Lichtschnittmessköpfen aufgenommen, welche am Umfang der Profilierwalzteile angeordnet sind.

Um quantitative Informationen aus den Messdaten zu erhalten, müssen die Datenpunkte vor der Anpassung den verschiedenen Semgenten zugeordnet werden. In der vorliegenden Arbeit werden dazu folgende Techniken gezeigt: (1) Abgleich der Messdaten mit einer Vorlage, (2) zuweisen der Daten zu den Segmenten, (3) sortieren der Daten und (4) bestimmen der optimalen Übergangspunkte zwischen den einzelnen Segmenten. Hierbei repräsentieren die CAD-Daten die ideale Geometrie. Diese Abfolge liefert aufgeteilte und sortierte Daten für die Anpassung mit der Methode der kleinsten Quadrate.

Alle Methoden wurden mit echten Querschnittsmessdaten, die mit einem automatisierten Messsystem mit fünf Lichtschnittmessköpfen aufgenommen wurden, getestet und verifiziert.

Table of Contents

List of Figures

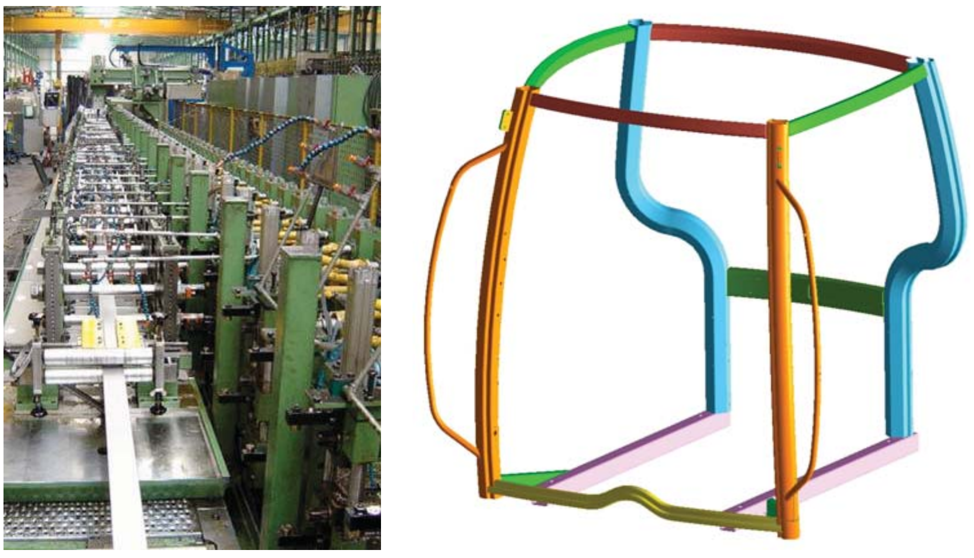
List of Tables

Chapter 1

Introduction

1.1 Motivation

Cold rolled profiled steel tubing are produced in a wide variety of more than 1000 types, which can exhibit closed or open cross-section. Moreover, the profiles can show holes along the running length. For quality control and for calibration of the profile-mill (See Figure 1.1a) the cross-section of the profile has to be measured. The steel tubing used in this thesis are mainly used for the space frame of a tractor (See Figure 1.1b).



(a) (b)

Figure 1.1: The cold roll profile mill to produce the steel tubing (a) and the application of the profiled steel tubing in a space frame of a tractor (b).

Presently the dimensions of the profile are measured manually during calibration of the profile-rolling mill. There are no further measurements during production. Adjusting the profile-mill is an elaborate process. The worker has to stop the mill at every adjusting step and measure some geometry parameters. This is an iterative process with several steps, which has an average duration of half a day. All profiles produced during calibration represent scrap which means that several tons of steel profile are discharged every time a new production is starting.

In order to increase the quality during processing and to decrease the calibration time, an automatic optical measurement system was developed.

The principle of operation is to use five light-sectioning heads, evenly distributed around the profile (See Figure 1.2). After some image processing steps the output is the contour of the profile as set of scattered data points.

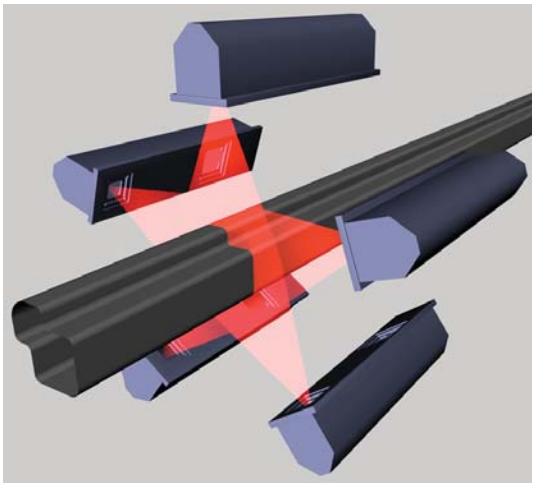
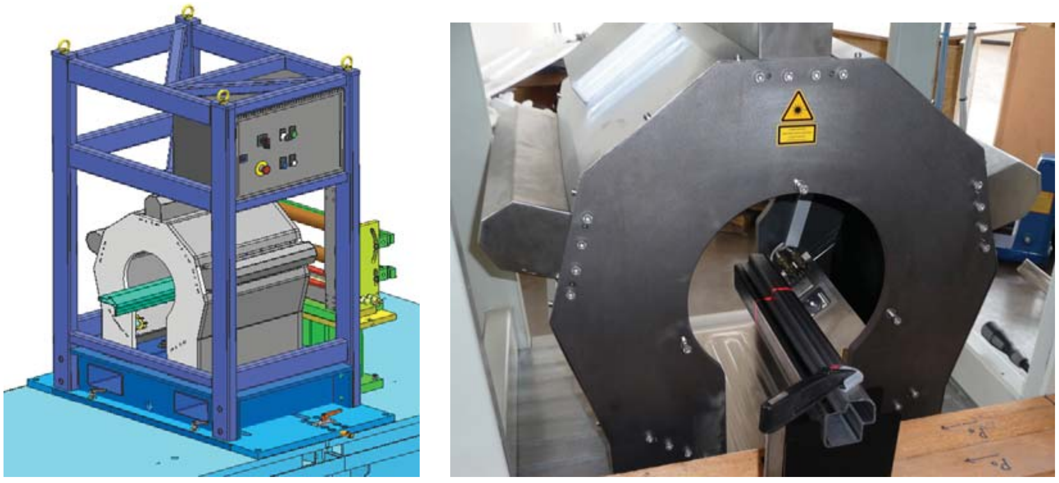


Figure 1.2: Principle of operation to measure the cross-section of a profile.

The final construction of this principle is depicted in Figure 1.3:



(a) (b)

Figure 1.3: Construction (a) and implementation (b) of the principle of operation to measure the cross-section.

In order to measure the profile the Institute for Automation at the University of Leoben developed an algorithm [10] for constrained fitting of lines and circles. For this algorithm segmented data is required. This means, the measured profile data has to be segmented into lines and circle segments before measuring.

1.2 State of the Art

From measurement an unsorted set of scattered data points is obtained on using five overlapping light-sectioning heads. Direct segmentation of this type of data means, that in one or more iteration steps a suitable segmentation in line and circle segments is found. During research no literature on this topic was found, so a new concept was developed (see Chapter 3). The idea was to perform a template matching algorithm to find the orientation of the measured data and perform an segmentation algorithm using the original CAD data.

There are several methods described in literature for matching two similar sets of data points. Most of these methods are dealing with object recognition for shapes. In [8] it is described how to use the *Orientation Indicator Index (OII)* and the *Point Based Reorientation Algorithm (PRA)* to detect the symmetry of a shape and transform the shape to a defined position. Using this algorithm with the data given from the light-sectioning measurement system, it is not possible to detect the position of the contour, because a pre-condition for a correct function of these algorithm is a solid shape.

The *Iterative Closest Point (ICP)* Algorithm [2] is also used for the matching of two contours. For this algorithm the number of points in original and template data has to be the same. As enhancement the *Trimmed Iterative Closest Point (TIPC)* [2] was introduced, where this condition does not matter. Another condition for this algorithm is, that all points have to be sorted along the contour. This is not possible for the sort of original data used in this application.

Also a patent to this topic called *Many to Many Point Matching* [11] was found. This would exactly be the algorithm, which performs the template matching. In case this is a patent, thus this algorithm can not be used for an application.

Summarizing, no algorithm to segment the data provided by the light-sectioning measurement system was found in literature. Moreover, suitable template matching and further computation algorithm are missing in literature.

1.3 Structure of the Work

The thesis is divided into seven chapters. At first there are some geometric, mathematical and imaging background in Chapter 2, used for the implementation of the algorithm in Chapter 4. Also the new concept developed in this thesis is provided in Chapter 3. At the end of the work, the test results are described in Chapter 5 and the conclusion and future work are presented in Chapter 6 and 7.

Chapter 2 gives an introduction to some geometric basics on homogeneous coordinates and vector analysis. Some mathematical basics for fitting lines and circles and some imaging background like segmentation, morphological operators and labeling are described in this chapter. All these algorithms and geometric topics are used in the new developed algorithm. A basic description of the algorithm is found in Chapter 3. Also the problem statement and the boundary conditions of the system are described. The final implementation of the algorithm can be found in Chapter 4. This chapter is divided into four main processing steps: template matching, assignment of the segmented data to model segments, sorting the data along an unique direction and finding the optimized transition point between the segments. In succession, Chapter 5 shows the result of testing the new algorithm with three different contours. In Chapter 6 the overall results are described and there are some suggestions for further development on this topic in Chapter 7.

Chapter 2

Geometrical, Mathematical and Imaging Background

2.1 Geometric Background

2.1.1 Homogeneous Coordinates

A common problem is, that points at infinity, e.g. the intersection point of two parallel lines [15, 12, 6], cannot be represented by cartesian coordinates in the Euclidean plane. For solving the problem, homogeneous coordinates in the projective plane have been introduced. Homogeneous coordinates represent a finite point in the projective plane by three coordinates. In general the number of coordinates required, is one more than the dimension of the projective space being considered. For planar representation of a line, three homogeneous coordinates are required, for representing a circle in the plane four coordinates. Another advantage of homogeneous coordinates is the easy computation of coordinate transformations and of some fitting algorithms.

A point in cartesian coordinates (x, y) is represented in homogeneous coordinates with (u,v,w), whereby

$$x = \frac{u}{w} \quad \text{and} \quad y = \frac{v}{w}. \tag{2.1}$$

So the homogeneous coordinates of a point $P = [x, y]$ in cartesian coordinates can be written as $P_h = [u, v, w]$.

Homogeneous Line Coordinates

A line in homogeneous coordinates [6] $\boldsymbol{l} = [l_1, l_2, l_3]^T$ and a point $\boldsymbol{p} = [x, y, 1]^T$ on this line is expressed as

$$l_1 x + l_2 y + l_3 = 0 \quad \text{or in matrix form} \quad \boldsymbol{p}^T \boldsymbol{l} = 0. \tag{2.2}$$

A common representation of a line is:

$$y = kx + d, \tag{2.3}$$

where k is the slope of the line, and d denotes the distance to the origin in the y axis.
On rearranging the homogeneous equation the coefficients can be compared:

$$y = -\frac{l_1}{l_2}x + \frac{l_3}{l_2}. \tag{2.4}$$

So the conversion between the representations can be defined as:

$$k = -\frac{l_1}{l_2} \quad \text{and} \quad d = \frac{l_3}{l_2}. \tag{2.5}$$

Homogeneous Circle Coordinates

A circle equation in can be written in homogeneous coordinates [6] $\boldsymbol{c} = [c_1, c_2, c_3, c_4]$ with a point $\boldsymbol{p} = [x, y, 1]^T$ on this circle:

$$c_1(x^2 + y^2) + c_2x + c_3y + c_4 = 0, \tag{2.6}$$

and in a common representation:

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0, \tag{2.7}$$

whereby $\boldsymbol{p}_c = [x_c, y_c, 1]^T$ is the center point and r is the radius of the circle.
Rearranging the equation gives:

$$x^2 - 2xx_c + x_c^2 + y - 2yy_c + y^2 - r^2 = 0, \tag{2.8}$$

and further:

$$(x^2 + y^2) - 2xx_c - 2yy_c + x_c^2 + y_c^2 - r^2 = 0. \tag{2.9}$$

Multiplying this equation with c_1 gives:

$$c_1(x^2 + y^2) - 2x_cxc_1 - 2y_cyc_1 + c_1(x_c^2 + y_c^2 - r^2) = 0. \tag{2.10}$$

In comparison of the coefficients with Equation (2.6), the coefficients can be written as:

$$c_1 = c_1, \tag{2.11}$$
$$c_2 = -2x_cc_1, \tag{2.12}$$
$$c_3 = -2y_cc_1, \tag{2.13}$$
$$c_4 = (x_c^2 + y_c^2 - r^2)c_1. \tag{2.14}$$

So the conversion between the representations of the circle can be defined as:

$$x_c = -\frac{c_2}{2c_1}, \quad y_c = -\frac{c_3}{2c_1}, \quad \text{and} \quad r^2 = \frac{c_4}{c_1} - x_c^2 - y_c^2, \tag{2.15}$$

Transformations of Coordinates

Using this homogeneous coordinates, a point can easily be rotated, shifted or scaled by multiplying the point with the corresponding transformation matrix [12]. The different matrices can be written as rotation matrix R, translation matrix T and the scaling matrix S:

$$\mathsf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathsf{T} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{and} \quad \mathsf{S} = \begin{bmatrix} s & 0 & 1 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.16}$$

whereby α is the rotation angle, Δx and Δy are the distances to shift in x and y axis and s is the scaling factor.

On multiplying a point $\boldsymbol{p} = [x, y, 1]^T$ with one of the matrices before, the result is a rotated, translated or scaled point $\boldsymbol{p}_R, \boldsymbol{p}_T$ or $\boldsymbol{p}_S$:

$$\boldsymbol{p}_R = \mathsf{R}\boldsymbol{p}, \quad \boldsymbol{p}_T = \mathsf{T}\boldsymbol{p}, \quad \boldsymbol{p}_S = \mathsf{S}\boldsymbol{p}. \tag{2.17}$$

This transformations can also be combined by multiplying the matrices. On combining the operations, the sequence has to be observed. Performing a rotation before a translation or reciprocally will not give the same result:

$$\mathsf{RT} = \begin{bmatrix} \cos\alpha & -\sin\alpha & \Delta x\cos\alpha - \Delta y\sin\alpha \\ \sin\alpha & \cos\alpha & \Delta x\sin\alpha + \Delta y\cos\alpha \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathsf{TR} = \begin{bmatrix} \cos\alpha & -\sin\alpha & \Delta x \\ \sin\alpha & \cos\alpha & \Delta y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.18}$$

2.1.2 Intersection of Two Lines

The presented method for the computation of the intersection of two lines is based on the principle of the duality [6]. A line can be represented as a point in line space, and vice versa. The intersection of two lines g and h can be expressed as connection of the two points in point space, which represent the two lines. This connecting line can be written as:

$$l : l_1x + l_2y + l_3w = 0 \tag{2.19}$$

Given are two lines in homogeneous coordinates [6]: g and h will intersect at a point $P = [x, y, w]$ in homogeneous coordinates.

$$g : g_1x + g_2y + g_3w = 0, \tag{2.20}$$
$$h : h_1x + h_2y + h_3w = 0. \tag{2.21}$$

The equations can also be written in matrix form:

$$\begin{bmatrix} l_1 & l_2 & l_3 \\ g_1 & g_2 & g_3 \\ h_1 & h_2 & h_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2.22}$$

Obviously the trivial solution $x = y = w = 0$ will solve the equation. In order to find the non-trivial solution, the determinant of the coefficient matrix has to be computed:

$$\det \begin{pmatrix} l_1 & l_2 & l_3 \\ g_1 & g_2 & g_3 \\ h_1 & h_2 & h_3 \end{pmatrix} = 0, \tag{2.23}$$

and further:

$$(g_2h_3 - h_2g_3)l_1 - (g_1h_3 - h_1g_3)l_2 + (g_1h_2 - h_1g_2)l_3 = 0. \tag{2.24}$$

On the comparison of the coefficients of Equation (2.24) with line l in Equation (2.19), the solution of the intersection point can be found as:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} g_2h_3 - h_2g_3 \\ -g_1h_3 - h_1g_3 \\ g_1h_2 - h_1g_2 \end{bmatrix}. \tag{2.25}$$

In order to find the cartesian coordinates x_c and y_c, x and y are divided by w:

$$x_c = \frac{x}{w} = \frac{g_2h_3 - h_2g_3}{g_1h_2 - h_1g_2}, \tag{2.26}$$

$$y_c = \frac{y}{w} = \frac{-g_1h_3 - h_1g_3}{g_1h_2 - h_1g_2}. \tag{2.27}$$

2.1.3 Projection of a Point onto a Line

In order to project a point $P = [u, v, 1]^T$ onto a line $l : l_1x+l_2y+l_3 = 0$, the characteristics of homogeneous coordinates are used. At first a orthogonal line l_{ortho} to l is computed. Thereby the first and the second line parameter l_1 and l_2 get switched and one of them gets negated:

$$l_{ortho} : -l_2x + l_1y + l_4 = 0, \tag{2.28}$$

whereby the third line parameter of the orthogonal line l_4 through the point P has to be computed separately:

$$l_4 = \begin{bmatrix} -l_2 & l_1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{2.29}$$

To get the projected point, the intersection point between the two lines has to be computed, using the algorithm described in Section 2.1.2.

2.1.4 Vector Projection

In order to perform the data sorting for lines, the length of vector $\boldsymbol{c}$ in Figure 2.1 is required.

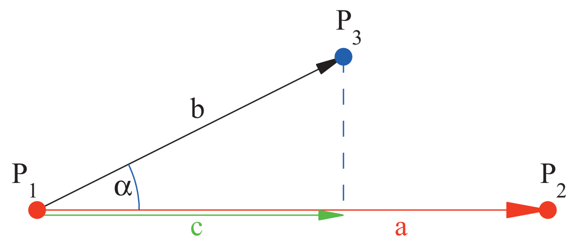


Figure 2.1: Vector $\boldsymbol{b}$ is projected orthogonal onto vector $\boldsymbol{a}$. The resulting vector $\boldsymbol{c}$ and its length has to be computed.

All Points P_1 to P_3 are known, so we can compute the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$P_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \qquad P_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \qquad P_3 = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix},$$

$$\boldsymbol{a} = P_2 - P_1 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix}, \qquad \boldsymbol{b} = P_3 - P_1 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{bmatrix}.$$

In order to compute the orthogonal projection $\mid c \mid$ of vector $\boldsymbol{b}$ to vector $\boldsymbol{a}$ the dot product [7] is used:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \mid \boldsymbol{a} \mid\mid \boldsymbol{b} \mid cos(\alpha). \tag{2.30}$$

The cos function is defined as:

$$cos(\alpha) = \frac{\mid \boldsymbol{c} \mid}{\mid \boldsymbol{b} \mid}. \tag{2.31}$$

By rearranging and combining Equation (2.30) and (2.31), the length l of the projection vector $\boldsymbol{c}$ from $\boldsymbol{b}$ to $\boldsymbol{a}$ can be computed:

$$l = \mid \boldsymbol{c} \mid = \boldsymbol{b} \frac{\boldsymbol{a}}{\mid \boldsymbol{a} \mid}. \tag{2.32}$$

2.2 Mathematical Background

2.2.1 Singular Value Decomposition

The singular value decomposition (SVD) [7] is a method to describe a matrix in form of a product of three special matrices U, Σ and V, such that:

$$\mathsf{A} = \mathsf{U} \Sigma \mathsf{V}^T, \tag{2.33}$$

whereby U and V contain the left and right singular vectors, respectively. Both matrices are unitary, which means:

$$\mathsf{U}^T \mathsf{U} = \mathsf{I} \quad \text{and} \quad \mathsf{V}^T \mathsf{V} = \mathsf{I}. \tag{2.34}$$

$\mathsf{\Sigma}$ is a diagonal matrix containing the singular values.

In this thesis the SVD is used to obtain the singular values and the right singular vectors out of a matrix D of mean free scattered data points:

$$\mathsf{D} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_N & y_N \end{bmatrix}, \tag{2.35}$$

where N denotes the number of data points.

In a geometric point of view, the result can be interpreted as following: the right singular vectors of a set of scattered data points show the main orientation of this set of data. These right singular vectors describe a orthogonal vector basis set. Each column vector represents a basis vector. In the matrix V the entries of the right singular vectors are:

$$\mathsf{V} = \begin{bmatrix} -v_2 & v_1 \\ v_1 & v_2 \end{bmatrix}. \tag{2.36}$$

The matrix U has the same dimension as the matrix D and contains the normalized coordinates of every point in this new vector basis set, defined by V.

$$\mathsf{U} = \begin{bmatrix} \hat{x}_1 & \hat{y}_1 \\ \vdots & \vdots \\ \hat{x}_N & \hat{y}_N \end{bmatrix} \tag{2.37}$$

The singular values in $\mathsf{\Sigma}$ describe this scaling factor. $\mathsf{\Sigma}$ is a diagonal matrix and can be written as follows:

$$\mathsf{\Sigma} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}, \tag{2.38}$$

where s_1 and s_2 describe the scaling factor in direction of the right singular vectors. The singular values are the two norm distances of the vector basis from the null space.

By multiplying the matrices U and $\mathsf{\Sigma}$:

$$\mathsf{U}_s = \mathsf{U}\mathsf{\Sigma}, \tag{2.39}$$

the matrix U_s contains the originally scaled data points.

Symmetry of Data

If the scaling factors in both directions are the same, i.e. $s_1 = s_2$, the scattered data, , is symmetric with respect to the right singular vectors.

Pseudo Inverse

A matrix A is called invertible [5, 7] if there exists a matrix A^{-1} such that

$$\mathsf{A}\mathsf{A}^{-1} = \mathsf{A}^{-1}\mathsf{A} = \mathsf{I}, \tag{2.40}$$

whereby I denotes the identity matrix and A^{-1} is called the inverse of A. The inverse is only defined if A is a square matrix and its determinate is unequal to zero:

$$\det \mathsf{A} \text{ is unequal to } 0. \tag{2.41}$$

For matrices, which do not fulfill these conditions, the inverse is not defined. So the Moore Penrose pseudo inverse [7] was introduced. The Moore Penrose pseudo inverse of a matrix B is defined by $\mathsf{B}^{+} = (\mathsf{B}^T\mathsf{B})^{-1}\mathsf{B}^T$ and can be computed using SVD:

$$\mathsf{B}^{+} = \mathsf{V}\Sigma^{+}\mathsf{U}^T. \tag{2.42}$$

Therefore, the pseudo inverse of Σ has to be computed. From the definition of Σ it is known, that Σ is a diagonal matrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}. \tag{2.43}$$

The pseudo inverse of this matrix is founded by inverting the entries of the main diagonal for non-zero entries. The inverse values of zero entries (or in a numerical issue, the values smaller than a border of δ) are set to zero by definition:

$$\sigma_{ii}^{+} = \begin{cases} \frac{1}{\sigma_{ii}} & \text{for } \ \sigma_{ii} \ \text{ is unequal to } \ 0 \\ 0 & \text{for } \ \sigma_{ii} < \delta. \end{cases} \tag{2.44}$$

2.2.2 Fitting a Line

Starting point of the computation is a line in homogeneous coordinates, which should be fitted [14, 4]. The line equation is given by:

$$\boldsymbol{p}^T\boldsymbol{l} = xl_1 + yl_2 + l_3 = 0, \tag{2.45}$$

where $\boldsymbol{p} = [x, y, 1]^T$ denotes the normalized homogeneous vector of a point on the line and $\boldsymbol{l} = [l_1, l_2, l_3]^T$ is the coefficient vector of the line parameters.

In case of a given set of scattered points, a point will not exactly lie on the line to fit. So a residual parameter $r_i, i = \{1 \dots N\}$, associated with the particular data point $p_i = [x_i, y_i, 1]^T, i = \{1 \dots N\}$ is introduced:

$$x_il_1 + y_il_2 + l_3 = r_i. \tag{2.46}$$

From this it follows that a set of N linear equations has to be solved to fit a line. By introducing a design matrix D the residual vector $\boldsymbol{r}$ can be computed as follows:

$$\mathsf{D}\boldsymbol{l} = \boldsymbol{r}, \tag{2.47}$$

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}. \tag{2.48}$$

In a least square sense, the residual vector has to be minimized:

$$\min E = \min_{\boldsymbol{l}} \|\mathsf{D}\boldsymbol{l}\|, \tag{2.49}$$

with the constraint:

$$\sqrt{l_1^2 + l_2^2} = 1, \tag{2.50}$$

whereby E is defined as:

$$E = \sum r_i^2 = \boldsymbol{r}^T\boldsymbol{r} = \boldsymbol{l}^T\mathsf{D}^T\mathsf{D}\boldsymbol{l} = \|\mathsf{D}\boldsymbol{l}\| \tag{2.51}$$

For solving this constrained minimization problem, the method of Lagrange Multipliers can be used:

$$\nabla_{\boldsymbol{l},\lambda}\{E - \lambda\mathsf{B}\} = 0, \tag{2.52}$$

whereby $\mathsf{B} = \boldsymbol{l}^T\mathsf{C}\boldsymbol{l} - \mathbf{1} = 0$. Substituting E and B leads to:

$$\nabla_{\boldsymbol{l},\lambda}\{\boldsymbol{l}^T\mathsf{D}^T\mathsf{D}\boldsymbol{l} - \lambda\left(\boldsymbol{l}^T\mathsf{C}\boldsymbol{l} - \mathbf{1}\right)\} = 0. \tag{2.53}$$

The derivations with respect to $\boldsymbol{l}$ and λ are:

$$\partial\boldsymbol{l} : \mathsf{D}^T\mathsf{D}\boldsymbol{l} - \lambda\mathsf{C}\boldsymbol{l} = \boldsymbol{l}\left(\mathsf{D}^T\mathsf{D} - \lambda\mathsf{C}\right) = 0 \text{ and} \tag{2.54}$$

$$\partial\lambda : \boldsymbol{l}^T\mathsf{C}\boldsymbol{l} - \mathbf{1} = 0, \text{respectively.} \tag{2.55}$$

$\boldsymbol{l}\left(\mathsf{D}^T\mathsf{D} - \lambda\mathsf{C}\right) = 0$ describes a generalized eigenvector problem, where C denotes:

$$\mathsf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{2.56}$$

to incorporate the constraint in Equation (2.50):

$$\boldsymbol{l}^T\mathsf{C}\boldsymbol{l} = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = l_1^2 + l_2^2 = 1 \tag{2.57}$$

In order to solve Equation (2.47), singular value decomposition (SVD) is used. It can be shown that the result obtained by SVD is equivalent to the eigenvector solution. If all data used for computation are mean-free, the constraint in Equation (2.50) is implicit incorporated [14].

At first the data points have to be mean free, where the mean values are $\bar{x} = \frac{1}{N}\sum x_i$ and $\bar{y} = \frac{1}{N}\sum y_i$. Next, the design matrix has to be split up into:

$$\hat{\mathsf{D}}_0 = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_N - \bar{x} & y_N - \bar{y} \end{bmatrix} \quad \text{and} \quad \hat{\mathsf{D}}_1 = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \tag{2.58}$$

and the line vector also has to be split up into:

$$\hat{\boldsymbol{l}} = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \quad \text{and} \quad l_3. \tag{2.59}$$

This yields the new equation:

$$\hat{\mathsf{D}}_0 \hat{\boldsymbol{l}} + \hat{\mathsf{D}}_1 l_3 = \boldsymbol{r}, \tag{2.60}$$

and in matrix description:

$$\begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_N - \bar{x} & y_N - \bar{y} \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} + \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} l_3 \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}. \tag{2.61}$$

In order to compute the line parameters l_1 and l_2, singular value decomposition is used. The result of the decomposition process of $\hat{\mathsf{D}}_0$ are the matrices U, Σ and V, where the right singular vector in V represented by the corresponding smallest singular value in Σ is the solution of $\hat{\boldsymbol{l}}$.

In order to compute the third line parameter l_3, which represents the normal distance to the origin, backsubstitution of the coordinates of the center point into the original homogeneous line equation is used:

$$l_3 = -\begin{bmatrix} \bar{x} & \bar{y} \end{bmatrix} \hat{\boldsymbol{l}}. \tag{2.62}$$

Computing the Residual

The smallest singular value s in the matrix Σ from the singular value decomposition is representing the residual. In common the standard deviation is defined by:

$$\sigma_e = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{r}_i^2} = \frac{\|\boldsymbol{r}_i\|_2^2}{\sqrt{N}}, \tag{2.63}$$

whereby the smallest singular value s represents the two norm distance of the residual vector $\boldsymbol{r}$:

$$s = \|\boldsymbol{r}_i\|_2^2, \tag{2.64}$$

On normalization by the number of points N, the residual is representing the standard deviation of the error e:

$$\sigma_e = \frac{s}{\sqrt{N}}. \tag{2.65}$$

2.2.3 Fitting a Circle

Starting point is the circle equation [14]:

$$(x - x_c)^2 + (y - y_c)^2 = r^2, \tag{2.66}$$

where x_c and y_c are the coordinates of the circle center point, x and y are the coordinates of a point on this specified circle and r represents the radius of the circle.

This equation can be written in parameter form:

$$c_1(x^2 + y^2) + c_2 x + c_3 y + c_4 = 0. \tag{2.67}$$

The transformation between these two equation forms can be written as follows:

$$x_c = -\frac{c_2}{2c_1}, \quad y_c = -\frac{c_3}{2c_1}, \quad r = \sqrt{x_c^2 + y_c^2 - \frac{c_4}{c_1}}. \tag{2.68}$$

There is a set of scattered points $p_i = [x_i, y_i, 1]^T, i = \{1 \dots N\}$, to which a circle has to be fitted by the algorithm. For the fitting process also the error vector $\boldsymbol{e} = [e_1 \dots e_N]^T$ has to be introduced:

$$c_1(x_i^2 + y_i^2) + c_2 x_i + c_3 y_i + c_4 = e_i. \tag{2.69}$$

This set of equations can also be written in matrix form:

$$\begin{bmatrix} x_1^2 + x_1^2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N^2 + y_N^2 & x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix}. \tag{2.70}$$

For processing the fit the design matrix D has to be split up. The new equation can be formulated as:

$$\hat{\mathsf{D}}_0 \hat{\boldsymbol{c}} + \hat{\mathsf{D}}_1 c_4 = \boldsymbol{e}, \tag{2.71}$$

$$\begin{bmatrix} x_1^2 + x_1^2 & x_1 & y_1 \\ \vdots & \vdots & \vdots \\ x_N^2 + y_N^2 & x_N & y_N \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} c_4 = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix}. \tag{2.72}$$

Before the first processing step, the scattered data points have to be mean free:

$$\hat{\mathsf{D}}_0 = \begin{bmatrix} x_1^2 + x_1^2 - \overline{(x^2+y^2)} & x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots & \vdots \\ x_N^2 + y_N^2 - \overline{(x^2+y^2)} & x_N - \bar{x} & y_N - \bar{y} \end{bmatrix}, \tag{2.73}$$

where $\overline{(x^2+y^2)} = \frac{1}{N}\sum(x^2+y^2)$, $\bar{x} = \frac{1}{N}\sum x_i$ and $\bar{y} = \frac{1}{N}\sum y_i$.

The problem to solve is now reduced to decompose the matrix $\hat{\mathsf{D}}_0$ to get the parameters c_1 to c_3:

$$\min_{\hat{\boldsymbol{c}}} \|\mathsf{D}\hat{\boldsymbol{c}}\| \quad \text{with the constraint} \quad \|\hat{\boldsymbol{c}}\| = 1. \tag{2.74}$$

This is done by using the singular value decomposition again. The right singular vector in V represented by the corresponding smallest singular value in S is the solution of $\hat{\boldsymbol{c}}$.

The fourth parameter is found by backsubstitution:

$$\hat{\mathsf{D}}_0\hat{\boldsymbol{c}} + \hat{\mathsf{D}}_1 c_4 = 0. \tag{2.75}$$

Rearranging the equation gives the circle parameter c_4:

$$c_4 = -\hat{\mathsf{D}}_0^+ \hat{\mathsf{D}}_1 \hat{\boldsymbol{c}}. \tag{2.76}$$

Computing the Residual

The computation of the circle residual is similar to compute the line residual. See Section 2.2.2 for the computation of the residual.

2.2.4 Fitting Circle with Constant Radius

For fitting a circle with constant radius [14], an iterative process has to be used. In this case the Gauss-Newton [1] algorithm is implemented. In order to find a estimation of the circle the error $\boldsymbol{e}$ of the circle points $p_i = [x_i, y_i, 1]^T$ must be minimized:

$$e_i = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2} - r \tag{2.77}$$

For the least square minimization the parameter vector $\boldsymbol{s} = [x_c, y_c]^T$ and the error vector $\boldsymbol{e}$ in dependence on this parameter vector $\boldsymbol{e}(\boldsymbol{s}) = [\boldsymbol{e}_1(\boldsymbol{s}) \ldots \boldsymbol{e}_N(\boldsymbol{s})]^T$, where N is the number of data points to fit the circle, are defined.

At first an initial starting vector $\boldsymbol{s}^0 = [x_c, y_c]^T$ of the iteration has to be defined. This starting vector can either be defined by a-priori knowledge (e.g. CAD data) or by an other fitting algorithm. The position of the starting point should lie next to the center point of the circle to fit. In a least square sense, the minimization problem is to minimize the error vector in dependence of the center point:

$$\min_{\boldsymbol{s}} E(\boldsymbol{s}) = |\boldsymbol{e}(\boldsymbol{s})|_2^2 = \sum_{i=1}^{N} e_i^2(\boldsymbol{s}) \tag{2.78}$$

The parameter vector is updated iteratively by:

$$\boldsymbol{s}^{k+1} = \boldsymbol{s}^k + \boldsymbol{\varepsilon}^k, \tag{2.79}$$

whereby k is the number of iteration and $\boldsymbol{\varepsilon}^k$ is computed by solving the normal equation:

$$\mathsf{J}_e(\boldsymbol{s}^k)^T \mathsf{J}_e(\boldsymbol{s}^k)\boldsymbol{\varepsilon}^k = -\mathsf{J}_e(\boldsymbol{s}^k)^T \boldsymbol{e}(\boldsymbol{s}^k). \tag{2.80}$$

Rearranging the equation gives:

$$\boldsymbol{\varepsilon}^k = \left(\mathsf{J}_e(\boldsymbol{s}^k)^T \mathsf{J}_e(\boldsymbol{s}^k)\right)^{-1} \mathsf{J}_e(\boldsymbol{s}^k)^T \boldsymbol{e}(\boldsymbol{s}^k), \tag{2.81}$$

whereby $\left(\mathsf{J}_e(\boldsymbol{s}^k)^T \mathsf{J}_e(\boldsymbol{s}^k)\right)^{-1} \mathsf{J}_e(\boldsymbol{s}^k)^T$ equals $\mathsf{J}_e(\boldsymbol{s}^k)^+$, the pseudo inverse of $\mathsf{J}_e(\boldsymbol{s}^k)$.

J_e is the Jacobian Matrix of the error vector $\boldsymbol{e}$ in dependence on $\boldsymbol{s}$:

$$J_e = \begin{bmatrix} \frac{\partial e_1}{\partial x_c} & \frac{\partial e_1}{\partial y_c} \\ \vdots & \vdots \\ \frac{\partial e_N}{\partial x_c} & \frac{\partial e_N}{\partial y_c} \end{bmatrix} = \begin{bmatrix} \frac{x_c - x_1}{\sqrt{(x_c-x_1)^2+(y_c-y_1)^2}} & \frac{y_c - y_1}{\sqrt{(x_c-x_1)^2+(y_c-y_1)^2}} \\ \vdots & \vdots \\ \frac{x_c - x_N}{\sqrt{(x_c-x_N)^2+(y_c-y_N)^2}} & \frac{y_c - y_N}{\sqrt{(x_c-x_1)^2+(y_c-y_1)^2}} \end{bmatrix} \tag{2.82}$$

The iteration is stopped, when $\boldsymbol{\varepsilon}^k$ reaches a minimum absolute error or a maximum number of iterations is reached. The solution of the iteration process will be $\boldsymbol{s}^{k+1}$, the center point of the fitted circle.

2.3 Imaging Background

2.3.1 Morphological Operators

Image morphology is used for image preprocessing, enhancing the object structure and for segmenting objects from the background [3, 9, 15]. It can be processed for binary and grayscale images. Following, grayscale image processing is described.

A structure element defines all neighbor points of to an image point $P = [x, y]^T$ relevant for the morphological operators. The representative point B expresses a local origin in the structure element. During operation, a structure element is moved systematically over the whole image. On moving, the grayscale value of the representative point B is replaced by the grayscale value of the neighbor points in different manner using the erosion and dilation operation. The result is assigned to a new image.

Examples of some typical structuring elements:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & B & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & B & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ B & 1 \end{pmatrix} \quad \begin{pmatrix} 1 \\ B \\ 1 \end{pmatrix}$$

The size and design of the structure element is essential for the result of the output.

Erosion and Dilation

On performing the erosion operation, the grayscale value of the representative point B is replaced by the minimum grayscale value of the neighbor image points and is assigned to the new image. The result of this operation is a reduced object size, defined by the size of the structure element.

Dilation is the reciprocal operation to erosion. The grayscale value of the representative point B is replaced by the maximum grayscale value of the neighbor image points and is assigned to the new image. The result of this operation is an enlarged object, also defined by the size of the structure element.

Both operations are not reversible. This means, the result will not be the original image on performing the operations in succession.

Opening and Closing

Opening means to perform the erosion and the dilation operation in succession using the same structure element for both operations. This operation opens small areas of the contour. It is often used to eliminate undesirable foreground pixels. Discontinuous parts of a contour are increased by this operation.

Closing means to perform first the dilation and in succession the erosion operation. This operation closes regions which are smaller than the structure element, also small background areas are eliminated. Contours of objects are smoothed by this operation.

2.3.2 Segmentation and Labeling

In literature [15, 3] segmentation for image processing is mostly used in connection with object recognition in real images. Techniques for performing this type of segmentation are thresholding, boarder tracing or edge based methods. Labeling means to extract this different objects from the image.

In this thesis a set of data is measured. This data set contains the cross-section of a cold rolled profile. Every profile consists of a concatenation of many line and circle segments. Segmentation is implemented in this thesis in the sense of finding the boarders between these circle and line segments, included in the measured data set. Therefore the new algorithm in Chapter 4 was developed.

2.3.3 Template Matching

Template matching [15, 3] is an approach to locate known objects in an image. The known object is called template. This template is moved systematically over the whole image, whereby a similarity value is computed. Based on this, a template can be found in a bigger image.

In literature for digital image processing [15, 3] only template matching for real images and image regions is described. In this thesis, the method of matching a template set of data points to measured set of data points. Therein the translational and the rotational part of the Euclidean transformation, that matches the template data to the measured data, has to be found. The translational part is found by matching the centers of gravity of the data sets. The rotational part is found by matching the main orientation of the data sets. These main orientations are found by computing the right singular vectors of the design matrices of the data sets (compare Section 2.2.1).

Chapter 3

New Concept

The aim of this thesis was to develop a new algorithm for automatic segmentation of scattered data into line and circle segments. As mentioned in Section 1.2 there were no algorithm for similar applications found in literature. This new algorithm is based on the combination of basic elements of geometry, mathematic and imaging.

3.1 Problem Statement

The starting point of the work was the CAD data of the profiled steel tubing being investigated. For each profile, the CAD data is processed to obtain model segments consisting of the following entries (see Figure 3.1c):

- Number of segments;
- Starting point of the segments;
- End point of the segments;
- Center point of circle segments;
- Direction of circle segments;
- Angle of circle segments;
- Length of line segments.

Based on this set of model segments, an ideal set of model data points is generated (see Figure 3.1d).

The next step is to get measured data points from the light-sectioning measurement system. This data is an 2D-array consisting of x and y coordinates of the contour points. Now the measured data points have to be assigned to the different segments in the same way as the model segments shows. The position and the angle of the measured contour is

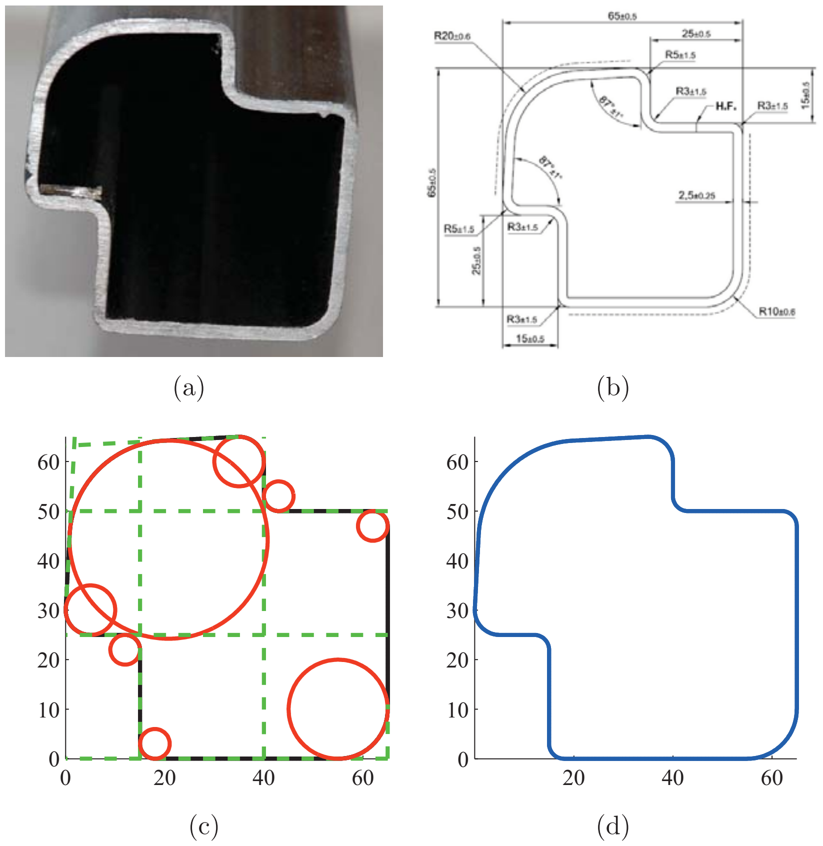


Figure 3.1: The profile, which should be measured (a), as a construction sheet (b), as model segments (c) and as model point data (d). Subfigure (a) and (b) are taken from [10] in agreement with the Institute of Automation at the University of Leoben.

not defined and the points are not fully sorted, which causes difficulties in assigning the points to the corresponding model segments.

3.2 Boundary Conditions and Requirements

In order to get good results in optical imaging applications, the most important thing is to take appropriate images from the camera. In this case the light-sectioning measurement system was optimized to acquire images with as good quality as possible, but there can be some influences from the environment. In fact the scattered plot includes some outliers and measurement noise. There will also be areas of overlapping in the data, because of the alignment of the light-sectioning sensing heads. Depending on the geometry of the profile, there will be more or less data points on different sections of the measured data. So the distribution of the points is not constant along the cross-section of the profiles. All this measurement errors and features of the measurement system should not take an influence on the result of the computation.

Another requirement is a fast computation time. During operation of the application every five seconds an images are available and should be processed in real time.

As a result of the production and measurement process, the measured data points are deviating from the model data points. In some cases, especially during calibration the profile-mill, there can be large differences between the model and measured data set. The algorithm should be resistant against such differences in a defined range.

3.3 Principle Processing Steps

The following processing steps were implemented in order to segment the scattered data points to line and circle segments.

1. **Template matching:** The orientation of the scattered data points is computed. On template matching the model data points are matched to the measured data points. So the position and the angle of the measured profile can be estimated;
2. **Data segment assignment:** This step is required to assign the measured data points to the appropriate geometric model segment. The model segments are derived from the CAD data;
3. **Data sorting:** For further processing it is necessary to sort the data along a unique direction. This is individually done for every segment;
4. **Finding the optimal transition point:** As a result of Step 2, a first estimation of the location of the transition points between the geometric segments is obtained. Now, an iterative optimization process is executed in order to find the optimal location of the transition points, depending on the actual measured data points.

Chapter 4

Implementation

This chapter deals with the implementation of different methods for segmentation of geometry data in *Matlab*® .
The idea is to overlay the measured data points and the model data points derived from CAD data which are depicted in Figure 4.1. The intersection points between the different segments are known from the model segments, which are also derived from CAD data. Based on this information the measured data points can be segmented.

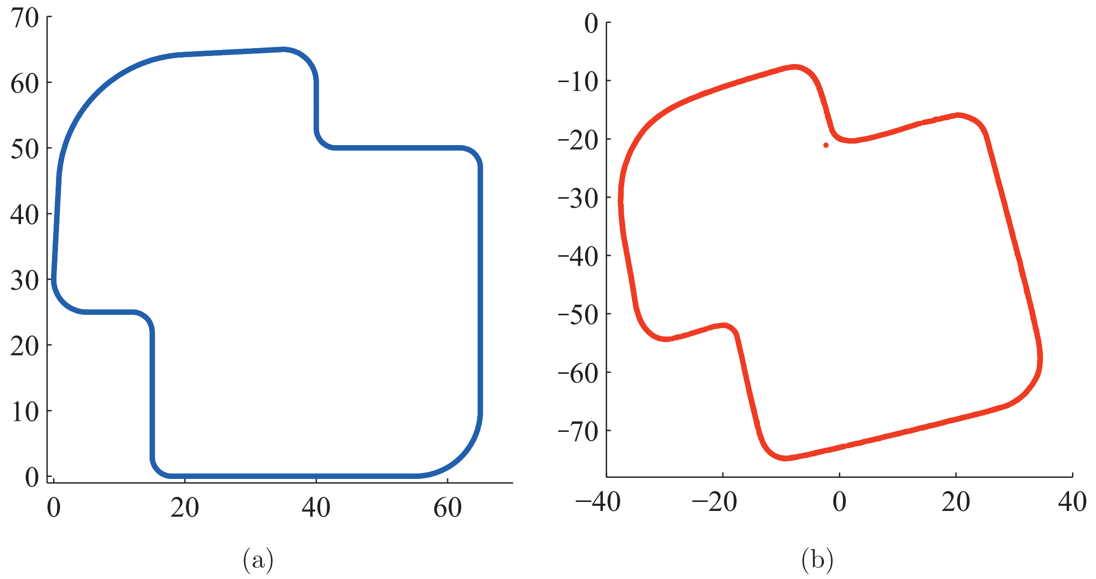


Figure 4.1: Model data points derived from CAD data (a) and measured data points (b): Main differences are the shift in x and y direction and the angle α. There are also some outliers and different radii as well as different dimensions.

4.1 Template Matching

The output of the template matching algorithm is the Δx and Δy shift and the angle $\Delta\alpha$ between the two contours. Before any computations can start, some preprocessing steps have to be performed (see Section 4.1.1), next the matching algorithm can be processed.

After matching the measured data points has to be rotated and shifted to the model data points, using the transformation matrix T:

$$\mathsf{T} = \begin{bmatrix} ccc\cos\Delta\alpha & -\sin\Delta\alpha & \Delta x \\ \sin\Delta\alpha & \cos\Delta\alpha & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The transformation is realized by the multiplication of a homogeneous point coordinate vector $\boldsymbol{p} = [\Delta x, \Delta y, 1]^T$ with the transformation matrix T. The result is gives the rotated and shifted point $\boldsymbol{p}_T = [x_T, y_T, 1]^T$:

$$\boldsymbol{p}_T = \mathsf{T}\boldsymbol{p}. \tag{4.2}$$

The multiplication of all points of a contour with the transformation matrix shifts and rotates the whole contour according to the output of the template matching algorithm.

4.1.1 Data Preprocessing

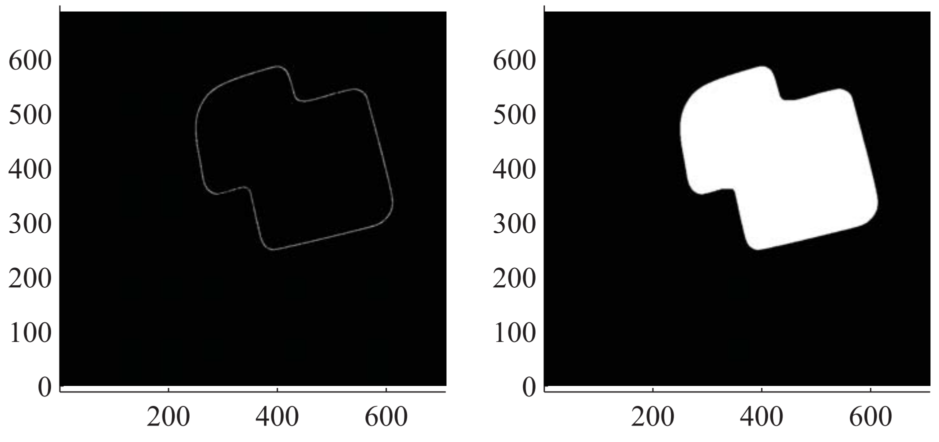
The measured data points are taken from a system of five light-sectioning sensing heads [13]. Depending on the geometry of the profile, there will be more or less data points on different sections of the measured data. There will also be areas of overlapping in the data. So the distribution of the points is not constant for the whole profile.

In order to distribute the points on the contour regularly, the following method has been developed. This is only for computing the Eigenvector. For measurement, the original measurement data set is used.

Step 1: At the beginning the measured points have to be transformed to an image. Therefore all points have to be moved to positive x and y coordinates. For a better resolution, the points get scaled by multiplying with five. Now all points get round to the nearest integer and are assigned to the appropriate pixel in the image.

Step 2: Performing the morphological dilation operation. The structure element used for the operation was a 10×10 matrix filled with ones. After dilation the *Matlab*® function *fill holes* is used. This is for filling the contour to get a solid object. At the end of the step an erosion operation is performed to get the initial size back. Therefore the same structure element as for dilation is used. This operations are shown in Figure 4.2. The morphological operators are described in Section 2.3.1.

Step 3: If there are outliers or other objects in the image, the operation in step two will produce more than one solid object. For further processing only the largest object is used.



(a) (b)

Figure 4.2: The contour transformed to an image (a) to get a solid object (b).

Therefore the objects get sorted by the area using labeling.

Step 4: Compute the contour of the object. The *Matlab*® function *contour* gives the points on the contour of the object in an regularly distribution. This contour is used for computing the Eigenvector in Section 4.1.2 instead of the measured data.

Step 5: Scale the contour by dividing the points by five and transform the contour to the originally used coordinates.

4.1.2 Eigenvector Method

Translation

At first the Center of Gravity (COG) of both contours has to be overlayed. Therefore the arithmetic mean of the x and y coordinates $\overline{x}$ and $\overline{y}$, of both contours, is calculated and shifted to the origin. N is the number of all points in the appropriate contour. The index M is allocated with the measured data points $x_{M,i}$ and $x_{M,i}$, the index CAD with the model data points $x_{CAD,i}$ and $y_{CAD,i}$ derived from CAD data:

$$\overline{x}_M = \frac{1}{N_M}\sum_{i=1}^{N_M} x_{M,i} \quad \text{and} \quad \overline{y}_M = \frac{1}{N_M}\sum_{i=1}^{N_M} y_{M,i}, \tag{4.3}$$

$$\overline{x}_{CAD} = \frac{1}{N_{CAD}}\sum_{i=1}^{N_{CAD}} x_{CAD,i} \quad \text{and} \quad \overline{y}_{CAD} = \frac{1}{N_{CAD}}\sum_{i=1}^{N_{CAD}} y_{CAD,i}, \tag{4.4}$$

whereby The next step is to build the translation matrices:

$$\mathsf{T}_M = \begin{bmatrix} 1 & 0 & -\overline{x}_M \\ 0 & 1 & -\overline{y}_M \\ 0 & 0 & 1 \end{bmatrix} \quad and \quad \mathsf{T}_{CAD} = \begin{bmatrix} 1 & 0 & -\overline{x}_{CAD} \\ 0 & 1 & -\overline{y}_{CAD} \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.5}$$

The translation matrices include the inverse COG coordinates $\overline{x}_M, \overline{y}_M, \overline{x}_{CAD}$ and $\overline{y}_{CAD}$ to shift the contour back to the origin. Multiply all contour points $\boldsymbol{p}_M$ and $\boldsymbol{p}_{CAD}$ with the translation matrices T_M and T_{CAD}, the translated points $\boldsymbol{p}_{M,T}$ and $\boldsymbol{p}_{CAD,T}$ get computed:

$$\boldsymbol{p}_{M,T} = \mathsf{T}_M \boldsymbol{p}_M, \quad \boldsymbol{p}_{CAD,T} = \mathsf{T}_{CAD} \boldsymbol{p}_{CAD}. \tag{4.6}$$

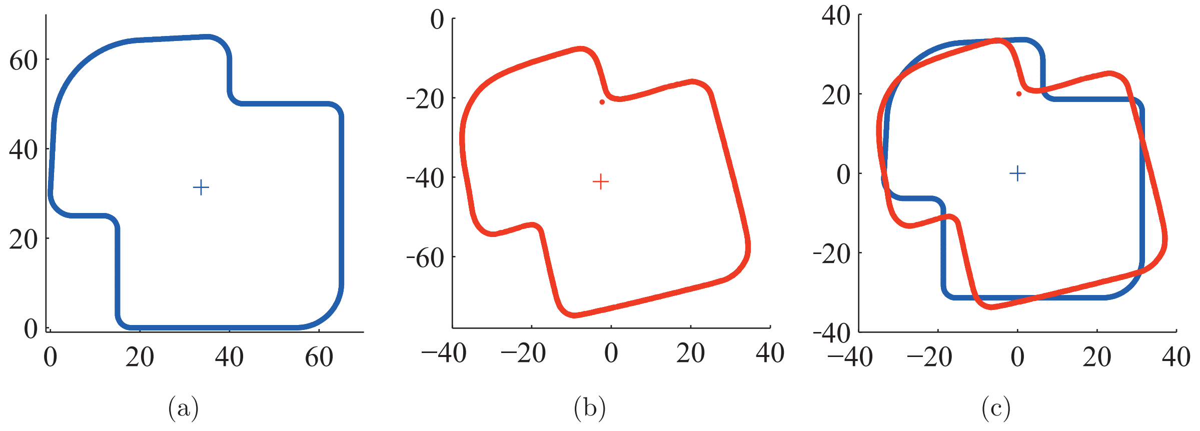


Figure 4.3: The COG is marked with an " + " in the model data points (a) and the measured data points (b). Then both contours are shifted to the origin (c).

Rotation

If the Center of Gravity is overlayed (see Figure 4.3c), the angular deviation between the two contours can be computed (see Figure 4.4). Therefore the eigenvector $\boldsymbol{e}_M$ for the measured data points and $\boldsymbol{e}_{CAD}$ for the model data points is computed. The computation of the eigenvector is described in Section 2.2.1. The angle-difference is the result of calculating the angle $\Delta\alpha$ between the eigenvectors, using the dot product [7].

$$\Delta\alpha = \arccos\left(\frac{\boldsymbol{e}_M \cdot \boldsymbol{e}_{CAD}}{\mid \boldsymbol{e}_M \mid\mid \boldsymbol{e}_{CAD} \mid}\right) \tag{4.7}$$

Due to the computation of the eigenvector based on the arc cosine function, the resulting angle $\Delta\alpha$ can switch at 180 degree. For fixing this problem, a maximum angle of $\pm$ 45 degree was defined. If the angle between the two eigenvectors is bigger than 135 degree and smaller than 275 degree, the angle is reduced to 180 degree.

This method works perfect for all non symmetric contours. For symmetric contours, the eigenvector strongly depends on the location of outliers as well as measurement noise. In these situations the boundary method is advantageous (see next section).

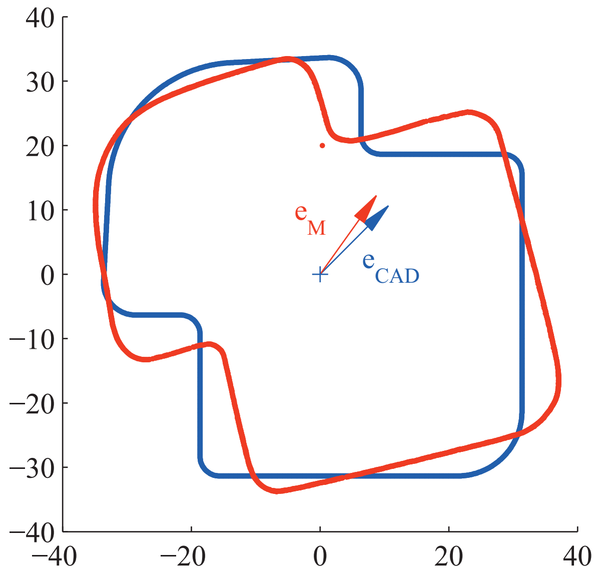


Figure 4.4: The eigenvector $\boldsymbol{e}_{CAD}$ of the model data points and the eigenvector $\boldsymbol{e}_{M}$ of the measured data points shows the angular deviation $\Delta\alpha$ between the two contours.

4.1.3 Boundary Method

For the boundary method the contour data has to be mean-free. Making the data mean-free by computing the Center of Gravity and shifting the contour to the origin is described in Section 4.1.2.

Now, the distance d between the maximum and minimum vertical coordinate of all contour points is computed for both contours. This is repeated for the contours after stepwise rotation in counter-clockwise direction from 0 to 180 degree.

The result is a diagram where the step angle ϕ is assigned to the x axis and the distance d is assigned to the y axis. This computation is done for the model data points and the measured data points separately (see Figure 4.5).

Then the maximum distance of every contour is computed and marked with a vertical line (see Figure 4.5). These maxima are at an corresponding angle. Computing the angle difference between the two maxima will give the rotational difference between the two contours.

This method is appropriate, as long as the measurement noise is low and occurring outliers are not too far away from the contour. For these situations outlier elimination techniques and noise reduction methods, e.g. using smoothing filters, are required.

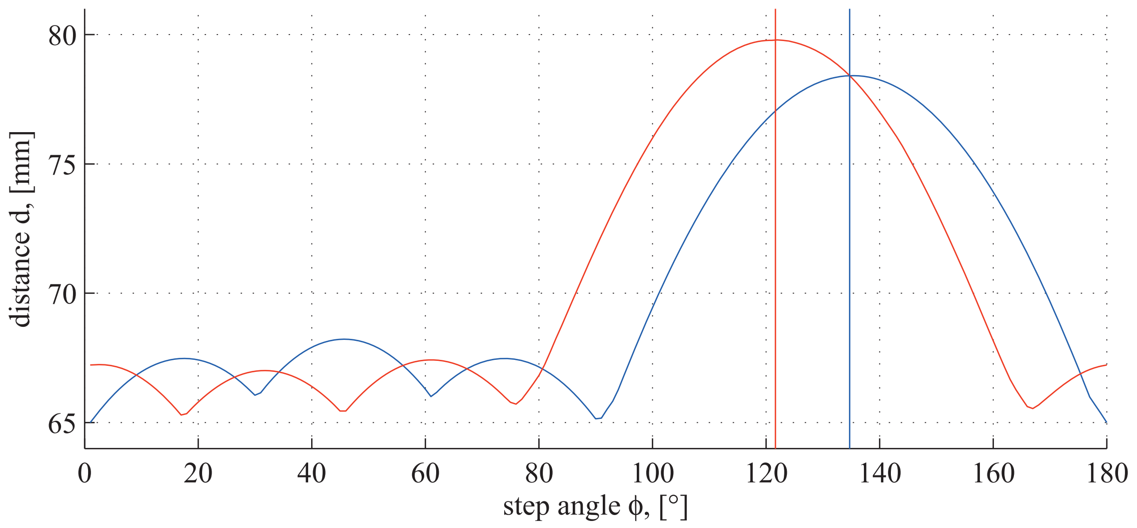


Figure 4.5: Maximum vertical spread for the contours being rotated by the step angle ϕ, computed for the model data points (blue line) and the measured data points (red line).

4.2 Data Segment Assignment

From the model segments the beginning, end and type of each data segment is known. We also know the center point of the circle segments from the model segments. In order to assign the measurement data points to the segments defined by the model segments, a bandwidth technique was introduced. There is an inner and outer boundary. Every point outside this boundary is an outlier and will not be considered for further measurement. The maximum bandwidth size is limited by the smallest radius of a circle segment.

4.2.1 Line Segments

Step 1: Introducing a bandwidth. The bandwidth describes a small band along the model data points with an inner and outer boundary. Compute the edge points of the bandwidth (I_s, I_e, O_s, O_e, see Figure 4.6). First of all the directional unit vector $\boldsymbol{d}$ from the starting point S to the end point E of the segment is required. These points are known from the matched model segments:

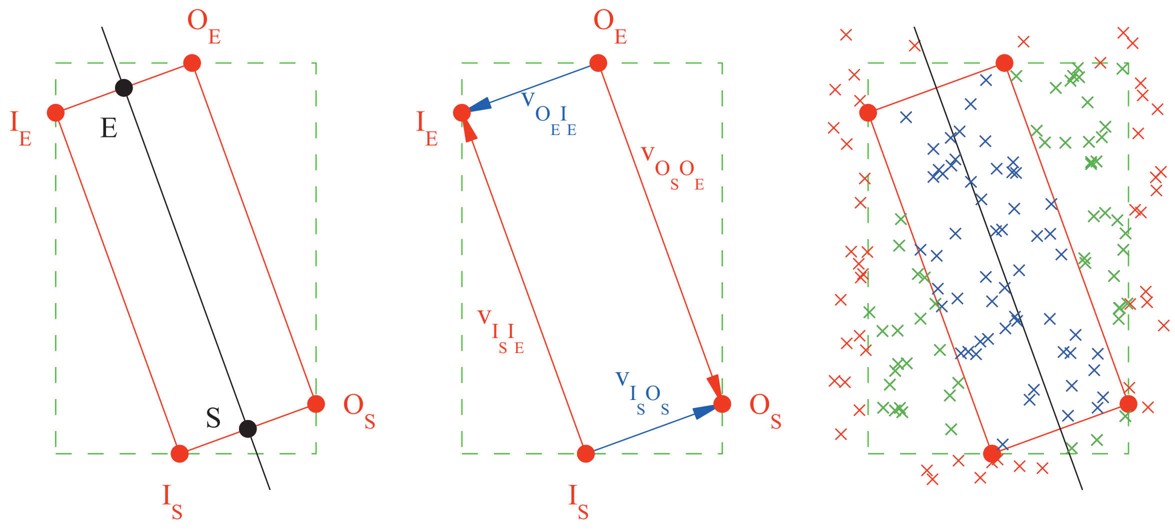
$$\boldsymbol{d} = \frac{\boldsymbol{v}_{ES}}{\mid \boldsymbol{v}_{ES} \mid} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}, \quad where \quad \boldsymbol{v}_{ES} = \boldsymbol{p}_S - \boldsymbol{p}_E. \tag{4.8}$$

Subsequently the two vectors normal to the unit vector $\boldsymbol{d}$ are computed. This is accomplished by interchanging the first and the second entry of the vector $\boldsymbol{d}$ and negating one of these alternating:

$$\boldsymbol{d}_{out} = \begin{pmatrix} -d_2 \\ d_1 \\ d_3 \end{pmatrix}, \qquad \boldsymbol{d}_{in} = \begin{pmatrix} d_2 \\ -d_1 \\ d_3 \end{pmatrix}. \tag{4.9}$$

The multiplication of the normal vectors $\boldsymbol{d}_{in}$ and $\boldsymbol{d}_{out}$ with the value of the bandwidth (bw) and adding the coordinates of the starting and ending points S and E gives the inner and outer point vectors of the bandwidth, respectively:

$$\boldsymbol{p}_{O_S} = \boldsymbol{d}_{out}bw + \boldsymbol{p}_S, \tag{4.10}$$
$$\boldsymbol{p}_{I_S} = \boldsymbol{d}_{in}bw + \boldsymbol{p}_S, \tag{4.11}$$
$$\boldsymbol{p}_{O_E} = \boldsymbol{d}_{out}bw + \boldsymbol{p}_E, \tag{4.12}$$
$$\boldsymbol{p}_{I_E} = \boldsymbol{d}_{in}bw + \boldsymbol{p}_E. \tag{4.13}$$



(a) Bounding Box (b) Vectors (c) Point Selection

Figure 4.6: Computation of line segments: create a bounding box (a), compute vectors (b) and find points inside the the rectangle, spanned by the vectors (c).

Step 2: Create a bounding rectangle parallel to the coordinate axis using the maximum and minimum points of the bandwidth box. Locate all points inside this box using the x- and y- coordinates of the points (see Figure 4.6). This assignment to the particular bounding box simplifies further computation steps.

Step 3: Computing the following vectors:

$$\boldsymbol{v}_{I_S O_S} = \boldsymbol{p}_{O_S} - \boldsymbol{p}_{I_S}, \tag{4.14}$$
$$\boldsymbol{v}_{O_E I_E} = \boldsymbol{p}_{I_E} - \boldsymbol{p}_{O_E}, \tag{4.15}$$
$$\boldsymbol{v}_{I_S I_E} = \boldsymbol{p}_{I_E} - \boldsymbol{p}_{I_S}, \tag{4.16}$$
$$\boldsymbol{v}_{O_E O_S} = \boldsymbol{p}_{O_S} - \boldsymbol{p}_{O_E}. \tag{4.17}$$

By computing the cross product of every directional point vector $\boldsymbol{v}_{I_S p_i} = \boldsymbol{p}_i - \boldsymbol{p}_{I_S}$, $i = \{1....N\}$, where N is the number of points inside the bounding box, and the vector of the bandwidth $\boldsymbol{v}_{I_S I_E}$, the sign of the z-coordinate shows whether the point lies on the either or the other side of $\boldsymbol{v}_{I_S I_E}$ (see Figure 4.7).

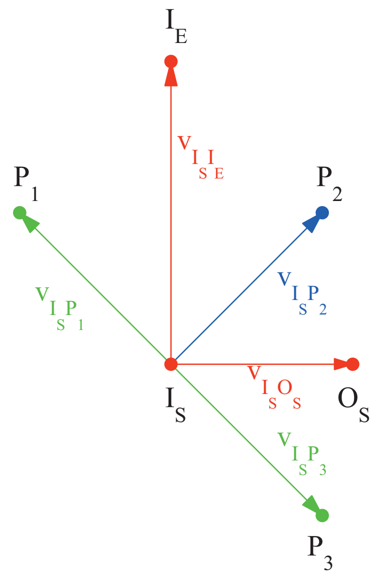


Figure 4.7: Computation of the cross product verify the location of a measured point with respect to the bandwidth borders.

If both of the following conditions are fulfilled, a point lies between the two vectors $\boldsymbol{v}_{I_S I_E}$ and $\boldsymbol{v}_{I_S O_S}$:

$$\boldsymbol{v}_{I_S I_E} \times \boldsymbol{v}_{I_S P_i} > 0, \tag{4.18}$$
$$\boldsymbol{v}_{I_S O_S} \times \boldsymbol{v}_{I_S P_i} < 0. \tag{4.19}$$

In Figure 4.7 only point P_2 will be correct for this equations.

Continuing the computation with the vectors $\boldsymbol{v}_{O_E I_E}$ and $\boldsymbol{v}_{O_E O_S}$ the sign of the z-coordinate of the cross product with every point vector $\boldsymbol{v}_{O_E P_i} = \boldsymbol{p}_i - \boldsymbol{p}_{O_E}$ has to be once positive and once negative. All points inside this bandwidth belong to the line segment.

4.2.2 Circle Segments

From the model segments the center point C of a circle, the starting point S and the end point E are known.

Step 1: In the first step the coordinate of the points of the inner and outer boundary I_S and O_S are computed using the intercept theorem:

$$\frac{r}{x_S - x_C} = \frac{r - bw}{x_{I_S} - x_C}, \tag{4.20}$$

$$\frac{r}{y_S - y_C} = \frac{r - bw}{y_{I_S} - y_C}, \tag{4.21}$$

$$\frac{r}{x_S - x_C} = \frac{r + bw}{x_{O_S} - x_C}, \tag{4.22}$$

$$\frac{r}{y_S - y_C} = \frac{r + bw}{y_{O_S} - y_C}. \tag{4.23}$$

Therein, the following variables are used for the radius (r) and the bandwidth (bw):

$$r = \mid \boldsymbol{p}_S - \boldsymbol{p}_C \mid, \tag{4.24}$$

$$bw = \mid \boldsymbol{p}_{I_S} - \boldsymbol{p}_S \mid = \mid \boldsymbol{p}_{O_S} - \boldsymbol{p}_S \mid . \tag{4.25}$$

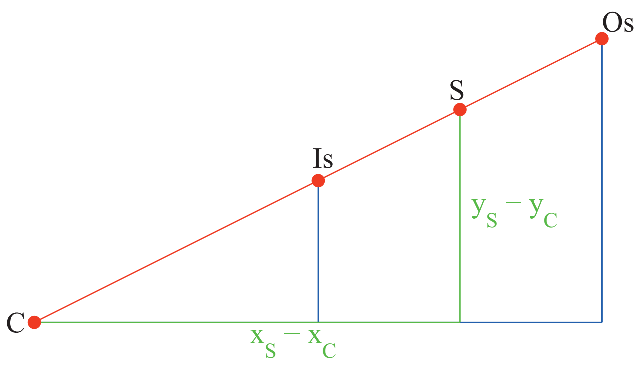


Figure 4.8: Use of the intercept theorem to compute the points of the inner and outer bandwidth

Rearranging the equations will give the coordinates of the inner point I_S and the outer point O_S:

$$x_{I_S} = \frac{r - bw}{r}(x_S - x_C) + x_C, \tag{4.26}$$

$$y_{I_S} = \frac{r - bw}{r}(y_S - y_C) + y_C, \tag{4.27}$$

$$x_{O_S} = \frac{r + bw}{r}(x_S - x_C) + x_C, \tag{4.28}$$

$$y_{O_S} = \frac{r + bw}{r}(y_S - y_C) + y_C. \tag{4.29}$$

The same procedure is repeated for the end point E of the circle segment, resulting in the point coordinates of the inner and outer bandwidth points I_E and O_E, respectively.

Step 2: Create a bounding square and take only points inside this square for further computations. Center point of the square is the center point of the circle segment in the model segment. Edge length is: $a = 2(r + bw)$.

Step 3: Compute vectors from the center point of the circle C to the start point O_S and end point O_E of the segment, and from every point inside the square to the circle center C. Furthermore the length of these vectors are computed.

Step 4: Check if the point is inside the angular segment, defined by the starting and end points S and E. Compute the cross product from every point vector to the vectors representing the beginning and end of the segment. If the point is inside, one result is positive, the other result is negative. On any other constellations the point is an outlier.

Step 5: Check if the point is within the bandwidth. The length l_i of the directional point vectors $\boldsymbol{v}_{CP_i} = \boldsymbol{p}_i - \boldsymbol{p}_c$, $i = \{1 \dots N\}$, where N is the number of points inside the angular segment defined in step 4, represents the distance of the points to the circle center. If the length meets the Condition (4.30), the point is inside the bandwidth and belongs to the segment.

$$(r - bw) < l_i < (r + bw) \tag{4.30}$$

4.3 Data Sorting

For further processing steps the measurement data points must be sorted along the perimeter of the contour. Here, the counter-clockwise direction is used.

4.3.1 Line Segments

First of all, a directional vector $\boldsymbol{d}$ is computed between the start- and end point of the segment. Furthermore, the directional vectors of the measurement point P_i, assigned to the segment, to the starting point S are computed:

$$\boldsymbol{v}_{SP_i} = \boldsymbol{p}_i - \boldsymbol{p}_S. \tag{4.31}$$

Now these directional vectors are projected onto $\boldsymbol{d}$. Section 2.1.4 is describes how to get the length of the projected vector on the directional vector $\boldsymbol{d}$. By sorting the points with increasing length, all points in this segment are arranged along the directional vector $\boldsymbol{d}$.

4.3.2 Circle segments

The points of circle segments can be easily sorted along the perimeter around the center of the circle, known from the model segments. To calculate the angle the dot product is used again:

$$\theta_i = \arccos\left(\frac{\boldsymbol{v}_{CP_i} \cdot \boldsymbol{v}_{CS}}{\mid \boldsymbol{v}_{CP_i} \mid\mid \boldsymbol{v}_{CS} \mid}\right). \tag{4.32}$$

By sorting the points on increasing the angle θ, all points in the segment are arranged along the circular arc, defined by the starting and ending points of the segment.

4.4 Find Transition Points

At this point the approximate location of the transition points are known from the model segments. The measurement data points are assigned to the appropriate segments and are sorted. If the measured data points are exactly the same as the model data points, the computation can be stopped at this point and the transition points between circles and lines are found. Normally there is a deviation between measured data points and model data points, so there have to be some more steps to find the exact transition points.

4.4.1 Residuals of Lines and Circles

The idea was to take two segments. By fitting a line or a circle into the segments, the residuum of the fit can be computed. Then add points from beginning of the second segment to the end of the first segment and compute the residuum again. Continuing this till the end of the second segment, the residuum of the line or circle of the first segment will increase. Computing this residuum in the inverse way by adding points from the end of the first segment to the beginning of the second segment, this residuum will also increase. At the end, an transition point can be computed by finding the point of the minimum residuum of both fitting processes.

This algorithm is illustrated by means of the example depicted in Figure 4.9. The structure consists of a line segment and a circle segment, whereas the exact location of the transition point is expected to be closer to the circle segment.

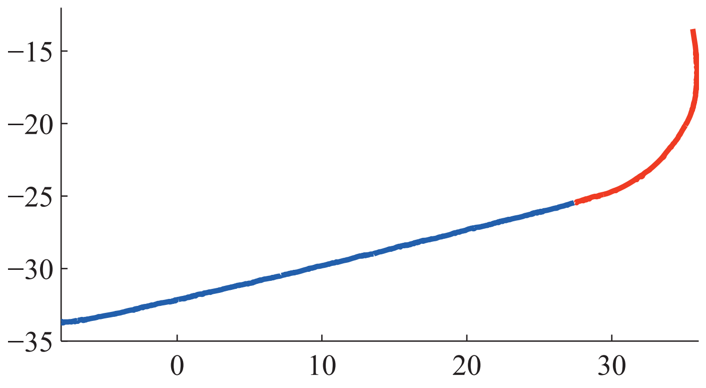


Figure 4.9: An example for a circle-line transition, which should be optimized.

Line Fitting

At first a number of data points for fitting the line has to be determined. In this case there have been used 200 points for fitting the line. These points are marked green in Figure 4.10. Taking more data points from the line reduces the influence of the circle points on the residual. So the number of points has to be constant, because otherwise the residual will be reduced on an increasing number of points.

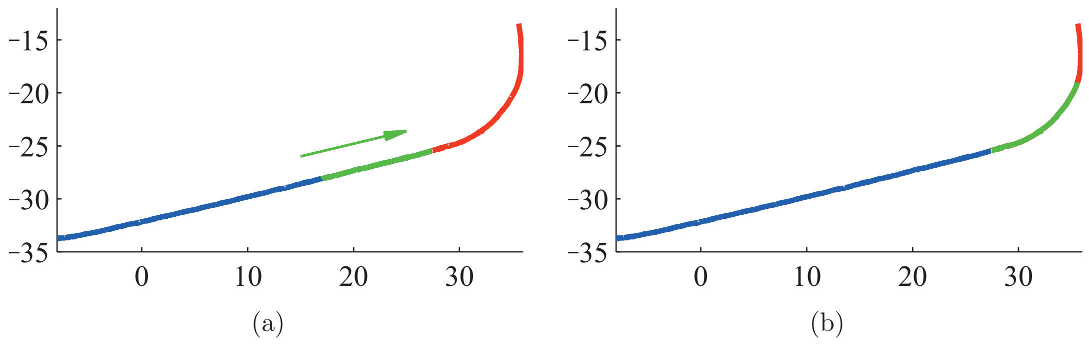


Figure 4.10: Used data points for line fitting (green) at the beginning (a) and the end (b) of the iteration process.

Next, this set of 200 points is moved stepwise along the direction of the arrow in Figure 4.10a. At every step a line fit is processed and the residual is computed. The line fit and how to compute the residual is described in Section 2.2.2.

In Figure 4.11 the residual of the iteration process is shown. In this diagram the residual is starting to rise after about 120 iteration steps. This is what was expected from Figure 4.9, where the transition point should be closer to the circle segment.

Circle Fitting

The circle fitting process is similar to the line fitting process, but for circle fitting all points of the circle segment have to be used. The quality of the circle fit is better if more

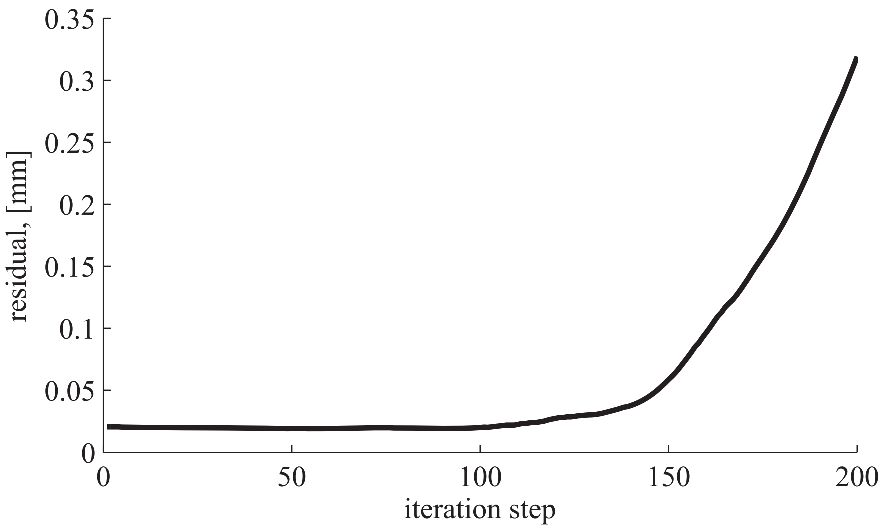


Figure 4.11: Residual of the iterative line fitting process

points of the circle segment are used for fitting. The iteration process is proceeded at the same direction as for the line fitting process. At the beginning 200 points from the end of the line are added to the beginning of the circle. Continuing the process at every iteration step one points at the beginning of the circle is deleted. At the end of the process only a few circle points are left. This is shown in Figure 4.12. For a stable process the number of points left can't get less than a predefined number of points.

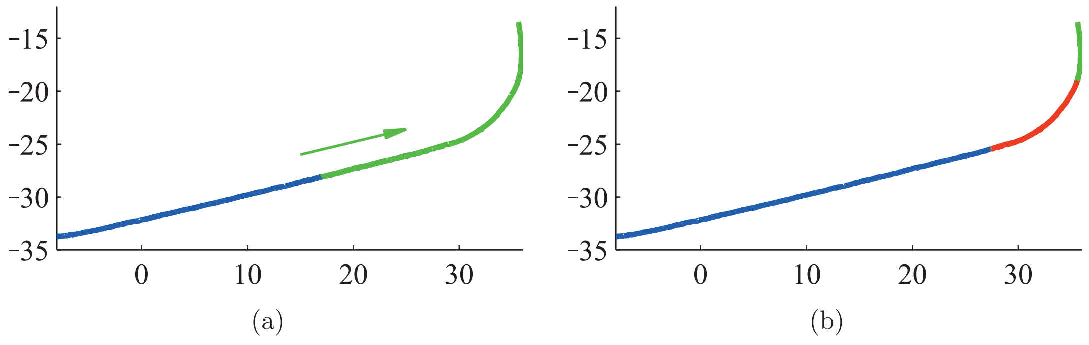


Figure 4.12: Used data points for circle fitting (green) at the beginning (a) and the end (b) of the iteration process.

The Figure 4.13 shows the residual of the circle fitting process. At every iteration step a residual has been computed. In this diagram the curve is falling to a nearly constant value of 0.04 beginning at step 120.

Finding the Transition Point from the Residual

By adding the curve of the line residual and the circle residual, the blue curve in Figure 4.14 is computed. There is a minimum of the curve at about iteration step 110, but this

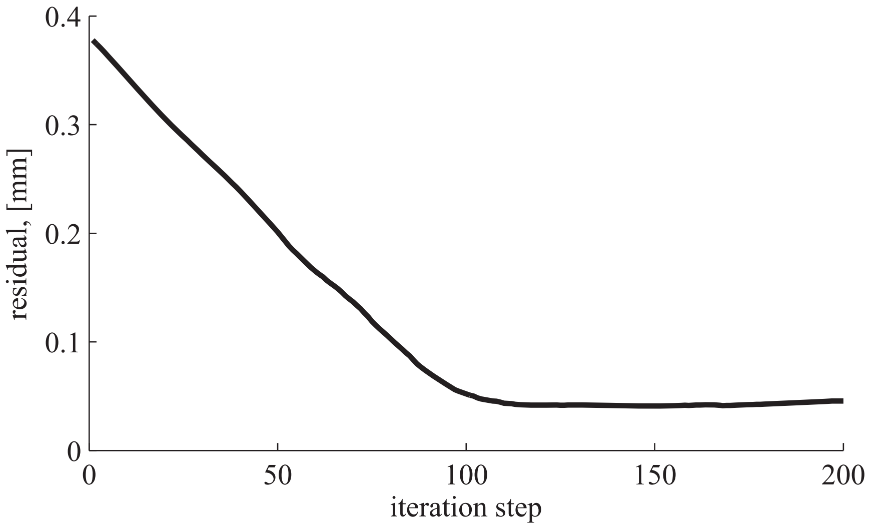


Figure 4.13: Residual of the iterative circle fitting process

minimum is not distinct.

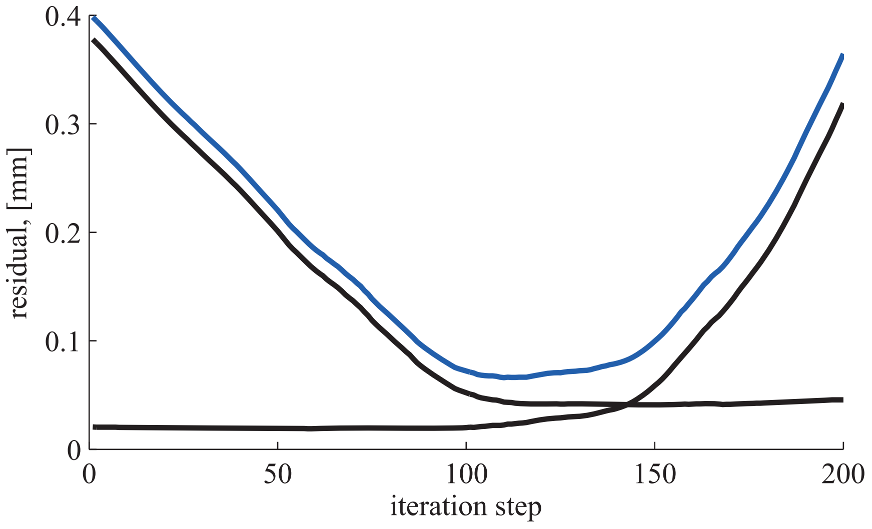


Figure 4.14: Sum of the residuals of the iterative line and circle fitting to compute the transition point.

Point 100 indicates the transition point specified by the matched model segments. Points left of point 100 belongs to the line segment, points on the right belongs to the circle segment. The value of the blue curve is nearly constant from step 100 to step 140. Because of this reason, a transition point can't reliably be computed out of this curve. As a result, an alternative approach has been implemented, which is described in the following section.

The next idea was to delete the points at the area where the residual is nearly constant. Further tests show, that this assumption will be useful for most of the transition problems. But there will be no solution for some problems, especially when the radius of the circle is very small. Then the residual of the circle fit won't change enough during the iteration process, so the entire set of points used for iteration get lost.

4.4.2 Fitting Circles

From template matching all circle segments and from the model segments, the radius and the angle of every circle segment is known. The main idea was to detect the circle segments and fit a circle with constant radius. Projecting this new found center point orthogonal on the fitted line next to the circle will give the new transition point. For a better understanding of the following algorithm, the same data set as used for the method *Residuals of Circles and Lines*, depicted in Figure 4.9, is used.

Fitting Circles and Lines

At first about ten data points at the beginning and the end of every segment have to be deleted. This is for better results in the following steps. Then a fitting algorithm as described in Section 2.2.2 for line segments and as described in Section 2.2.4 for circle segments with constant radius is performed. The radius of the circle segment is known from the model segment. The result of these fitting algorithm will be a line in homogeneous coordinates l_1 and the coordinates of the center point P_C of the fitted circle.

Projecting the Center Point to the Line and Find the Intersection Point

The center point of every circle and the homogeneous line coordinates are known from the fitting before. By projecting the center point of the circle onto the line, the best transition point for the circle-line transition is found.

At first a line orthogonal to the fitted line l_{ortho} has to be computed, and next the intersection point P_{int} of these two lines has to be computed. This is a basic operation dealing with homogeneous coordinates and is described in Section 2.1.1. Figure 4.15a is an illustration of this part of the algorithm.

Find the Optimized Transition Point

After finding the intersection point between the two lines, as computed before, the optimized transition point P_{trans} between the line and the circle segment has to be found.

At first a directional vector $\boldsymbol{d}$ from the starting point S of the line segment to the intersection point of the two lines P_{int} is computed:

$$\boldsymbol{d} = \boldsymbol{p}_{int} - \boldsymbol{p}_S. \tag{4.33}$$

Subsequently, the distance l_i of every point of the two segments to starting point of the line segment in direction of the vector $\boldsymbol{d}$ is computed. This is described in Section 2.1.4.

Also the length of the vector $\boldsymbol{d}$ l_d is computed. If the distance l_i is greater than the length l_d of the vector $\boldsymbol{d}$, the point belongs to the circle segment, otherwise the point belongs to the line segment.

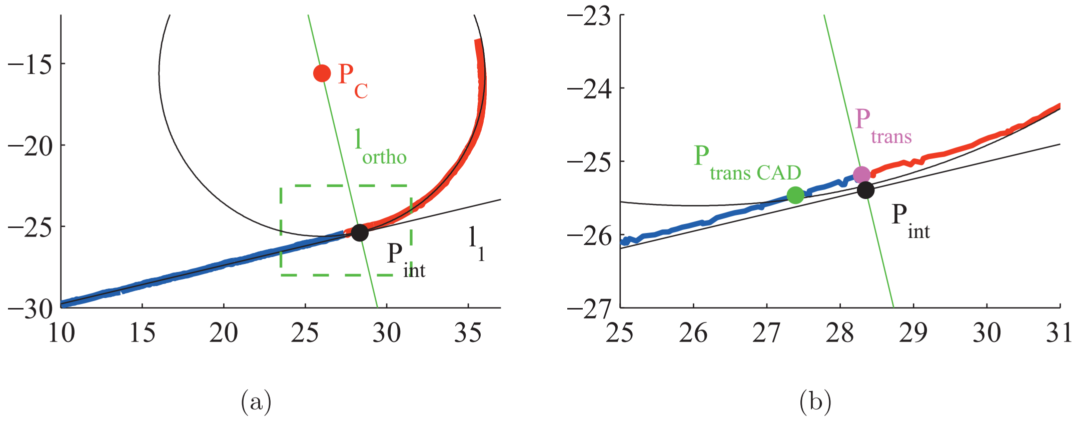


Figure 4.15: Finding the optimized transition point on projecting the circle center orthogonal on the line. The left subfigure shows the global problem, the right figure is zoomed to see the detailed points.

Figure 4.15b shows the old transition point $P_{trans\ CAD}$ from the matched model segments, the intersection point P_{int} of l_1 and l_{ortho} as well as the new transition point P_{trans} of the two segments after segmentation using the algorithm above. As supposed before, the new transition point is closer to the circle segment.

Chapter 5

Experimental Work and Results

This chapter deals with testing three different profiles, as you can see in Figure 5.1.

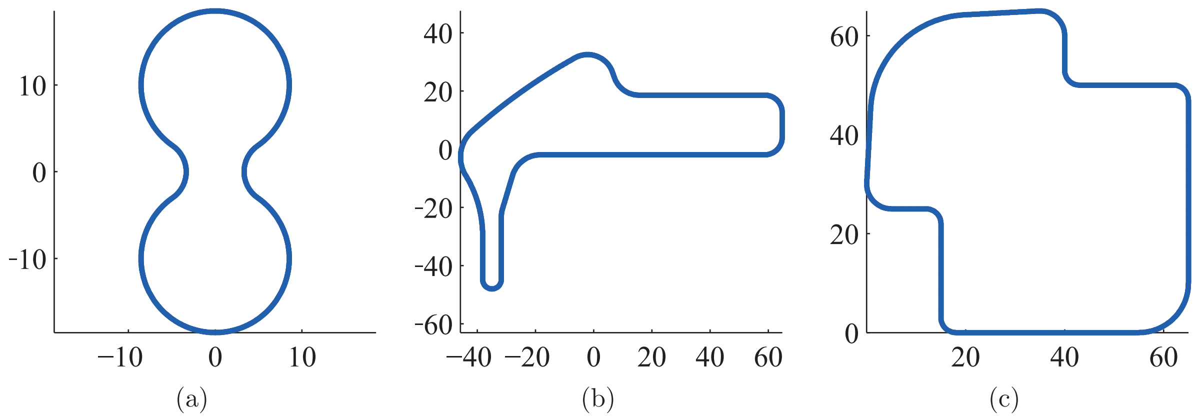


Figure 5.1: (a) shows a profile where no line segment is included. (b) and (c) show some mixed profiles with circle and line segments. The profiles are further named profile 1-3.

The pros and cons of every processing step are shown in this chapter. At first, the template matching algorithm including data preprocessing is tested with artificially generated and measured data. Also the result of template matching for symmetric data is shown. Subsequently the results of step two and three, data segment assignment and data sorting are shown. At the end, the overall results of step four, finding the transition point, are shown, where also some examples are included.

5.1 Template Matching

Before starting the template matching algorithm, it is necessary to prepare the measured data sets. Then the algorithm has been tested with artificially generated data by adding normally distributed noise to the model data points; and in succession the measured data points have been used for further testing.

5.1.1 Data Preprocessing

The aim of data preprocessing is the location and elimination of outliers as well as generating a regularly distributed contour.

Input data for the preprocessing process is the measured data set, the output is a sorted contour, where all points are regularly distributed along the perimeter (see Figure 5.2). The algorithm is described in Section 4.1.1.

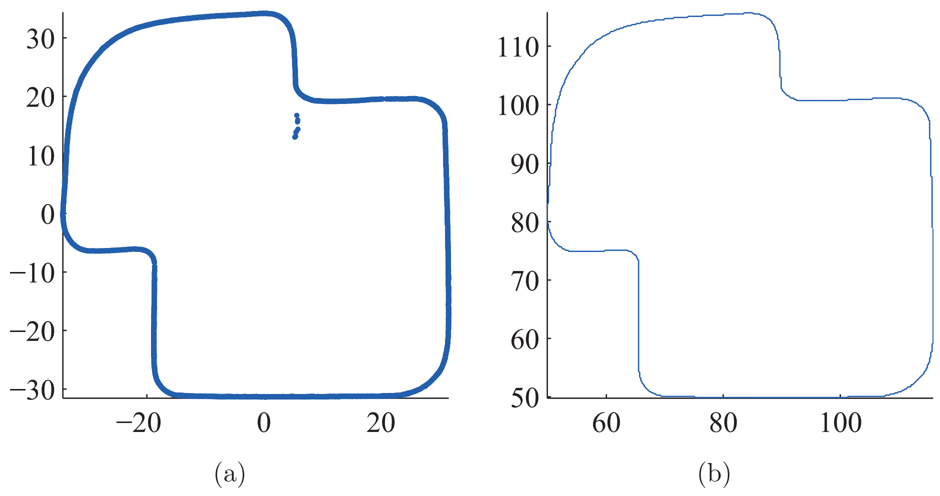


Figure 5.2: Measured data as input of the preprocessing algorithm (a) and the contour data as output (b).

In some cases a part of the contour is not visible during measurement, e.g. there are holes in the profile. During the operation of the algorithm the morphological operation dilation and erosion are used. This is for creating a continuous contour which will get filled in the next processing step. The structure element defines the maximum distance between the contour points, which can be closed. If parts of the contour are not visible, the distance get greater than the morphological operation can close the contour. So in the next step, there is no continuous contour which can be filled to get a solid object.

The second row in Figure 5.3 shows that the contour can not be filled any more. So the contour of the object also can't be computed and in succession the template matching algorithm can not be processed.

In conclusion, this algorithm is suitable for all closed contours.

5.1.2 Template Matching with Artificially Generated Data

For testing the template matching algorithm at first artificially generated data, including 2000 data points, are used instead of measured data points. The contour was shifted in x axis and y axis at 10mm and was rotated around the origin by 10 degrees. Figure 5.4 shows the results of this algorithm. Tables 5.1, 5.2 and 5.3 show the results in numbers, using noise levels of 0.1, 0.2, 0.5 and 1.0.

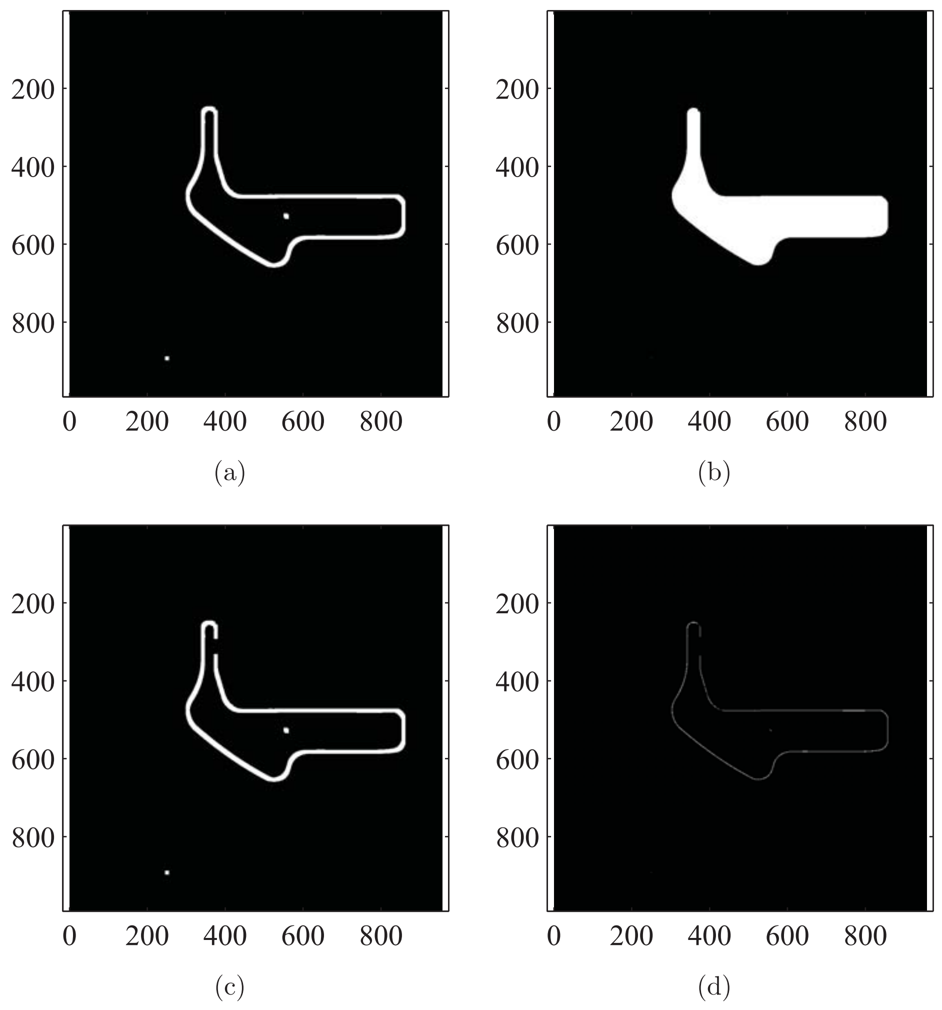


Figure 5.3: In the first row, a complete contour was processed, while the same contour in the second row has a hole in its surface. Column 1 is after dilation and column 2 is after filling the contour.

Profile 1						
		Value	Noise 0.1	Noise 0.2	Noise 0.5	Noise 1.0
x shift	$[mm]$	10.000	9.998	10.002	10.043	10.095
y shift	$[mm]$	10.000	10.021	9.973	9.964	9.912
angle	$[°]$	10.000	10.279	10.561	10.175	9.953

Table 5.1: Template matching results of profile 1

The matching results of profile 1 are very good. This can be reasoned by the small geometry and the high number of 2000 data points. All shifting values are in a range of $\pm 0.1mm$ as well as the angle is in a range of $\pm 0.5°$.

Matching the x axis in profile 2 gave an error of -1.1 to -1.3mm, whereas the y axis was

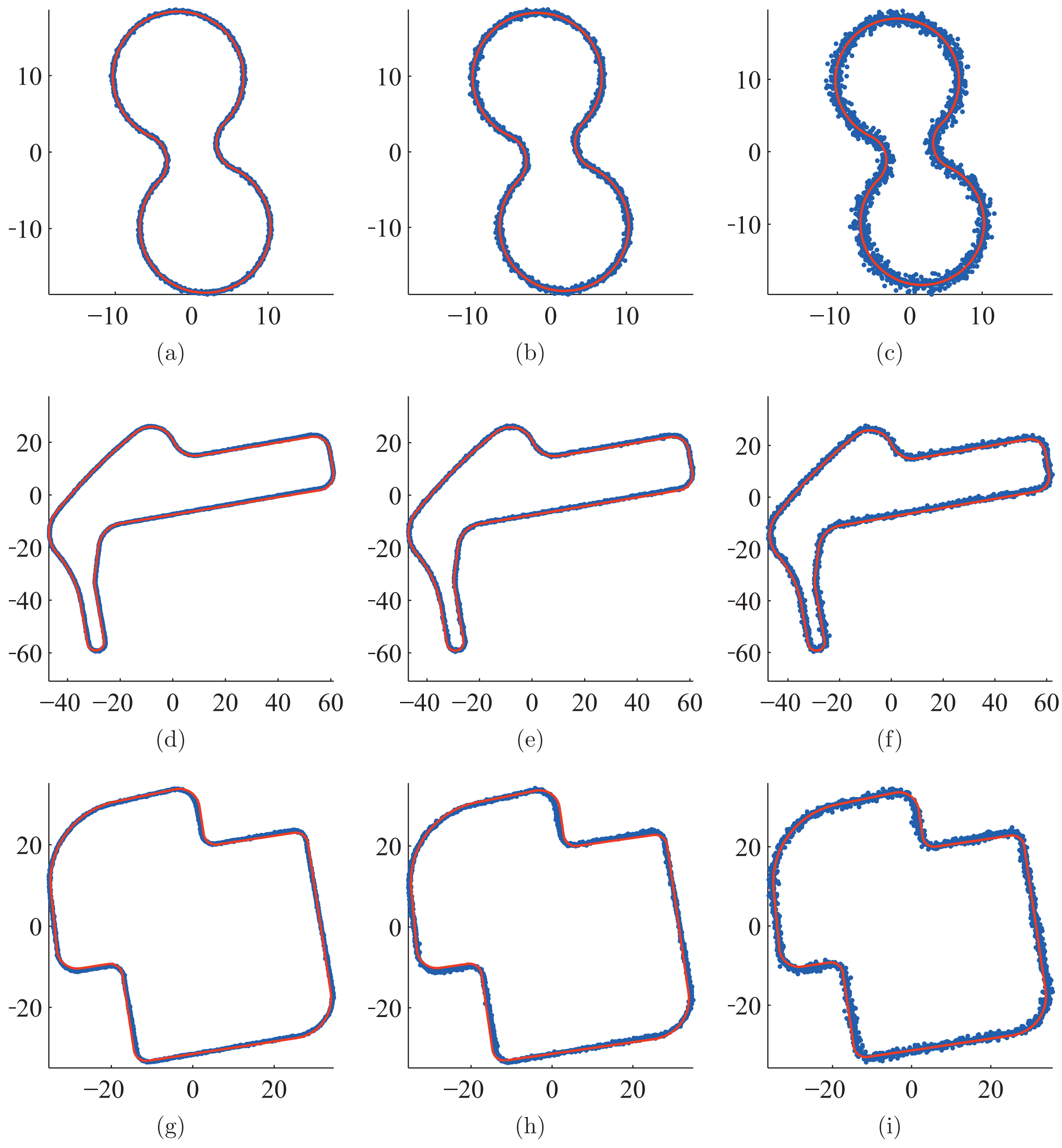


Figure 5.4: Result of the template matching algorithm using artificial generated noisy data points. The rows show the profiles 1-3, the columns show the noise levels 0.1, 0.2 and 0.5 (from left to right).

with an error of +0.3 to - 0.5mm. Also the angle error was greater than in profile 1 with a value of -0.3 to -0.7 degree.

Shifting profile 3 to the expected position was no problem with the developed algorithm. The error was smaller than $\pm 0.1mm$, but the angle error was greater than $\pm 2.2°$.

In summary the error in shifting and rotating the contour is mostly depending on the

Profile 2						
		Value	Noise 0.1	Noise 0.2	Noise 0.5	Noise 1.0
x shift	$[mm]$	10.000	8.972	8.976	8.974	8.761
y shift	$[mm]$	10.000	10.339	10.234	10.096	9.559
angle	$[°]$	10.000	9.404	9.494	9.315	9.777

Table 5.2: Template matching results of profile 2

Profile 3						
		Value	Noise 0.1	Noise 0.2	Noise 0.5	Noise 1.0
x shift	$[mm]$	10.000	10.027	10.032	10.033	9.921
y shift	$[mm]$	10.000	10.056	10.083	9.989	10.097
angle	$[°]$	10.000	8.514	8.474	7.834	11.532

Table 5.3: Template matching results of profile 3

geometry of the contour. If the profile is more orientated along an axis, the shifting will work better, as seen at profile 3. If the profile is only orientated relative to one axis, the result will be better in the orthogonal axis to the orientation axis. This can be seen in profile 1 (orientated to the y-axis) and in profile 2 (orientated to the x-axis).

The rotation error is also depending on the geometry. If the profile is rotational symmetric, the rotation error is greater than on non symmetric geometries (see Profile 3). All errors (shifting and rotating) are independent of the noise level. For symmetric contours the boundary method was introduced in Section 4.1.3, because the result of the algorithm above is strongly depending on the location of outliers for symmetric profiles. The result of the boundary method can be seen in Section 5.1.4.

5.1.3 Template Matching with Measured Data

The results on testing with measured data points are a bit different to artificially generated data points. The data points include outliers and the geometry is similar, but not equal to the model data points. In Figure 5.5 different measured data sets have been processed with the algorithm.

The matching algorithm can only be evaluated visually, because the shifting and rotation parameters of the measured data are not known.

Profile 1 was matched very good in consideration of the big deviation between model data points and measured data points. The measured data points were still in range of the bandwidth and the segments could be allocated to the different segments in the next step.

Because of the dimensions of profile 2 and profile 3, the angle takes a very large influence on the result of the algorithm. With an error of one degree, the deviation between model data

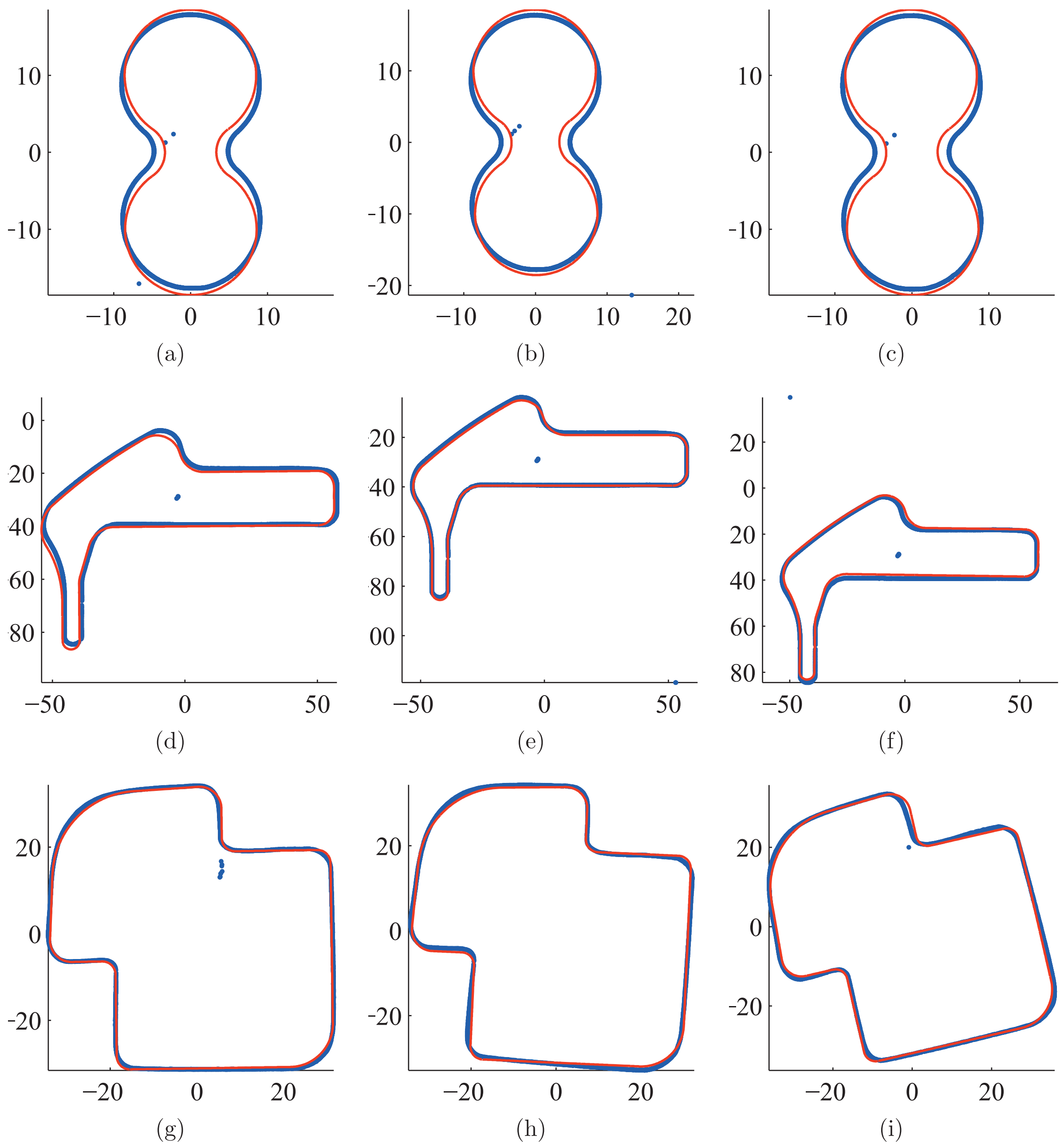


Figure 5.5: Template matching results, tested with real measured data points for profile 1-3.

points and measured data points would be too big to identify some segments. Especially the circles with a small radius or segments with less data points are affected.

Outliers and measurement noise took no influence on the result of matching of these measured data points.

5.1.4 Template Matching for Symmetric Profiles

To test the boundary method a symmetric profile is introduced and shown in Figure 5.6. This method is only tested with artificially generated data.

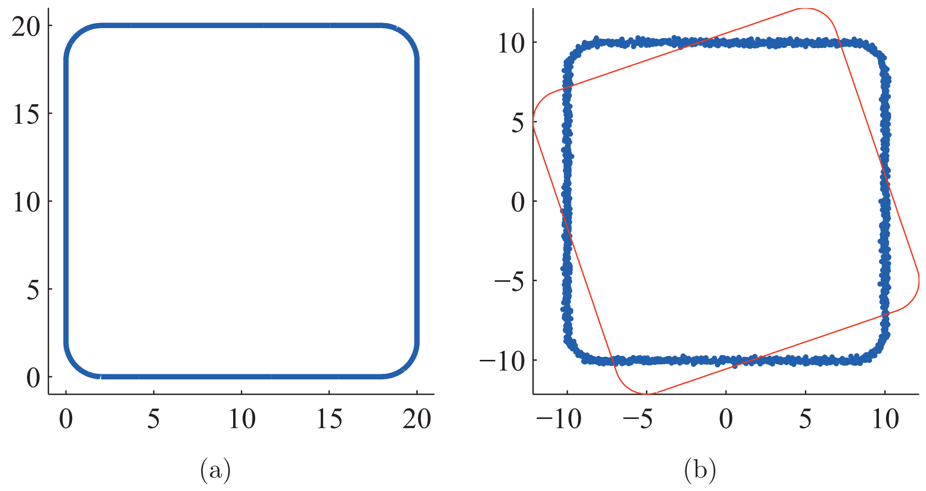


Figure 5.6: (a) shows the introduced symmetric profile. Matching the profile, as done in Section 4.1.2, gives the result shown in (b).

As shown in Figure 5.6b, the original template matching algorithm doesn't work for symmetric geometries. In order to fix this problem, the boundary method was introduced. Figure 5.7 shows the same test as done before: Shifting the original data in x and y at 10mm and rotate at 10 degree. Then the boundary method was performed. During data generation the same noise level of 0.1, 0.2 and 0.5 was used. Table 5.1.4 shows the results.

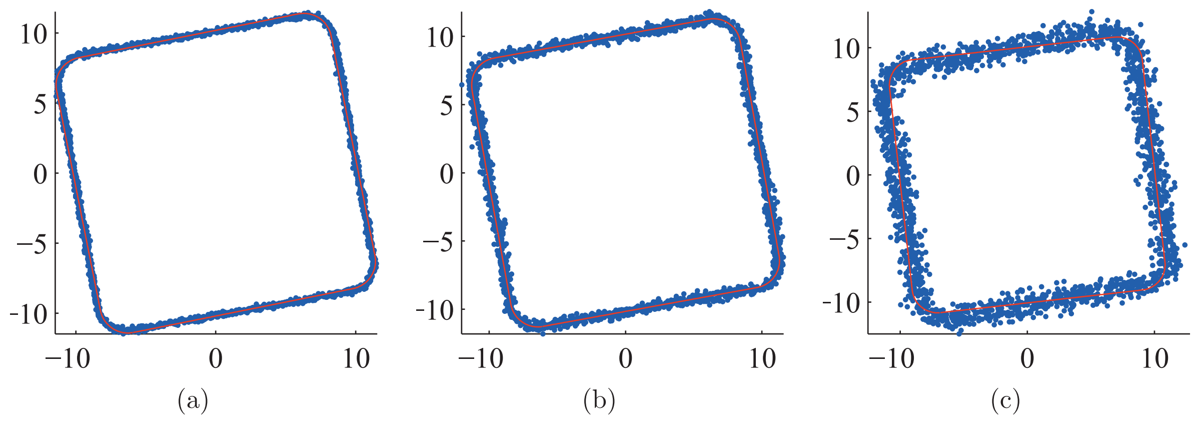


Figure 5.7: Result of the boundary method for template matching at different noise levels of 0.1, 0.2 and 0.5.

The results are strongly depending on the noise level. On increasing noise, the shifting error and especially the angle error also increase. So if there are any outliers, the algorithm

Point Symmetric Profile						
		Value	Noise 0.1	Noise 0.2	Noise 0.5	Noise 1.0
x shift	$[mm]$	10.000	9.933	10.029	10.351	11.380
y shift	$[mm]$	10.000	10.049	9.981	9.775	9.137
angle	$[°]$	10.000	9.652	10.153	11.657	16.546

Table 5.4: Template matching results of the symmetric profile

is not stable any more. In order to fix this problem outlier elimination techniques, e.g. smoothing filters, have to be applied.

5.2 Data Segment Assignment

In order to set the data to the particular segments, an accurate performance of the template matching algorithm is assumed. If the dimensions of the measured profile exceed the introduced bandwidth, these points are handled as outliers. Figure 5.8a shows the result of the boundary point computation. The points marked green, are the start and end points of the segments from CAD data. Based on these points the inner boundary (marked with red points) and the outer boundary (marked with black points) are computed. Finally, the points inside the boundary got assigned to the particular segments (see Figure 5.8b). The outliers inside the contour are successfully eliminated.

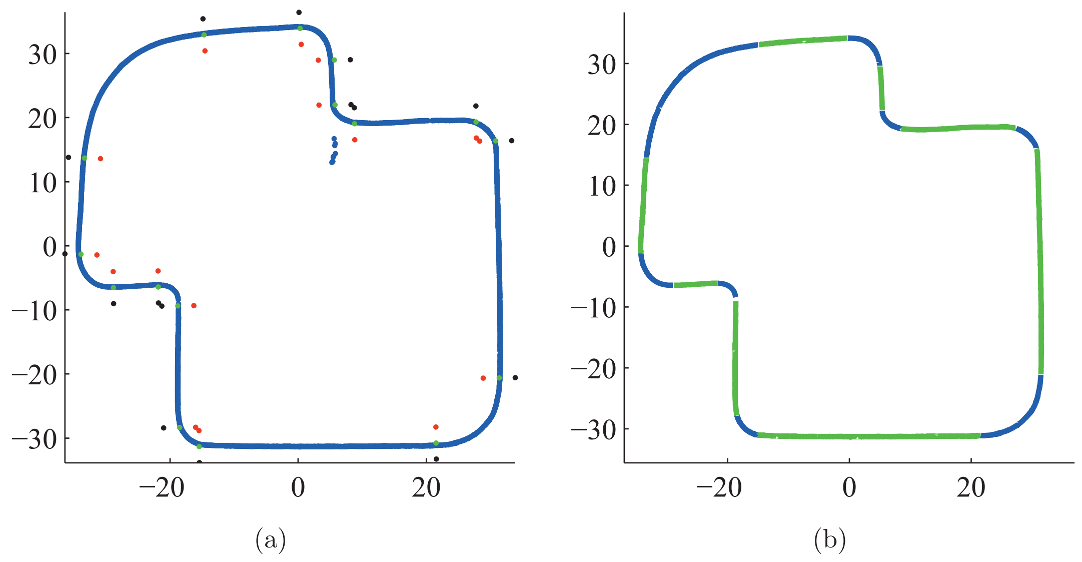


Figure 5.8: (a) shows the points of the inner and outer boundary and (b) shows the segmented profile.

5.3 Data Sorting

For finding the correct transition points, the data in the different segments have to be sorted along an unique direction. The results for line sorting are shown in Figure 5.9. The line is sorted along the x axis of the coordinate system. The left figure shows a typical overlapped data set, which is sorted in the right figure.

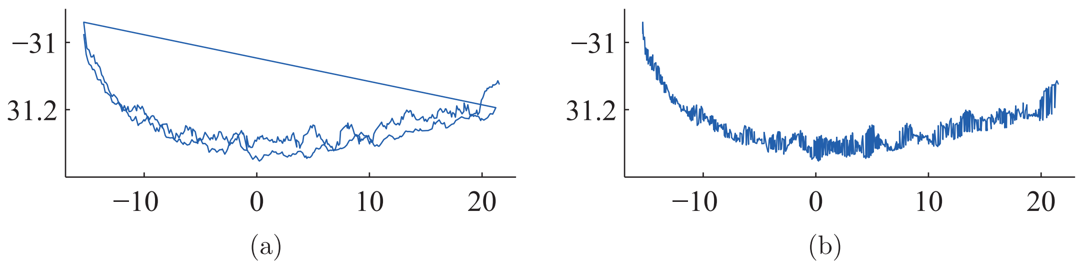


Figure 5.9: Unsorted (a) and sorted (b) line segment.

The unsorted circle arc data set in Figure 5.10a is sorted around the expected center point of the circle from CAD data.

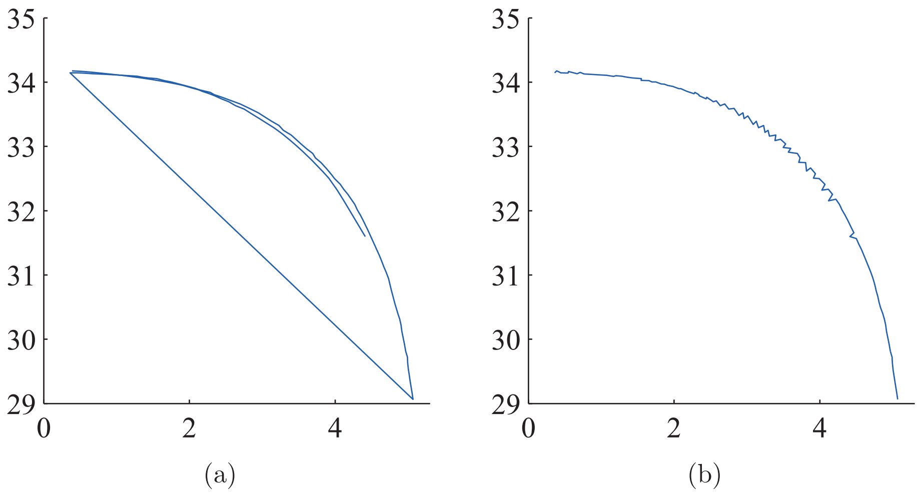


Figure 5.10: Unsorted (a) and sorted (b) circle arc segment.

There were no problems in sorting the segment data using the algorithm developed in Section 4.3.

5.4 Finding Transition Points

5.4.1 Finding the Transition Point by the Residuals of Circles and Lines

After template matching, assigning the data to the segments and sorting the data, the profile was segmented as shown in Figure 5.11. This algorithm was only tested with profile 3. All transition points, marked with a red x in the figure, are matched from the model segments. They are numbered from 1-16.

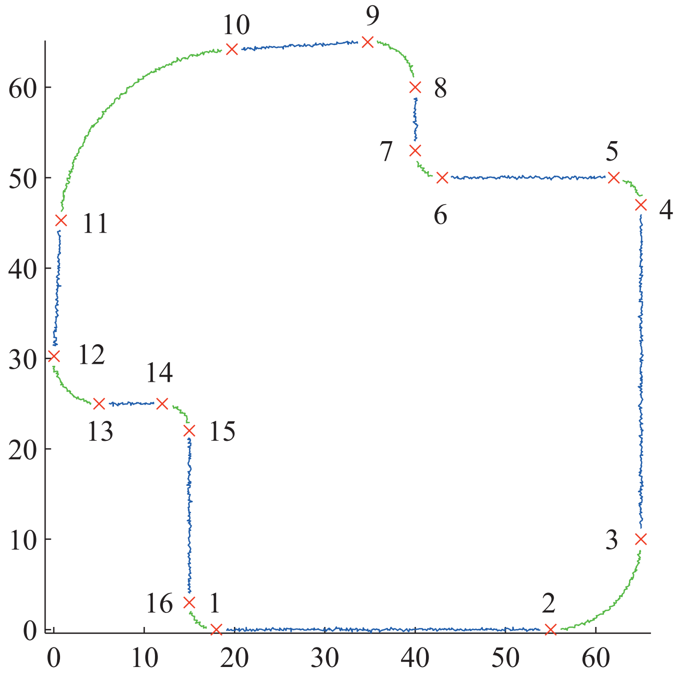


Figure 5.11: Profile matched and segmented by the model segments. The transition points are marked with a red x and numbered from 1-16.

In Figure 5.12 the residuals of transition point 2 and 3 as well as 15 and 16 are exemplarily shown. The residual on the transition of the left element to the right is marked with a blue line, the residual on the transition of the second element to the first is marked with the red line. The green line in this figure is the sum of the residuals. The number of points used for the iteration process is depending on the number of points available in the different segments. On decreasing number of points in one of the concerning segments, also the number of points for iteration will decrease until a predefined minimum.

In Figure 5.12a the sum of the residuals has a definitive minimum. If all of the points on this minimum level are deleted, the transition is clearly defined. In Figure 5.12b, c and d there are less points in use for iteration, because a line or circle segment is very small. There is no definitive minimum in the green curve, and so the transition can’t be defined. In this case all points used for iteration get deleted and as a result, the segments become very small. On measuring the profile, a circle fit is performed on this segments. This will

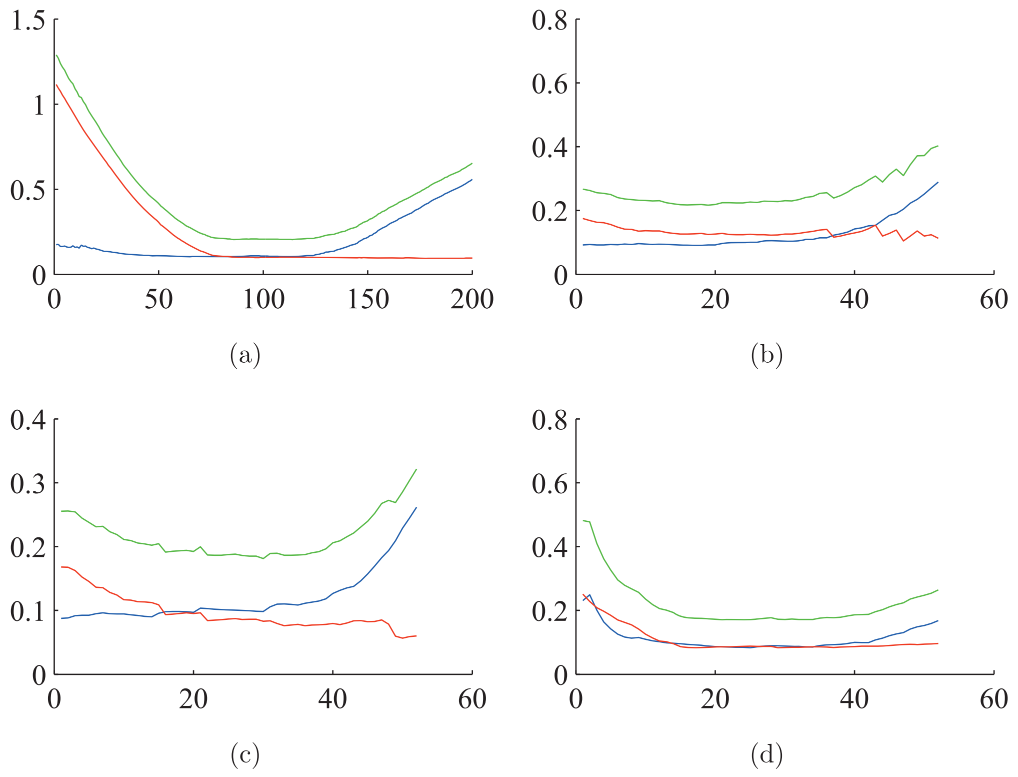


Figure 5.12: (a) shows the residuals of transition point 2, (b) belongs to transition point 3, (c) and (d) belongs to transition point 15 and 16, respectly

cause an increasing error on a decreasing number of points of the circle segment.

The result of the algorithm is shown in Figure 5.13. Due the detection of points at areas where the residuals are nearly equal, there are large distances between the segments.

5.4.2 Finding the Transition Point by Fitting Circles

This method was developed as an alternative to the algorithm *Finding the transition point by the residuals of circles and lines*. Using this algorithm, only 10 data points around the transition point get deleted. This is a big advantage of this algorithm. The circle segments in the measured data set look like ellipses. In compliance with this situation, the optimized transition point can only be defined by a tangency condition to the next segment.

At first also the template matching algorithm was performed. After segmentation by the model segments and sorting the data, this algorithm can be performed. In Figure 5.14a the blue line describes the profile segmented by the model segments. The red lines and circles are fitted into the different segments. The radius of the fitted circles is predefined by the model segment. There is no tangency condition implemented in this step.

The next step is to project the center points of the circles orthogonal on the lines. This yields the optimized transition point. In Figure 5.14b the transition points of the model

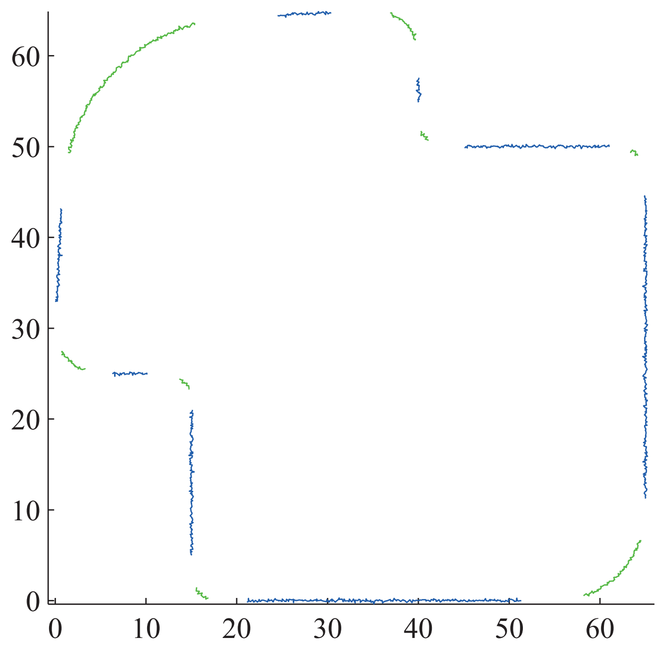


Figure 5.13: Profile matched and segmented by the residuals of circles and lines.

segments (green) and the new transition points, calculated by this algorithm (red) are shown. Now the different segments get recalculated by the new transition points. Around every point about 10 points are deleted to gain better results in measuring. Figure 5.15 shows segmented contours of the different profiles.

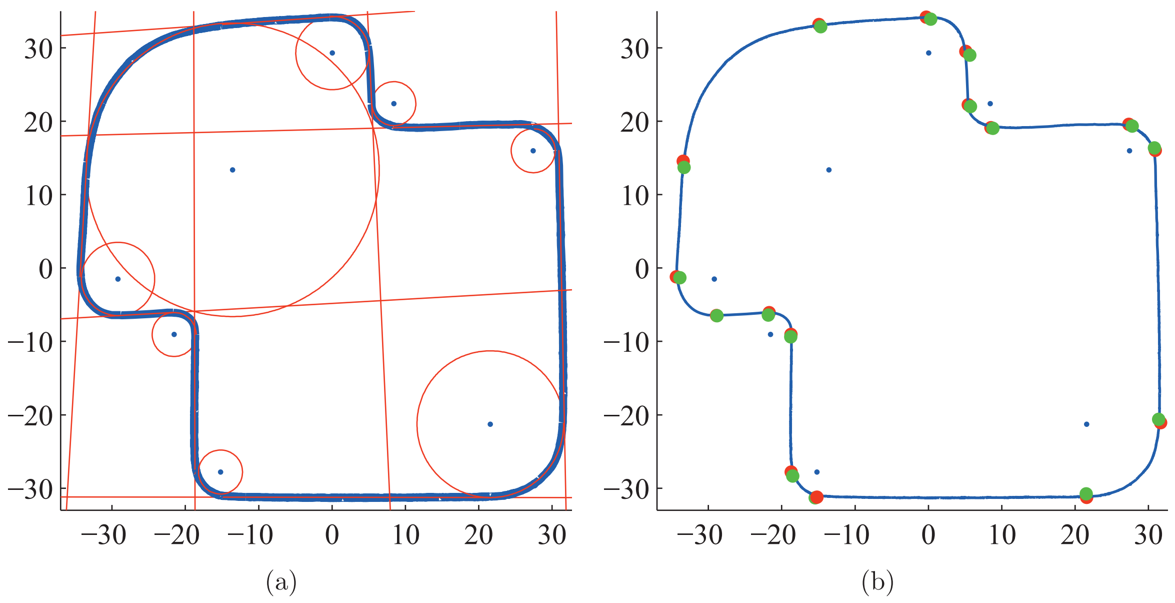


Figure 5.14: (a) Result of fitting circles and lines to the segments. (b) Result of the algorithm to find the intersection points by fitting circles and lines. The green points are the original transition points by the model segments, the red points are the new found transition points by this algorithm.

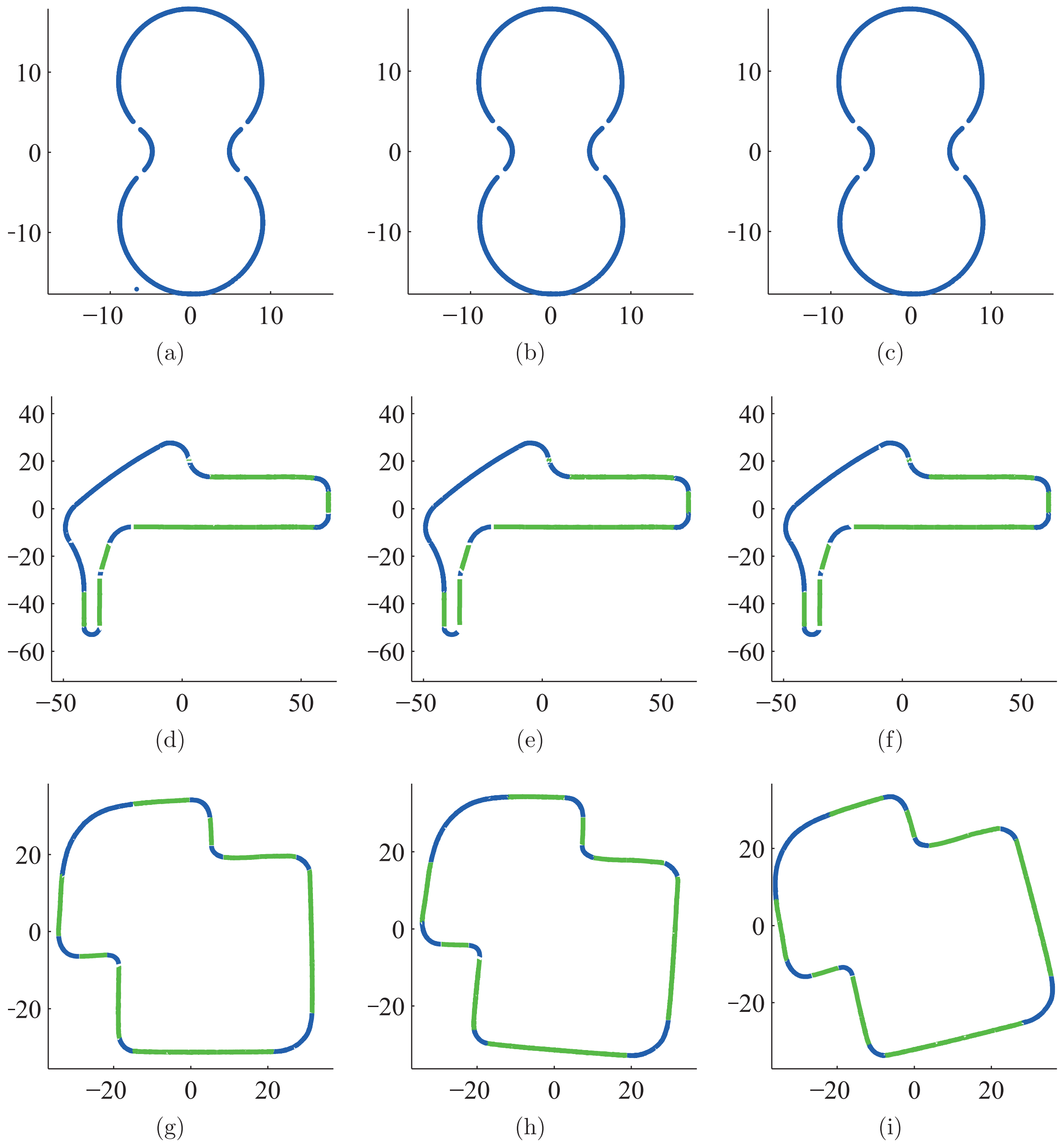


Figure 5.15: Final result of the registration and segmentation algorithm. The profiles 1-3 are tested with three different measured data sets. Line segments are shown with a green line, circle segments are shown with a blue line.

Chapter 6

Conclusion

With the new developed algorithm all profiles tested in this thesis could reliably be verified.

The representation of data points as an image for data preprocessing is only suitable for closed profiles, open profiles cannot be measured using this algorithm. In succession, template matching is also only working well for all closed and non-symmetric profiles.

This algorithm is stable for the segmentation, sorting of the data, and identification of the transition points between the geometric objects. The separation of outliers from gaussian perturbation is a critical issue. Some portions of the gaussian perturbations may be identified as outliers. This can lead to a bias in the fit results.

In conclusion, it could be shown, that the newly developed algorithm is working well for all profiles tested in this thesis. The presented algorithm is suitable for all non symmetric profiles. The eigenvector method fails for symmetric profiles since there is no unique orientation. For this reason further research is proposed (see Chapter 7) before applying the algorithm in a productive environment.

Chapter 7

Future Work

7.1 Improvements of the Existing Algorithm

7.1.1 Data Preprocessing

As it was shown in Section 5.1.1 the preprocessing algorithm is not suitable for profiles with holes in its surface. The reason for this are the morphological operation in combination with the filling operation.

In future steps the preprocessing should not be done by converting the measured data into an image, since this causes a loss in accuracy and introduces further errors. Instead, at first outliers should be eliminated and then the measurement noise can be compensated by performing a smoothing filter operation.

7.1.2 Template Matching

Template matching works well for artificially generated data. For measured data, the eigenvector method is inexact, because the center points of the measured data set and the model data does not match. A suggestion would be to match only the circle segments and not the whole contour. This suggestion would also work for symmetric profiles.

7.2 Developing of Further Algorithms

During this work some difficulties were observed with the algorithm under very specific conditions. There anomalies justify doing research on alternative solutions. The following two issues have been identified as the most important areas for research:

7.2.1 Localization via Correlation

A different approach to separate the line and circle segments is to represent the data points as an image and localize the circle segments. Therefore the model data points of each circle segment which should to be found in this image is also represented as a template image. By performing the cross-correlation, every position where the template image and the image match is indicated by a local maxima and yields an estimate of the location of the circle segment. So the location of the circle segment can be estimated.

This correlation has to be done for every circle segment known from the model data. Now, this circle segments can be connected, using the tangent condition between the segments, to generate the closed profile. Based on this tangent points on the circle, the transition points between the different segments can be estimated.

7.2.2 Projection onto Basis Functions

Another segmentation approach is the iterative projection of the data points onto the basis function (e.g. a line) and partitioning the data points into smaller segments. The following steps are performed in each iteration step:

1. Finding the major and minor orientational axes of the data;
2. Separate the geometry along the minor orientational axis;
3. Project the separated data on the selected basis function;
4. Compute the residual of the projection;
5. Check if the residual is lower than an absolute error, or if a numerical border is reached.

If the condition in point 5 is fulfilled, no further computation is needed and the basis function describing to the data points is found. Otherwise the iteration is started again using one portion of the geometry as initial geometry.

In case of a line used as basis function, the whole geometry is described as concatenation of line segments. Many concatenating small line segments indicates the presence of a circle segment. Finally the different segments found by this algorithm have to be associated to the line and circle segments, defined in the model data.

Appendix A

Matlab Source Code

A.1 Template Matching

```matlab
function [x_shift_original,y_shift_original,x_shift_kontur,y_shift_kontur,a,sym] = template_matching_fn_svd(pointsOrig,points)

% Uses (syntax) :
%   [x_shift_original,y_shift_original,x_shift_kontur,y_shift_kontur,a,sym] = template_matching_fn_svd(pointsOrig,points)
%
% Input Parameters :
%   pointsOrig := template data set
%   points := measured data set
%
% Return Parameters :
%   x_shift_original, y_shift_original := translation of the template data set
%   x_shift_kontur, y_shift_kontur := translation of the measured data set
%   a := angle between the data sets
%   sym := symmetry of the template data set


minx = min(points(1,:));
miny = min(points(2,:));

x = 0 - minx +50;
y = 0 - miny +50;

T = [ 1 , 0 , x ;
      0 , 1 , y ;
      0 , 0 , 1];

impoints = T * points;

impoints1 = impoints;

impoints = impoints.*5;

maxX = max(impoints(1,:))+100;
maxY = max(impoints(2,:))+100;

% convert to an image

image = [zeros(maxY,maxX)];

impoints = round(impoints);

for i=1:length(impoints)
    image(impoints(2,i),impoints(1,i)) = 1;
end

% Morphologial operators

image = imdilate(image,[ones(10,10)]);
image = imdilate(image,[ones(10,10)]);
image = imfill(image,'holes');
image = imerode(image,[ones(10,10)]);
image = imerode(image,[ones(10,10)]);
```

```

% labeling

[labels,noObjects] = bwlabel (image);

for i=1:noObjects
    blankimage = zeros (size(image));
    indices = find (labels == i);
    blankimage(indices)=1;
    area1(i) = length (indices);
end
[vals, ind] = sort( area1 );

imageLabel = zeros (size(image));
indices = find (labels == ind(noObjects));
imageLabel (indices) = 1;

fig1 = figure;
hold on;
kontur = contour(imageLabel,1);
hold off;
close;

kontur = kontur(1:2,2:end);

kontur(1:2,:) = kontur(1:2,:)./5;
kontur(3,:) = ones(1,length(kontur(1,:)));

T = [ 1 , 0 , -x ;
      0 , 1 , -y ;
      0 , 0 , 1];

kontur = T * kontur;

% original measurement of the angle

x_shift_original = mean(pointsOrig(1,:));
y_shift_original = mean(pointsOrig(2,:));
x_shift_kontur = mean(kontur(1,:));
y_shift_kontur = mean(kontur(2,:));

xk = x_shift_kontur;
yk = y_shift_kontur;
x  = x_shift_original;
y  = y_shift_original;

% angle model data

pointsOrig1 = pointsOrig(1:3,:);

T = [ 1 , 0 , -x ;
      0 , 1 , -y ;
      0 , 0 , 1];

pointsOrig1 = T * pointsOrig1;

[U,S,V]=svd(pointsOrig1');
sym = S(1,1) / S(2,2);
Vorig = [V(1,2);V(2,2)];

% angle measured data

kontur = kontur(1:3,:);

T = [ 1 , 0 , -xk ;
      0 , 1 , -yk ;
      0 , 0 ,  1];

kontur = T * kontur;
points = T * points;

[U,S,V]=svd(kontur');
Vmeas = [V(1,2);V(2,2)];

% compute the resulting angle

a_orig = atan2(Vorig(2),Vorig(1));
a_meas = atan2(Vmeas(2),Vmeas(1));

a = -(a_orig - a_meas);

if sign(a) == 1 AND abs(a) > pi/2
    a = a - pi;
end

ROrig = [cos(a)  , -sin(a), 0;
         sin(a)  , cos(a)  , 0;
         0       , 0       , 1];

pointsOrig1 = ROrig * pointsOrig1;
```

A.2 Template Matching for Symmetric Contours

```matlab
function [x_shift,y_shift,deltaPhi] = template_matching_fn_mimmax(pointsOrig,points)

% Usage:
%   [x_shift,y_shift,deltaPhi] = template_matching_fn_mimmax(pointsOrig,points)

% Input Parameters:
%   pointsOrig := template data set
%   points := measured data set
%
% Return Parameters :
%   x_shift, y_shift := translation of the measured data set
%   deltaPhi := angle between the data sets

% mean-free original data
pointsOrig1 = pointsOrig(1:3,:);
xmOrig = mean(pointsOrig1(1,:));
ymOrig = mean(pointsOrig1(2,:));

for i=1:length(pointsOrig1(1,:))
   pointsOrig1(1,i) = pointsOrig1(1,i)-xmOrig;
   pointsOrig1(2,i) = pointsOrig1(2,i)-ymOrig;
end

% mean-free measured data
points1 = points(1:3,:);
xm = mean(points1(1,:));
ym = mean(points1(2,:));

for i=1:length(points1(1,:))
   points1(1,i) = points1(1,i)-xm;
   points1(2,i) = points1(2,i)-ym;
end

[phiMin, phiMax] = ( findDataOrientation(points1(1:2,:)) );
[phiMinOrig ,phiMaxOrig] = ( findDataOrientation(pointsOrig1(1:2,:)) );

if phiMin > phiMax
    phiMin = phiMin - pi/2;
end
if phiMinOrig > phiMaxOrig
    phiMinOrig = phiMinOrig - pi/2;
end

deltaPhiMin = ( phiMinOrig - phiMin );
deltaPhiMax = ( phiMaxOrig - phiMax );

deltaPhi = mean([deltaPhiMax, deltaPhiMin]);

s = sin( deltaPhi );
c = cos( deltaPhi );
R = [c , -s , 0 ;
     s , c , 0 ;
     0 , 0 , 1 ];

pointsOrig = R * pointsOrig;

xmOrig = mean(pointsOrig(1,:));
ymOrig = mean(pointsOrig(2,:));

x_shift = xm - xmOrig;
y_shift = ym - ymOrig;
```

```matlab
function [phiMin, phiMax] = findDataOrientation( points )

% Usage:
%   [phiMin, phiMax] = findDataOrientation( points )

% Input Parameters:
%   points := data set of points
%
% Return Parameters :
% phiMin := minimum angle
% phiMax := maximum angle

noPoints = 360;
phis = linspace( 0, pi/2 , noPoints );

% calculate the dimension of the patch along each direction

for k = 1:noPoints

    c = cos( phis( k ) );
    s = sin( phis( k ) );
    R = [c, -s; s, c];
    %
    pointsR = R * points;
    dist(k) = max( pointsR(2,:) ) - min ( pointsR(2,:) );

end

% find the minimum and the maximum dimension, as this indicates the major orientation of the data set

[maxD, maxAt] = max( dist );
[minD, minAt] = min( dist );
phiMin = phis( minAt );
phiMax = phis( maxAt );
```

A.3 Data Sorting

```
function [Geometry] = sort_segment_data(Geometry)

% Usage:
%   [Geometry] = sort_segment_data(Geometry)

% Description: sorts the data in every segment along an unique direction

% Input Parameters:
%   Geometry := Structure containing all geometry data

% Output Parameters:
%   Geometry := Structure containing all geometry data

k = [];
for j=1:1:length(Geometry.edges)

    if Geometry.edges(j).typeNo == 1
        p1 = Geometry.edges(j).Tp1;
        p2 = Geometry.edges(j).Tp2;
        richtung = p2-p1;

        postition = [];
        pos = [];

        for i=1:length(Geometry.edges(j).relData)
            v = Geometry.edges(j).relData(:,i) - p1;
            position(1:3,i) = Geometry.edges(j).relData(:,i);
            position(4,i) = dot(richtung,v)/sqrt(sum(richtung.^2));
        end

        pos = sortrows(position',4);
        Geometry.edges(j).relData = pos(:,1:3)';
        clear pos;
        clear position;
    else
        p1 = Geometry.edges(j).Tc;
        p2 = Geometry.edges(j).Tp1;

        richtung = p1-p2;

        postition = [];
        pos = [];

        for i=1:length(Geometry.edges(j).relData)
            v = Geometry.edges(j).relData(:,i) - p1;
            position(1:3,i) = Geometry.edges(j).relData(:,i);
            position(4,i) = dot(richtung,v) / ( sqrt(sum(richtung.^2)) * sqrt(sum(v.^2)) );
        end

        pos = sortrows(position',4);

        Geometry.edges(j).relData = pos(:,1:3)';

        clear pos;
        clear position;
    end
end
```

A.4 Fitting a Line

```
function [line,res] = fitLine2d(x, y)

% Usage:
%   [Line,Residual] = fitLine(x, y)

% Description: This routine fits a line to a set of data points and computes the residual

% Input Parameters:
%   x: the x-coordinates of the points
%   y: the y-coordinates of the points

% Output Parameters:
%   line: the line vector [l1 l2 l3]

if (nargin == 1)
    y = x(2,:);
    x = x(1,:);
else
    if (size(x,1) > 1)
        x = x’;
        y = y’;
    end
end
noPoints = length(x);

meanX = mean( x );
meanY = mean( y );

xs = x’ - meanX;
ys = y’ - meanY;

% Design Matrix A:

A = [xs, ys];

% apply ’singular value decomposition’ to the design matrix

[U, S, V] = svd(A);
l12 = V(:, end);

l3 = -l12’ * [meanX; meanY];

line = [l12; l3];

% compute the residual

res = S(2,2) / sqrt(noPoints);
```

A.5 Fitting a Circle

```
function [circle1,circleResidual] = fitcircle(pointsCircleX, pointsCircleY)

% Usage:
%   [Line,Residual] = fitLine(x, y)

% Description: This routine fits a line to a set of data points and computes the residual

% Input Parameters:
%   x: the x-coordinates of the points
%   y: the y-coordinates of the points

% Output Parameters:
%   circle1: the homogeneous circle coordinates [c1 c2 c3 c4]
%   circleResidual: The residual of the circle fit

x = pointsCircleX;
y = pointsCircleY;
w = ones(size(pointsCircleX));

x = x./w;
y = y./w;
w = w./w;

% Design matrix D:

D = [ x’.^2+y’.^2 , x’ , y’ ];

D0 = ones(size(x’));

Dhat = D - D0*pinv(D0) *D;

[U,S,V] = svd( Dhat );
```

```matlab

circle = V(:,end);

circle = [circle ; -pinv(D0)*D*circle ];

C1 = circle(1);
C2 = circle(2);
C3 = circle(3);
C4 = circle(4);
%
%
circle1.radius = sqrt( -circle(4) / circle(1) + circle(2)^2 / (4*circle(1)^2) + circle(3)^2 / (4*circle(1)^2) );
circle1.x = -C2/(2*C1);
circle1.y = -C3/(2*C1);

% Compute the residual

circleResiduals = sqrt((x - circle1.x).^2 + (y-circle1.y).^2) - circle1.radius;
circleResidual = sqrt( sum( circleResiduals.^2) / length( pointsCircleX) );
circle1.Residual=S(3,3);
```

A.6 Fitting a Circle with Constant Radius

```matlab
function [ Circle, iterations ] = fitCircleXYNL( Data, R , h )

% Uses (syntax) :
%   [ Circle, iterations ] = fitCircleNL( Data, R )
%
% Input Parameters :
%   Data := 2D homogeneous data in matrix form (2xn)
%   R    := known radius of the circle
%   h    := known starting point of the iteration
%
% Return Parameters :
%   Circle := the circle in conic vector form (1x6)
%   iterations := the number of iterations required for convergence.

x = Data(1,:)';
y = Data(2,:)';

epsilon = 1e-9 ;
maxIterations = 10;
iterations = 0 ;

% Gauss-Newton Iteration :

while ( norm(h) > norm(u)*epsilon ) AND (iterations < maxIterations )

    den = sqrt( ( u(1) - x ).^2 + ( u(2) - y ).^2 ) ;

    J = [ ( u(1) - x ) ./ den , ( u(2) - y ) ./ den] ;

    F = ( den - R ) ;

    h = -J\F ;

    u = u + h ;

    iterations = iterations + 1 ;

end

% Return the circle as a general conic :

Circle.M2 = [ 1, 0, 1, -2*u(1), -2*u(2), sum(u(1:2).^2) - R^2 ] ;
Circle.M = [1     0     -u(1);
            0     1     -u(2);
            -u(1) -u(2) sum(u(1:2).^2) - R^2];
```

References

[1] W. Alt. *Nichtlineare Optimierung.* Vieweg Verlag, Braunschweig, 2002.

[2] D. Chetverikov, D. Svirko, and D. Stepanov. Robust reorientation of 2d shapes using the orientation indicator index. Technical report, Computer and Automation Institute, Budapest, Hungary and Center for Applied Cybernetics FEE, Czech Technical University, Czech Republic, 2002.

[3] C. Demant, Streicher-Abel B., and Waszkewitz P. *Industrielle Bildverarbeitung.* Axel Springer Verlag, Berlin Heidelberg, 1998.

[4] E. Fauster. *Statistical Uncertainty Analysis for Image Processing Algorithms in Metric Vision Systems.* Doctoral thesis, University of Leoben, Austria, 2008.

[5] J. Gallier. *Geometric Methods and Applications for COmputer Science and Engineering.* Axel Springer Verlag, New York, 2002.

[6] A. Gfrerrer. *Analytische Projektive Geometrie.* Lecture notes, Graz University of Technology, Austria, 2003.

[7] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[8] V. H. S. Ha and J. M. F. Moura. Robust reorientation of 2d shapes using the orientation indicator index. Technical report, Digital Media Solutions Lab Samsung Information Systems America Irvine, Canada and Department of ECE Carnegie Mellon, University Pittsburgh, 2005.

[9] P. Habercker. *Praxis der Digitalen Bildverarbeitung und Mustererkennung.* Carl Hanser Verlag, München Wien, 1995.

[10] M. Harker, P. OLeary, and R. Neumayr. Profile measurement via circle-line spline fitting. *IEEE Trans. Instrum. Meas.*, pages 1531 – 1536, 2009.

[11] R. H. Kraft. System and method of point matching measured positions to template positions. Technical report, Rudolph Technologies Inc., Flanders, USA, 2004.

[12] D. Marsh. *Applied Geometry for Computer Graphics and CAD.* Axel Springer Verlag, London Berlin Heidelberg, 1999.

[13] R. Ofner. *Three-Dimensional Measurement via the Light-Sectioning Method*. Dissertation, University of Leoben, Leoben, Austria, 2000.

[14] P. Schalk. *Metric Vision Methods for Material and Product Inspection*. Doctoral thesis, University of Leoben, Austria, 2007.

[15] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Brooks/Cloe Publishing Company, 2nd edition, 1999.