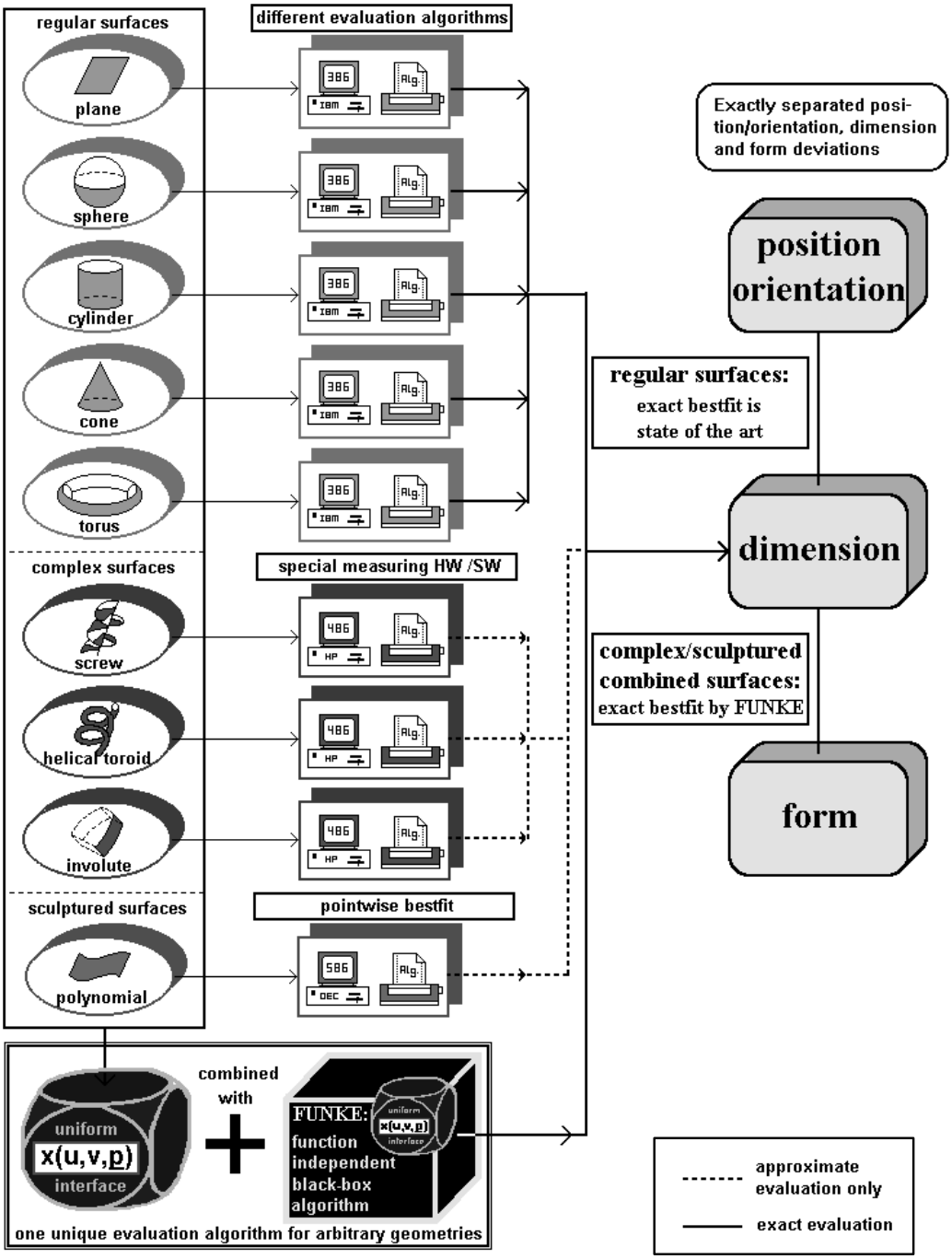
Diss. ETH No. 11319

Three Dimensional Feature Independent Bestfit in Coordinate Metrology



David Sourlier

1995

Diss. ETH No. 11319

Three Dimensional Feature Independent Bestfit in Coordinate Metrology

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH

for the degree of
Doctor of Natural Sciences

presented by
David Michael Sourlier
Dipl. Phys. ETH
born January 29, 1959
citizen of La Scheulte, Bern

accepted on the recommendation of
Prof. Dr. Walter Gander, examiner
Prof. Dr. Fritz Rehsteiner, co-examiner
Dr. Wolfgang Knapp, co-examiner.

1995

Acknowledgements

I especially thank Prof. Dr. W. Gander, head of the Institute of Scientific Computing, for his competent assistant and for the continuous interest he has shown in my thesis at the Institute of Machine Tools and Manufacturing. His excellent hints and tips were of great importance for this work, which took place in the field where engineering, data processing and scientific computing meet.

I am indebted to Prof. Dr. F. Rehsteiner, head of the Institute of Machine Tools and Manufacturing for the interesting work I had at his institute and for accepting to be co-examiner. His excellent English knowledge made writing in a foreign language a lot easier for me.

The same thanks go to Dr. W. Knapp of the engineering office Knapp in Schleitheim (SH) who accompanied this thesis as a co-examiner with all his profound knowledge of dimensional metrology. Through all the time he willingly supported the Dimensional Metrology group as an external consultant.

In good memory I have also the various interesting and informative talks I had with Prof. Dr. D. Debra of Stanford University about my work during the time he was a guest professor at our institute.

Furthermore, profound discussions took place between my colleague Dipl. Ing. F. Bucher and me. I thank him for the positive criticism and for the interest he always contributed to my work.

In addition to others not mentioned here personally, I especially thank my family for the patience they had with me through all the time.

Danksagung

Prof. Dr. W. Gander, Vorsteher des Instituts für wissenschaftliches Rechnen, danke ich für sein kompetentes Engagement und das Interesse mit welchem er meine Arbeit am Institut für Werkzeugmaschinen und Fertigung stets begleitet hat. Seine hervorragenden Tips und Hinweise waren äusserst wichtig für das Zustandekommen dieser Arbeit, welche im Grenzgebiet zwischen Ingenieurwissenschaft, Informatik und Numerik angesiedelt ist.

Prof. Dr. F. Rehsteiner, Vorsteher des Instituts für Werkzeugmaschinen und Fertigung verdanke ich die interessante Arbeit am Institut sowie das Korreferat; Seine vertieften Englisch-Kenntnisse waren mir eine grosse Hilfe bei der Abfassung dieser Arbeit in einer Fremdsprache.

Derselbe Dank geht auch an Dr. W. Knapp vom Ingenieurbüro 'Knapp' in Schleitheim (SH), welcher das Korreferat mit seinen profunden Kenntnissen der dimensionellen Messtechnik versehen hat und welcher dem Sektor 'Messtechnik' schon früher jeweils mit gutem Rat und Tat zur Seite gestanden ist.

In bester Erinnerung habe ich weiter die interessanten und lehrreichen Gespräche, welche ich mit Prof. Dr. D. Debra von der Stanford University über meine Arbeit führte, in der Zeit wo er an unserem Institut als Gast-Professor tätig war.

Ebenso vertiefte Diskussionen fanden mit meinem Kollegen Dipl. Ing. F. Bucher statt. Ihm danke ich für die allzeit positive Kritik und das stetige Interesse, welches er meiner Arbeit entgegenbrachte.

Neben all denen, welche hier nicht namentlich erwähnt sind, danke ich insbesondere meiner Familie für die grosse Geduld, welche sie die ganze Zeit über mit mir aufbringen musste.

Abstract

For the exact Gaussian bestfit in coordinate metrology a *feature independent* solution method is proposed: For arbitrary *standard*, *complex*, *compound* or *sculptured* features a bestfit module can be generated in a short time by just interfacing their describing parametric function in its most simple form (e.g. $\mathbf{x}_{plane}(u, v) = (u, v, 0)$) to the proposed, *generally* applicable bestfit software **FUNKE**.

In contrast to standard surfaces, *exact* Gaussian bestfit for *complex*, *compound* or *sculptured* features has not been – up to now – part of today's standards nor part of the commercial state-of-the-art software.

The *feature independence* is achieved by consistently splitting feature-*specific geometry* description from the *generally* treatable *position/orientation*-description.

Further advantages can be achieved by substituting the normally needed *implicit distance function* by a *parametric* surface function. For example, there are improved treatment of surface or measurement specific *constraints*, *probe radius* correction and *deflection* compensation as well as simplified *interfacing* with CAD-systems.

By appropriate solution methods which *exploit* the sparse structure of the resulting system of equations, we solve the problem that for each measuring point two *additional* unknowns, u_i and v_i are introduced using a parametric representation. Computational *costs* and *memory* demands can then be restricted to a *linear* dependence on the number of measuring points.

Finally, together with an *easy* test method for standard surfaces, a *probing simulation* is proposed, which should not only be restricted to verify the exact implementation of **FUNKE** but could also be used for the *verification* of arbitrary bestfit algorithms.

Kurzfassung

Für exakte Ausgleichsrechnung nach Gauss auf dem Gebiet der Koordinaten-Messtechnik wird eine *Flächen-unabhängige* Lösungs-Methode vorgeschlagen. Konkret heisst dies, für beliebige *reguläre*, *komplexe*, *zusammengesetzte* oder *frei* approximierte Flächen kann ohne weiteren Aufwand ein Modul für deren Best-Einpassung generiert werden. Dies geschieht durch blosses Anbinden der parametrischen Flächen-Beschreibung in einfachst möglicher Darstellung (z.B. $\mathbf{x}_{Ebene}(u, v) = (u, v, 0)$) an die *allgemein* verwendbare Ausgleichs-Software **FUNKE** (= **F**unktions-**U**nabhängige **N**ominalstellen-**K**orrigierte **E**inpassung).

Im Gegensatz zu regulären Flächen ist *exakter* Ausgleich nach Gauss für *komplexe*, *zusammengesetzte* oder *frei approximierte* Flächen weder in den heutigen Normen-Werken verankert, noch Teil von kommerzieller Standard-Software.

Flächen-Unabhängigkeit wird erreicht, durch konsequente Abtrennung von flächen*spezifischer Geometrie*-Information von *allgemein* implementierbarer *Lage*-Information.

Zusätzliche Vorteile ergeben sich dank dem Ersatz der normalerweise notwendigen *impliziten* Distanz-Funktion durch die *parametrische* Flächen-Funktion. U.a. sind dies: Einfache und verbesserte Behandlung von Flächen- oder Messproblem-spezifischen *Nebenbedingungen*, *Radius*- und *Biegungs*-Korrektur sowie *Datenaustausch* mit CAD-Systemen.

Dem Problem, dass bei Verwendung der parametrischen Beschreibung für jeden Messpunkt zwei *zusätzliche* Unbekannte u_i, v_i auftreten, wird durch entsprechende Lösungsmethoden Rechnung getragen, welche die schwach besetzte Struktur des resultierenden Gleichungs-Systems *ausnützen*. So können *Rechenzeit* und *Speicher*bedarf auf *linearen* Anstieg mit der Anzahl Messpunkte begrenzt werden.

Zusammen mit einer *einfachen* Test-Methode für reguläre Flächen wird am Schluss eine *Antast-Simulation* vorgeschlagen, welche nicht nur bei **FUNKE**, sondern ganz allgemein auch für die *Verifikation* von beliebigen Ausgleichs-Algorithmen Verwendung finden könnte.

Contents

List of Figures

Chapter 1

Introduction

1.1 Preface

1.1.1 Relevance

Relevance of SW in CM

Dimensional Metrology is a science that may have existed even before human beings learned to write and count: the direct comparison of two dimensions (gauging) can take place at any time, by no means of letters or figures! On the other hand, the evaluation concepts of Coordinate Metrology (CM), based on full position/orientation, dimension and form analysis, have only been developed since computers learned to calculate efficiently. By this contrast the relevance of scientific computing and software for CM can be illustrated.

CMM-system consists of a SW and a HW part

A coordinate measuring system consists of two parts, hardware and software. The hardware of a coordinate measuring system is very similar to a numerically controlled (NC) machine in manufacturing. Here it will be treated only insofar as there are consequences for some software aspects.

SW makes CM flexible

The software turns a coordinate measuring machine (CMM) into a very versatile measuring tool. The *driver* software (like the hardware) allows for the same flexibility we have in NC-manufacturing in the automatic measurement as well. On the other hand, this is not true for the *evaluation* software.

Data evaluation: Reverse process of manufacturing

The evaluation of data is in some sense the reverse process of manufacturing. In manufacturing we produce a physical model based on a known mathematical model. The parameter bestfit as a key task of data evaluation in CM, however, determines the parameters of the mathematical model belonging to a given physical model.

Nominal values: Starting point for optimization

Measuring a surface, we know – in contrast to digitizing and reconstructing a surface from scratch – the surface class which the mathematical model is based on and nominal values of all unknown parameters. In some sense this could be an advantage and in some sense a restriction, however, we have to use this information as a starting point to search for the unknown exact parameters.

CM-flexibility restricted by the available SW

As explained above, parameter bestfit can be regarded from the mathematical

point of view as the reverse process of NC-manufacturing, so we cannot use the same or similar algorithms for both processes. While for standard surfaces such as planes, spheres, cylinders, cones and toroids, exact solutions and algorithms are well-known (e.g. [GS74], [For91], [BFH94]), for more complex and sculptured surfaces only approximation methods are in use today. However, for any surface type we need a specialized bestfit algorithm. The flexibility of CM is restricted by the availability of appropriate evaluation algorithms and software.

1.1.2 New Concept

Separating geometry from posit./orient. allows surface independent bestfit

This thesis shows a surface-independent algorithm which allows exact bestfit for arbitrary surfaces. We achieve this goal by separating geometrical description from position/orientation description. Thus only the function which defines the geometry is surface specific; everything not surface specific is integrated in the generally applicable algorithm.

Exact sculptured-surface algorithm

The algorithm differs from other universal algorithms, e.g. [GT92], by the fact that it performs not a pointwise fit but a real surface-fit, fully corresponding to the exact, surface-oriented algorithms that are state of the art in the CM of standard surfaces. Because it is based on a parametric surface representation $\mathbf{x}(u, v, \mathbf{p})$, it can be applied to any type of surface, also to sculptured surfaces, to complex surfaces (e.g. to ellipsoids, involutes, toroid-screws) or to compounds of surfaces (e.g. to a square block with unknown height, width and depth).

Costs

Due to its structure, it is not more cost-intensive for the bestfit of these special surface-types compared to conventional algorithms for standard surfaces. Thus it can be used without any drawbacks to standard measurement problems as well.

Standard problems

On the contrary, we have some advantages even there. For example, in addition to the desired bestfit parameters, it also delivers the contact points between probe and surface. This enables *compensation of probe tip deflection* not only for analogous probing devices but also for *switching* ones. Furthermore it allows the solution of a great variety of constraint problems in a quite simple way.

FUNKE

In the following we call the software based on this algorithm 'FUNKE'(SPARK) (**F**unktions-**U**nabhängige **N**ominalstellen-**K**orrigierte **E**inpassung), which means 'function-independent bestfit, correcting nominal surface coordinates'. This is because the nominal surface coordinates (u, v) are corrected here *simultaneously* with all other parameters. This is in contrast to algorithms calculating pointwise or serially, where the nominal values remain either implicitly fixed or where they are re-corrected independently of position/orientation and geometry parameters.

1.2 Background

1.2.1 General Parameter Fit

Parameter-fit as a standard problem

Parameter-fitting is a multifaceted task that is needed in many areas of natural and engineering sciences. For its general solution a lot of standard, mathematical methods exist and a great amount of literature has been published [Sch91].

Geometrical fit is more specific

A much more specific task is the position/orientation fit of a geometrical object with unknown dimension parameters into a given cloud of points in an euclidean 2D/3D space. This is exactly what is meant when we speak of bestfit in any CM-context. Beside CM this task may be more specified to areas like Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), surveying, geodesy, robotics, image processing and microscopy, or other sciences dealing with a geometrical space.

Simplified bestfit

For most purposes, e.g. for surface approximation in CAD or for fitting any functionality in natural sciences, an arbitrary algorithm can be taken into consideration which in some sense satisfactorily performs this job [Pra87]. For example we can insert the points (x_i, y_i, z_i) into any surface representation which describes the surface explicitly:

$$z = z(x, y, \mathbf{P}) \ , \tag{1.1}$$

or implicitly,

$$f(x, y, z, \mathbf{P}) = 0 \ , \tag{1.2}$$

and determine the parameters $\mathbf{P}$ given by these equations, based on the simple fact that a point lying on the surface satisfies the equations above.

1.2.2 Simple (Algebraic) Parameter Fit

Minimal point-number fit

In CM we can also go ahead like this, as long as we have the same number of points as unknown parameters. More often than for measured points, this fact appears for calculated points, e.g. if we have to determine the center of a circle passing through the measured centers of 3 bores. The individual bore we should measure with a measuring point number larger (normally 3-5 times) than the minimally required one. But also here we can use this simple type of parameter fit to calculate an approximate bestfit as a starting point for the exact minimization. Doing so, we can use any arbitrary implicit or explicit function, which describes the surface.

Bestfit as a constrained problem

An additional problem arises in the quite frequent case when these parameters $\mathbf{P}$ are not independent from each other. This normally leads to a constrained problem. For example, consider a cylinder, with $\mathbf{P} = (\mathbf{r}, \mathbf{n}, r)$, where $\mathbf{n}$ is the orientation vector of the axis, $\mathbf{r}$ its position vector (c.f.(7.5) in section 7.2) and r the radius of the cylinder. In order to obtain uniqueness while fitting $\mathbf{P}$ by (1.2) we need to impose constraints, e.g.:

$$\| \mathbf{n} \| = 1 \qquad \text{and} \qquad (\mathbf{r}, \mathbf{n}) = 0 \ .$$

That is to say the orientation vector $\mathbf{n}$ is defined to be a unit vector, and the position vector $\mathbf{r}$ is defined to be orthogonal to the orientation vector $\mathbf{n}$.

Redundancy of position and direction vector

This illustrates the redundancy which is involved in this kind of position/orientation description. A infinite number of position and direction vectors could describe the position/orientation of the same cylinder. However this is the usual description of a cylinder in CM, because from the geometrical point of view it is the most clear. On the other hand, from the computational point of view, redundancy is annoying.

1.2.3 Distance-Based (Geometric) Bestfit

Problematic nature with overdetermined problems

As soon as we are dealing with a point number larger than the minimally required one, a special situation arises in the bestfitting problem of CM. This normally results in an unsolvable, overdetermined system of equations. Nevertheless, a unique solution is required in order to obtain comparable results. We have then to define an unambiguous computational goal for bestfitting a surface into given measuring points [DIN86], [Bri89], [Lot90]. It is not enough to choose some numerically correct optimization method: only the optimum with regard to a well-defined optimization criterion can be accepted as the exact solution.

Objective function is based on the orthogonal distances

This optimization criterion is always based on the (orthogonal) distances between points and surface. Even if this objective function is much more difficult to impose ([GS74], [Spa86], [Pra87], [For87]), than a simple parameter estimation according to (1.1) or (1.2), it delivers a more suggestive result. An apparent example of this fact is given in [GGS94], on a theoretically constructed ellipse, fitted by these two ways. The two methods used lead to extremely different results. [PvH90] shows significant differences also on a practically relevant example between the simplified (linear) bestfit of a circle based on the equation

$$f(x, y, m_x, m_y, r) = (x - m_x)^2 + (y - m_y)^2 - r^2 = 0 \ , \tag{1.3}$$

and the exact, distance-based bestfit.

Theoretical example with a circle

With some theoretical considerations we can get a *quantitative* idea of the resulting differences. Assume a measuring point (x_i, y_i) in distance d_i from the unknown, but exactly bestfitted circle. This assumption can be expressed by the following equation

$$(x_i - m_x)^2 + (y_i - m_y)^2 = (r + d_i)^2 \ . \tag{1.4}$$

Least squares fit based on the unmodified circle equation (1.3)

We now use (1.3) to fit the unknown parameters m_x, m_y, r in the least squares sense, i.e. we minimize the norm of a residual vector $\mathbf{e} = (e_1, \ldots, e_n)$

$$\sum_{i=1}^{n} e_i^2 = \min \ ,$$

with the individual residuals

$$e_i = (x_i - m_x)^2 + (y_i - m_y)^2 - r^2$$

belonging to the respective measuring point (x_i, y_i).

Explicit residuals

Using (1.4) we recognize that minimizing

$$\sum_{i=1}^{n} e_i^2$$

is equivalent to minimizing

$$\sum_{i=1}^{n}((r+d_i)^2 - r^2)^2 = \sum_{i=1}^{n} 4r^2 d_i^2 + 4r d_i^3 + d_i^4 \tag{1.5}$$

instead of

$$\sum_{i=1}^{n} d_i^2 \ . \tag{1.6}$$

Geometrical interpretation

Minimizing expression (1.5) makes no geometrical sense. The term d_i^4 could be interpreted geometrically to such an extent, that big deviations d_i are weighted too much (compared to the minimization of (1.6)). Furthermore geometrically interpreting the term $4r^2 d_i^2$, we can state that minimization (1.5) would 'prefer' small circles. (Figure 1.1)

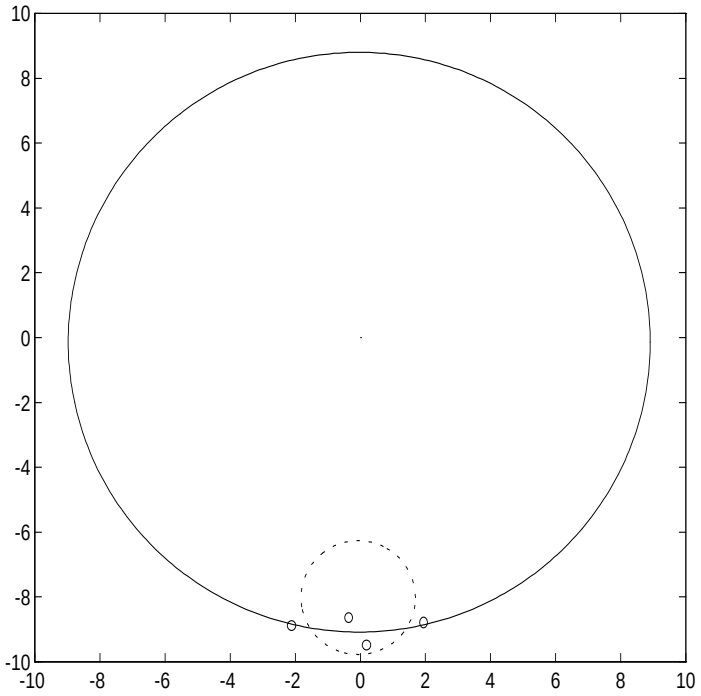


Figure 1.1: Algebraical (- - -) versus geometrical (—) bestfit

Distance-Function: Example circle

That is why an appropriate distance function is needed for surface fitting in CM. This is a special implicit function f, normalized in such a way that

$$f(x, y, z, \mathbf{P}) = d_{(x,y,z)} \ ,$$

while $d_{(x,y,z)}$ is the orthogonal distance of the point (x, y, z) to the surface. With the example of a circle we get the distance function:

$$f(x, y, m_x, m_y, r) = \sqrt{(x - m_x)^2 + (y - m_y)^2} - r = 0 \; .$$

Distance-Function: Special potential function

Such a distance function can be interpreted as a potential function with the potential $d_{(x,y,z)}$ and with equipotential surfaces equidistant to the nominal surface [Gaw89]. Later on (section 7.2) we will give a simple method for constructing this function for standard surfaces which are generated by straight lines and circles. The offset surface then belongs to the same surface class as the nominal surface.

Offset-curve of an ellipse

This statement virtually never applies to more complex or to sculptured surfaces. Even a simple surface such as the ellipse does not comply; its offset surface for example (with offset d) can be expressed by means of the surface normal $\mathbf{n}_{ellipse}$ as:

$$\underbrace{\begin{pmatrix} x(t) \\ y(t) \end{pmatrix}}_{\mathbf{x}_{offset}} = \underbrace{\begin{pmatrix} a \cdot \cos(t) \\ b \cdot \sin(t) \end{pmatrix}}_{\mathbf{x}_{ellipse}} + d \cdot \underbrace{\begin{pmatrix} \frac{b \cdot \cos(t)}{l(t)} \\ \frac{a \cdot \sin(t)}{l(t)} \end{pmatrix}}_{\mathbf{n}_{ellipse}} = \begin{pmatrix} (a + d \cdot \frac{b}{l(t)}) \cdot \cos(t) \\ (b + d \cdot \frac{a}{l(t)}) \cdot \sin(t) \end{pmatrix} \qquad (1.7)$$

with

$$l(t) = \sqrt{a^2 \cdot \sin^2(t) + b^2 \cdot \cos^2(t)} \; .$$

Ellipse degenerates to a circle

Only in the case $a = b$ (ellipse degenerates to a circle) does $l(t)$ become independent of t. This allows us to write (1.7) as

$$\underbrace{\begin{pmatrix} x(t) \\ y(t) \end{pmatrix}}_{\mathbf{x}_{offset}} = \underbrace{\begin{pmatrix} (a + d \cdot 1) \cdot \cos(t) \\ (a + d \cdot 1) \cdot \sin(t) \end{pmatrix}}_{\mathbf{x}_{circle}} , \qquad (1.8)$$

representing an other circle with radius $(a + d)$.

1.2.3.1 Distance Function for Sculptured Surfaces

Approximate distance function

In the past a lot of effort was put [GPS80], [Goc82], [WP83], [WG87], [Gaw89], [Goc90], [Xio91] towards overcoming this problem with constructing an approximating implicit function $f(\mathbf{x}, \mathbf{P}) \cong d_{(x,y,z)}$ with the required metric properties (distance function). Starting from an arbitrary implicit function $F(\mathbf{x}, \mathbf{P})$ (without this metric property) we can transform it in such an approximate implicit distance function $f(\mathbf{x}, \mathbf{P})$ by

$$f(\mathbf{x}, \mathbf{P}) = \frac{F(\mathbf{x}, \mathbf{P})}{\| \, \mathbf{grad}(F(\mathbf{x}, \mathbf{P})) \, \|} \cong d_{(x,y,z)} \; . \qquad (1.9)$$

Interpretation of the approximate distance formula

This formula is for example proposed in [GPS80], [Goc82], but without any explanation. To give an explanation, consider a point $\mathbf{x}^*$ lying on the surface and a point $\mathbf{x}$ very near to this point but not lying on the surface. We perform the linearization:

$$F(\mathbf{x},\mathbf{P}) + \mathbf{grad}(F(\mathbf{x},\mathbf{P}))^T \cdot (\mathbf{x}^* - \mathbf{x}) \cong \underbrace{F(\mathbf{x}^*,\mathbf{P})}_{0 \ on \ surface} = 0 \ . \tag{1.10}$$

If we additionally assume x^* as the footpoint belonging to $\mathbf{x}$, the vector $(\mathbf{x}^* - \mathbf{x})$ has length d and points in the same or in the opposite direction as $\mathbf{grad}(F(\mathbf{x}))$. In the case of opposite direction (i.e. the 'effective' deviation vector $(\mathbf{x} - \mathbf{x}^*)$ has *same* direction) we get

$$\mathbf{grad}(F(\mathbf{x},\mathbf{P}))^T \cdot (\mathbf{x}^* - \mathbf{x}) = - \parallel \mathbf{grad}(F(\mathbf{x},\mathbf{P})) \parallel \cdot \underbrace{\parallel (\mathbf{x}^* - \mathbf{x}) \parallel}_{d} \ . \tag{1.11}$$

Now we obtain (1.9) by inserting (1.11) in (1.10).

Approximating implicit function

Before (1.9) can be applied, an (arbitrary) approximating implicit function $F(\mathbf{x},\mathbf{P})$ has to be found. For sculptured surfaces this can be quite a difficult task. In [Goc82] it is performed for a turbine blade, for the profiles in parallel planes, not for the surface as a whole. It is done by means of a conformal mapping in the complex plane. As shown there, even then it is a difficult numerical problem to achieve an approximation with sufficient accuracy.

Separated directions

In [WP83] and [Gaw89] an approach called the method of 'separated directions' is used:

$$F(\mathbf{x}) = \sum_{l=0}^{k} q_l(z, r_l(x)) \cdot y^l \ , \tag{1.12}$$

(where q_l, r_l are arbitrary functions, e.g. splines) with low degree k (normally 2), allowing an approximation for quasi-cylindric workpieces (with small curvature in direction y). An additional problem [WP83] arises in the fact that there are points not belonging to the surface but still satisfying the equation $F(\mathbf{x}) = 0$.

CAD/CM: Different surface-representations

Furthermore it is a big disadvantage that the exact parametric surface representation as used in CAD for data exchange and geometrical operations, as well as in CAM (Computer Aided Manufacturing) and in CAQ (Computer Aided Quality Control) for driving a CNC-machine by calculating a sequence of surface-points [Sim91], [Neu91], [Kle93], has to be replaced here by an approximating implicit function [Goc82], [WP83], [Gaw89], just for measurements. Going ahead like this, each individual measurement problem has to be solved by its special way.

1.2.4 Bestfit with Orthogonal Distance Regression

Regression versus Orthogonal Distance Regression

A conventional parameter fit for fitting any functionality is often taking into account the uncertainty of the ordinate (output) values only while the abscissa (input) values are assumed to be exact. (This is known as 'regression' [Sch91].)

The methods of 'Orthogonal Distance Regression' (ODR), e.g. [BD89], [BR90], [GSW90] could remedy this problem. They take into account the uncertainty of the abscissa values as well.

This means that the objective function is not the Gaussian error

Normal regression

$$\sum_{i=1}^{n}(z_i - f(x_i, y_i, \mathbf{P}))^2 = \min ,$$

but the Gaussian error with corrected abscissa values (x_i^*, y_i^*)

ODR

$$\sum_{i=1}^{n}(z_i - f(x_i^*, y_i^*, \mathbf{P}))^2 = \min ,$$

while

$$\sum_{i=1}^{n}(x_i - x_i^*)^2 + (y_i - y_i^*)^2 = \min \ .$$

These methods can be adapted to the geometrical bestfit in CM [BCF92] as far as the surface can be represented in a explicit form. A real surface on a workpiece can often be described only partially in explicit form. For example, only one half of a sphere can be described explicitly. This is a drawback of the explicit function representation for CAD/CAM-purposes.

ODR for explicit functions

Using weights for the errors of abscissa and ordinate values and letting the ordinate weights go to infinity, ODR can also be applied to implicitly defined surfaces. This is shown for the example of an implicitly defined ellipse in [GGS94].

ODR for implicit functions

The algorithm FUNKE, derived in the following, is dealing with any kind of parametric surface description

ODR as a special case of FUNKE

$$x(u, v) , \quad y(u, v) , \quad z(u, v) .$$

If we can find a re-parameterization

$$(u, v) \mapsto (u^\star, v^\star)$$

in such a way that 2 of 3 coordinate functions are the trivial functions

$$x^\star(u^\star, v^\star) = u^\star \quad \text{and} \quad y^\star(u^\star, v^\star) = v^\star$$

together with a general, third function

$$z^\star = z^\star(u^\star, v^\star) = z(u(u^\star, v^\star), v(u^\star, v^\star)) ,$$

we arrive at a procedure quite similar to that used by ODR. This is obvious because this special case can be formulated as the ODR problem for the explicit function

$$z^\star = f(x^\star, y^\star) .$$

Invertible re-parameterization

This becomes possible if we can explicitly invert the re-parameterization

$$(u, v) \mapsto (u^\star(u, v), v^\star(u, v)) \equiv (x(u, v), y(u, v)) \ .$$

Then we obtain the third function $z^\star(u^\star, v^\star)$ by simply inserting the inverted functions $u(u^\star, v^\star)$ and $v(u^\star, v^\star)$ into $z(u, v)$.

Example: sphere

As an example, consider a sphere:

$$\begin{aligned} x(u, v) &= r \cdot \cos(u) \cdot \cos(v) \\ y(u, v) &= r \cdot \sin(u) \cdot \cos(v) \\ z(u, v) &= r \cdot \sin(v). \end{aligned}$$

Starting from the re-parameterization

$$u^\star(u, v) = r \cos(u) \cos(v) \qquad \text{and} \qquad v^\star(u, v) = r \sin(u) \cos(v)$$

we get the function

$$v(u^\star, v^\star) = \arcsin(\sqrt{1 - \frac{u^{\star 2} + v^{\star 2}}{r^2}}) \quad \text{or} \quad v(u^\star, v^\star) = -\arcsin(\sqrt{1 - \frac{u^{\star 2} + v^{\star 2}}{r^2}})$$

In this case we do not need the inverted function $u(u^\star, v^\star)$ because z is only a function of v. Inserting this into the z-function from the original surface description, we get:

$$\begin{aligned} x(u^\star, v^\star) &= u^\star \\ y(u^\star, v^\star) &= v^\star \\ z(u^\star, v^\star) &= \sqrt{r^2 - u^{\star 2} - v^{\star 2}} \quad (\quad \text{or} \quad -\sqrt{r^2 - u^{\star 2} - v^{\star 2}}). \end{aligned}$$

We recognize that we can describe only a semi-sphere with this special type of parameterization.

Most convenient parameterization can be chosen

Furthermore, if bestfit becomes possible based on arbitrary parameterizations, we are able to choose the most convenient one for each individual case. (See examples in the following sections.)

Sparse structure exploitation

ODR as well as the FUNKE algorithm can exploit the sparsity of the resulting system of equations.

Separation of posit./orient. from geometry

In the above special case, there still remains a difference between ODR and the FUNKE algorithm: ODR treats all unknown parameters in the same way, while FUNKE separates position/orientation from geometry (shape). This results in a special treatment of the corresponding parameters. On the other hand, it allows one to define a newly introduced surface by its purely geometric parameters only. Instead of giving its full description, including position/orientation parameters, we have only to specify rotation and translation symmetries.

Rank-deficiency caused by symmetries

This way we overcome the problem discussed in [BCF92] at the same time. There it is shown by the example of a straight line fit in parametric form that the

redundancy of the parametric description results in a singular system of equations. The surface-independent concept of FUNKE explicitly deals with the rotation and translation parameters, thus it is able to eliminate them according to the given surface symmetries. So we arrive at a non-singular system of equations.

FUNKE compared to ODR

As we will see in the following chapters, FUNKE could be designated as an ODR algorithm generalized to parametric surfaces or specialized (and optimized) to surfaces and curves in 2D and 3D. It deals especially with position/orientation bestfit, combined with dimension bestfit, making position/orientation bestfit surface-independent.

1.2.5 Bestfit Based on Parametric Representation

Advantages of parametric representation

Due to the problems with implicit and explicit function representations as discussed above, the advantage of being able to fit a surface in parametric form

$$x = x(u,v)\ , \quad y = y(u,v)\ , \quad z = z(u,v)$$

seems obvious. With parametric representation it is no problem to describe a closed surface (e.g. sphere) or workpieces containing different surfaces. It is a very transparent representation form and it can be easily derived from an arbitrary (NC-)manufacturing process. (E.g. by recording the position of each axis involved as a function of time.)

Parametric surface transformable like a single point

The parametric representation transforms under a roto-translation (rotation R and translation $(\bar{t_x}, \bar{t_y}, \bar{t_z})$) like a single coordinate vector:

$$\begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \\ 1 \end{pmatrix} = \begin{bmatrix} & R & & \bar{t_x} \\ & & & \bar{t_y} \\ & & & \bar{t_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x'(u,v) \\ y'(u,v) \\ z'(u,v) \\ 1 \end{pmatrix} . \tag{1.13}$$

Instead of coordinates x, y, z, coordinate functions $x(u,v), y(u,v), z(u,v)$ are transformed.

Qualification for posit./orient. bestfit

The parametric form seems to be particularly suitable for any position/orientation bestfit due to this characteristic. So rotation/translation parameters can be easily separated from all other parameters.

Standard interfaces for parametric surface-description

Dealing with polynomial surfaces or modern description formats for sculptured surfaces like NURBS (=*N*on-*U*niform *R*ational *B*-*S*plines) [ES89], the surface coefficients of the parametric representation can be directly imported in an easy way by standardized formats (VDA-FS, IGES, STEP), [VDA82], [Org90], [ISO94] from nearly every 3D CAD-system dealing with sculptured surfaces.

Parametric representation important for CAD/CAM/CAQ

So it is not astonishing that there is especially in the CAD/CAM/CAQ world some effort given [DBPK84], [Ker87], [MYL92] towards searching for possibilities to adapt the parametric representation for position/orientation bestfit. This will also be the way we will proceed in the following.

Point-cloud fit as a CAD/CAM-proof method

Until now there have been a lot of different approaches: The most simple way is to fit the cloud of measuring points against the cloud of the corresponding nominal points on the parametric surface [GT92].

Improved point-cloud fit

This method can be improved by projecting the obtained deviations to the surface normals in the nominal points [GT92]. Others [DBPK84] correct the results by searching for the shortest distances to the bestfitted surface.

Bestfit by interpolating measuring points

[Kru86], [PH89] and [vH89] fit b-splines through the measuring points and then interpolate the measured points at the locations of the nominal points by intersecting them with the respective surface normals. Similar to the substitution with an implicit function there is again a complex approximation problem to be solved.

Point-to-point fit alternated with nearest distance-searching

The approaches above do not deliver the true bestfit solution. Thus several papers searching for exact bestfit solutions [Gaw89] mention the possibility of alternating this point-to-point bestfit with a point-to-surface distance searching algorithm. [Gaw89] abandons this possibility due to the immense costs and [Ker87] and [MYL92] try it with a small number of measuring points. Even then [MYL92] reports problems with high costs.

Difficult control over nested optimizations

Proceeding like this, there are a lot of nested non-linear optimization problems to be solved (see numerical example in section 6.3.2.4). If a single one of the individual optimization processes fails, there is a big danger of not arriving at the true result. It will be shown in section 5.3.3 how these methods can be interpreted as simplifications of FUNKE; this happens by neglecting certain parts of the coefficient matrix of the resulting system of equations.

Problems caused by two additional parameters

So in spite of all the apparent advantages mentioned, big problems come along with the parametric representation which are exclusively caused by its two additional parameters u, v to be determined, describing a single point on the surface. These parameters do not appear in the explicit or implicit representation; this means, either we always know the exact location of a surface point (x, y, z) (in the case of an explicit function), or we do not have to care for it (in the case of an implicit function). On the other side, fitting any surface parameters of a parametric function, we have to search for an additional pair of surface coordinates (u, v), for each additional measuring point involved.

Posit./orient. hides in polynomial coefficients

Above we discussed the problem of redundancy in an implicit or parametric function representation, caused by rotation or translation symmetries. If dealing with a polynomial in parametric form, we are concerned with a special type of redundancy: The position/orientation of a polynomial is already incorporated into the geometry-describing parameters, i.e., in its polynomial coefficients. So if we want to bestfit a sculptured surface, the concept of separating position/orientation from geometry, as used by FUNKE, has vital relevance. Otherwise we would not be able to bestfit it without simultaneously changing its shape.

Surface reconstruction

The task of fitting a polynomial or correcting its nominal coefficients according to given measuring points should be distinguished from the task of fitting a polynomial or a sculptured surface (which may be assembled by a large number of

individual polynomials) with respect to position/orientation but letting its nominal shape remain unchanged, i.e. neither modifying its surface coefficients nor changing the relative position/orientation of the individual polynomial patches [Gar91], [SB94], [KS95]. The former task can be performed in many ways: in classical ones as used in most CAD's, or more sophisticated ones [Hen74], [SH76], [BCF92], which lead to a nonlinear problem by varying the (u, v) parameters as well. The latter task can be solved in one exact way only.

Both types of polynomial-fit available

FUNKE can principally be used for both tasks. Used for the latter one it delivers the desired exact solution, used for the former one, it corresponds to the above mentioned, more sophisticated algorithms. Additionally, it allows the purposely freezing/modifying of individual polynomial-coefficients.

Chapter 2

Basics

2.1 Concepts of Coordinate Metrology

2.1.1 Preliminary Remarks

No manual handling means cost-saving and maximum objectivity

A numerically controlled (NC) coordinate measuring machine is something like a robot, not a working one, but an information delivering one. It can be moved to any position given by the (approximate) nominal 3D/2D coordinates and it sends back the exactly measured 3D/2D coordinates. Because a NC-CMM-system carries out measurements and evaluations with a minimum of manual handling, it saves costs (for a high number of pieces) and at the same time it guarantees (supposing that the NC-program is well-designed) a maximum of objectivity because the measuring procedure itself is uninfluenced by the individual operator.

No exact driving path but exact surface points are guaranteed

It is important to notice that the measured coordinates are not guaranteed to refer exactly to the same locations as the nominal ones. This means some spread around the theoretical driving path is still allowed. On the other hand, we know that they represent a 3D/2D point *lying exactly* on the actual surface in the vicinity of the pretended nominal point.

Surface-oriented measuring and data evaluation

As a consequence, we always have to measure surfaces rather than curves or points in CM, the latter to be determined as intersections of surfaces. Furthermore for accurate data evaluation we should never fit points to points, but points to surfaces or vice versa [Gaw89].

Coordinate system given by a sculptured surface

This becomes particularly important in cases where we do not know the exact workpiece coordinate system. Due to the lack of appropriate methods – especially when dealing with complex or sculptured surfaces – the coordinate system is often determined by those surfaces that are easiest to determine, rather than those representing best the specified functionality of the workpiece.

Functionality-based coordinate system

As an example consider a turbine blade. The turbine base is normally given by standard surfaces while the blade is described by a sculptured one. Of course the form of the *blade* is most important for functionality. However the form deviations are mostly expressed in the easy-to-determine coordinate system given by the

turbine base. A much better approach would be to express the blade deviations in its own – optimally determined – coordinate system [Gaw89]. This way, possibly unnecessarily corrections of the form of the blade are avoided. Instead, we take into account the relative position/orientation deviation of the blade by correcting the *base* accordingly.

Normal procedure to get a workpiece coordinate system

In defining a coordinate system in CM according to the standards [ISO81b], the individual surfaces are weighted differently. Consider a coordinate system defined by three planes: With one plane we perform the 'spatial alignment' to determine the 'primary orientation'. With a second plane we perform the 'plane alignment' to determine the 'secondary orientation' while the third plane determines the 'origin' only. So the first plane is weighted most, because it determines 3 degrees of freedom (DOF) (2 rotations an 1 translation), while the second determines 2 DOF (1 rotation and 1 translation) and the third only one DOF (1 translation). From the point of view of the desired workpiece-functionality, this is quite a suggestive procedure in a lot of cases. However, sometimes a coordinate system is desired which should be defined by all surfaces together with equivalent weights. This may occur in manufacturing, for example, if we have to cut out the nominal shape from a moulded workpiece. But this type of measurement problem cannot be solved by standard commercial software.

CMM as a flexible measuring tool

A CMM is a flexible and universal instrument. This is not only true for a NC-driven CMM but also for a manually operated one. Classical dimensional metrology requires for nearly every class of workpieces (or surfaces respectively) a different measuring device. The diameter of a bore is normally measured with a tool other than the distance between two planes; to inspect e.g. threads, we need special devices as well. The variety of workpiece classes measurable by CM is only restricted by the software available (and of course by the actual probing possibilities, e.g. very small or large workpiece dimensions). This further implies the usefulness of developing software dealing with complex surfaces [WBH75], [Wae80], [Wol84], [Sta90], [SB93] and [Nag94]. So we are able to check the precision of cog-wheels or ball screw drives without additional and specialized equipment.

Special evaluation concept of CM

Apart from these obvious advantages, the new possibilities offered by the evaluation concept used in CM are of key importance even if they are not visible at first sight. This concept could be summarized like this:

> The individual metrology parameters are not determined by individual, *direct* measurements, but they are collectively *evaluated* from coordinate data, all recorded in the *same* measuring base.

Only in this way does it become possible to *decouple* position/orientation, dimension, and form deviations from each other. This topic will be further discussed in section 2.1.2.

Form deviations ideally represent random errors

Due to an easy interpretation, the *ideal* case is that only the latter type of deviations include the *random* errors, with unsystematic behavior and are therefore

not further analyzable. The 'systematic form deviation' could be expressed then by additional dimension parameters.

Allowing a more complex substitute feature than nominal one

A member of a more complex class of surfaces should be allowed as a substitute feature, even if the nominal feature is a simple standard surface. This implies that the chosen class of substitute features contains the class of nominal features as a special subclass.

Example: Elliptic cylinder substitutes circular cylinder

As an example, circular cylinders can be interpreted as special types of elliptic cylinders. To express this explicitly, a suitable parameterization must be chosen. This means that, instead of parameterizing the class of elliptic cylinders with the half axes $\mathbf{p} = (a, b)$ as dimension parameters, it is more suitable to include ellipticity ϵ as an explicit parameter by $\mathbf{p} = (r, \epsilon)$:

$$\mathbf{x}(u, v, \mathbf{p}) = \begin{pmatrix} r \cdot \cos(u) \\ \epsilon \cdot r \cdot \sin(u) \\ v \end{pmatrix} .$$

By keeping the parameter ϵ fixed on the value 1 we allow only the (sub-)class of nominal surfaces for bestfit; by varying ϵ we can additionally allow the more general (upper-)class of elliptic cylinders as substitute surfaces and so record a systematic form deviation (ellipticity) by *one single* parameter.

Appropriate SW desired

Therefore if we are interested in an easy interpretation of the measuring results and if we want use them not only for tolerance control but also for improving the manufacturing process (by eliminating systematic error influences), we should be able to fit complex features (here e.g. an elliptic cylinder), not only standard ones. This requires appropriate software, even when dealing with *standard* nominal features.

Problem: Determining conicity of a cylindric bore

Putting systematic deviations into additional dimension parameters is sometimes not evident at first sight. For example it appears in the well-known method of fitting a bore (representing nominally a circular cylinder) to a cone to determine its conicity. Even if only standard surfaces are involved here, this can cause problems in several commercial SW packages [Dri89]. These problems are related to the fact that the top of the cone goes to infinity (in the case the cone is nearly a cylinder).

Solution by appropriate parametric description

Fitting by using parametric representation, we can overcome this problem by replacing the normal parameterization of the cone with half-angle $(\gamma) = \mathbf{p}$ as dimension parameter

$$\mathbf{x}(u, v, \mathbf{p}) = \begin{pmatrix} v \cdot \cos(u) \\ v \cdot \sin(u) \\ v \cdot \cot(\gamma) \end{pmatrix} ,$$

by a parameterization with dimension parameters $\mathbf{p} = (\gamma, r)$

$$\mathbf{x}(u, v, \mathbf{p}) = \begin{pmatrix} (v \cdot \sin(\gamma) + r) \cdot \cos(u) \\ (v \cdot \sin(\gamma) + r) \cdot \sin(u) \\ v \cdot \cos(\gamma) \end{pmatrix} .$$

While the former parameterization includes the subclass of *planes* (with $\gamma = \pi/2$) Either planes or cylinders as subclass

$$\mathbf{x}(u, v, \mathbf{p}) = \begin{pmatrix} v \cdot \cos(u) \\ v \cdot \sin(u) \\ 0 \end{pmatrix} ,$$

the latter parameterization includes the subclass of *cylinders* (with $\gamma = 0$)

$$\mathbf{x}(u, v, \mathbf{p}) = \begin{pmatrix} (0 + r) \cdot \cos(u) \\ (0 + r) \cdot \sin(u) \\ v \cdot 1 \end{pmatrix} .$$

Keeping the position in z (top of the cone) fixed, we arrive at the same number of unknowns. There is therefore no redundancy in the description, which could cause rank-deficiency in the resulting system of equations. Impeding redundancy

2.1.2 Position/Orientation, Dimension, and Form

The main goal of CM is the dimensional analysis of a workpiece with respect to position/orientation, dimension and form deviations of its surfaces. Primary evaluation concept of CM

The tolerances defined by a design engineer [ISO85] are normally prescribing limits for combined and correlated deviations as a sum of these three individual types of deviations. Often this is not suitable for the intended functionality of the workpiece [Wir88], [GMPW90], [Reg90], [Wir93] and it is especially impossible to adjust a manufacturing process based on the error-feedback of combined deviations only. Totalized deviation and tolerances

For example, a position/orientation deviation is often a virtual deviation only, and this is explained by the trivial fact that rotating and translating a surface does not change its geometry; only compared relatively to the position/orientation of another surface does it represent a true deviation. Posit./orient. do not get any stand-alone deviation

However, even considering the surface on its own, position/orientation has to be determined first, just to free the exact dimension and form deviations from any other deviations. Afterwards the measured position/orientation can be neglected assuming we have no reference position/orientation in the form of a workpiece coordinate system given by some other surfaces. Posit./orient. has to be determined first to get correct dimension and form deviations

So – considered separately – this type of deviation does not exist in reality, it was created only by the chosen measuring arrangement. It is all the more important to remove the amount of its influence from the totally measured deviation. Settling this by software, the results become independent of the actually chosen measuring base. Deviation due to actual measurement coordinate system

Thus it is a good idea to separate the geometrical description of a surface completely from its position/orientation description. Especially by using the parametric representation, this can be performed in a much more transparent way than with the implicit representation. Posit./orient. separated from geometrical surface description

Any direct measurement delivers a totalized deviation

It is easy to verify that any kind of measurement which aims to determine an individual parameter directly, actually determines such a combined deviation. Only from the evaluation side does it become possible to analyze these different types of deviations and to free them from any correlations between each other.

E.g. multi-point measuring technique

Example: Measuring a shaft at n locations, is a measurement where the dimension parameter 'radius' of several cylinders should be determined directly (Figure 2.1).

2 point measurement

In principle, the radius of an individual cylinder could be determined by a two point measurement somewhere on its circumference. But this way the measured dimension would be influenced (among others) by the form deviation of the cylinder.

2 possibilities to get elliptic cross-sections

Multiple two point measurements along the dashed lines would yield different cylinder radii at different rotational angles. By averaging, the accuracy of the evaluated dimension can be improved, but still we *cannot decide* on the cylinder's form deviation. For example, if the cross-sections along the dashed lines turn out to be elliptic [Hir88], we do not know if the cylinder really has an (elliptic) form deviation or if it has a correct dimension and there is some orientation deviation.

3D-Bestfit provides complete deviation-analysis

Qualitatively we could decide this (in this simple case only) by correctly interpreting the correlation between the deviations measured by probe 3 and the deviations measured by probe 4. In general, the correct *quantitative* interpretation becomes possible only by bestfitting a cylinder through 3D-points which are all taken in the same measuring base. ([SW80], [Buc89])

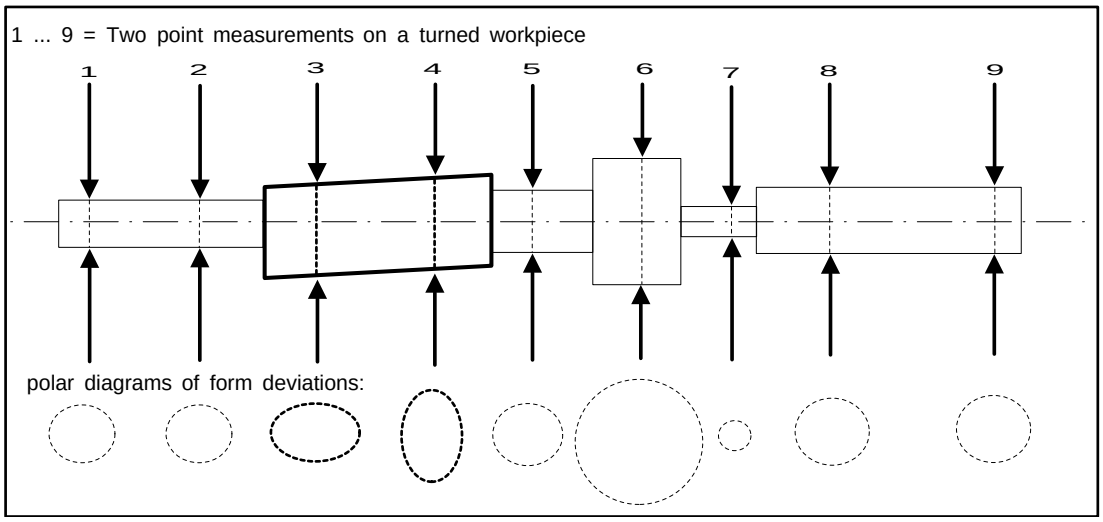


Figure 2.1: Ambiguity (form ↔ pos./orient.) of a two point measurement

2.2 State of the Art of Commercial CM-Software

2.2.1 Problems Solved

CM of standard surfaces

For the CM of standard surfaces, software modules have been developed during

the last 15-20 years (Leitz/DEA/B&S/TESA, Zeiss, SIP, Metromec etc.) which perform this job in an accurately defined way, starting with a given set of measuring points (3D coordinates), a given class of substitute features (to be taken into consideration), and finally a given optimization criterion.

Comparable results

Even if there are some differences between different commercial software products with respect to user interface, treatable constraints and visualization of calculated results, the basic job still remains the same. Results delivered by software 'A' should agree with those delivered by software 'B', supposing that they are dealing with exactly the same input data, consisting of the 3 parts mentioned above [Dri89]. Recently there have been some efforts afoot to develop reference softwares, test data sets, and check-software tools [CF92], [Hop93], [DH93], [Kna93], which check and guarantee this.

2.2.2 Some Unsolved Problems

Today complex and sculptured surfaces do not dispose of comparable, exact bestfit procedures

So it can be observed, that, though on one hand the simple bestfit problem for standard surfaces is solved to a large extent, on the other hand no comparable methods exist for complex or sculptured surfaces. This is an important reason why today's standards (e.g. standards for cog wheels [ISO81a]) do not meet the theoretically exact demands as well.

Exact bestfits of any standard surface-compounds are not state of the art too

Furthermore, besides some special applications, there are no commonly known methods for any kind of simultaneous fit of surface compounds into a given cloud of measuring points, even if exclusively dealing with standard surfaces. As an example we can take the bestfit of a square block (as a compound of 6 planes including angles of 90 degrees) with respect to position, orientation, height, width and depth.

Different procedures for different CM-problems

Finally for different types of standard surfaces as well as for an individual surface fitted with respect to different constraints, different data evaluating algorithms are also necessary. This fact is increasing both, the costs for developing CM software and the costs for validating commercial CM software by implementing reference software for test purposes. Of course it increases also the likelihood of implementation errors.

This thesis as a contribution to these problems

For these reasons a standardization and generalization of bestfit procedure, as a main goal of this thesis, seems to be of great benefit [Lot83]. The method explained herein is intended to contribute to all of the above mentioned types of CM-software problems. In this context, appropriate algorithms have been developed, implemented and verified.

Chapter 3

Definitions

3.1 Standardized Interface

3.1.1 Special Parametric Representation

Same parametric interface for any surface type

The first step towards deriving a function-independent bestfit method is the definition of an appropriate interface between the generally applicable bestfit procedure and the individual surface-functions, which have to be additionally implemented. Every surface in the 3D-space, no matter if it is a regular, complex, or polynomial surface, can be described as follows:

$$\mathbf{x}(u, v, \mathbf{P}) = \begin{pmatrix} x(u, v, \mathbf{P}) \\ y(u, v, \mathbf{P}) \\ z(u, v, \mathbf{P}) \end{pmatrix} . \tag{3.1}$$

Surfaces coordinates u and v

A surface is always described by two independent parameters (a curve only by one), here called u and v, which define the location of a specific point on the surface. In the following they are referred to as surface coordinates. There are other parameters $\mathbf{P}$ which stay fixed for a given surface.

Example: Plane in parametric form

For example, we can express a plane in parametric form as:

$$\mathbf{x}(u, v, \mathbf{P}) = \mathbf{r} + u \cdot \mathbf{n_u} + v \cdot \mathbf{n_v} . \tag{3.2}$$

Fix-parameters

Besides the surface coordinates u, v, we have in total 9 remaining fixed parameters $\mathbf{P} = (\mathbf{r}, \mathbf{n_u}, \mathbf{n_v})$, i.e., the components of the position vector $\mathbf{r}$ and of the two direction vectors $\mathbf{n_u}$ and $\mathbf{n_v}$.

Description for a certain coordinate system

Notice that this parametric description is not independent of the chosen coordinate system: it looks different in every coordinate system. So these parameters describe the position and orientation of the plane for a certain coordinate system only.

Same surface described differently

But for the same coordinate system and the same position and orientation of the plane there also exist an infinite number of possible descriptions.

Various position and directions vectors are allowed

The description above is highly redundant: The position vector $\mathbf{r}$ can be chosen, pointing to any arbitrary point on the plane. The two direction vectors $\mathbf{n_u}$ and $\mathbf{n_v}$ can be varied by their length, by the angle enclosed and by a rotation around the normal direction, still describing the same plane.

Searching for the simplest description

We choose these vectors and a coordinate system in such a way that the parametric description turns out as simple as possible.

Suitable coordinate system

The most suitable coordinate system is a principal axes system. If we define a coordinate system with the plane to be described coincident with the xy-plane, we can choose the position vector

$$\mathbf{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

and the direction vectors

$$\mathbf{n_u} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad \mathbf{n_v} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} .$$

Hence the resulting parametric description is very simple:

$$\mathbf{x}(u, v) = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix} . \tag{3.3}$$

Fixed parameters of the plane do not contain any geometry information

All the 9 additional fixed parameters $\mathbf{P}$ have disappeared. We recognize that they contain only information about position/orientation but not about geometry. In contrast to most other surfaces and curves, a plane can not change its geometry. The class of all planes contains only one geometrical 'instance' because planes differ only by their position and orientation. So for the plane we do not need any fixed parameters describing its geometry.

Geometry parameters

In general we can express the dimension (e.g. cylinder radius) or, generally speaking, the actual geometry (e.g. coefficients of Bézier polynomials) of a surface by g geometric parameters (Not to be confused with the fixed parameters $\mathbf{P}$!)

$$\mathbf{p} = (p_1, \ldots, p_g)$$

where g can vary between 0 (as in the case of a plane) and a very large number (in the case of a high-degree polynomial).

Separated geometry parameters

For the method described we should strictly separate geometrical parameters from position/orientation parameters right from the beginning.

Parameters not distinguished by classical methods

This is not usual for classical evaluation methods. For example, fitting a circle by varying its center (m_x, m_y) and radius r leads to an objective function

$$Q_2(m_x, m_y, r) = \min ,$$

with three unknowns $\mathbf{P} = (m_x, m_y, r)$ treated equally. But only the unknown r describes the geometry, while the coordinates of the center (m_x, m_y) describe only the position and may be eliminated by moving to an appropriate coordinate system. So $\mathbf{p}$, defined in the sense above, consists of only one element $\mathbf{p} = (r)$.

Avoiding redundancies

Therefore we are specializing the purely geometrical description of a surface to a description in a principal axes system. By doing so we keep the parametric description as simple as possible, even if it is not absolutely necessary for the described algorithm. In this way we are able to avoid redundancies as much as possible and to separate geometry from position/orientation as far as possible.

Polynomial: Geometry and posit./ orient. cannot be decoupled

In fact, in some cases (polynomial surfaces), the parameters simultaneously describing geometry and position/orientation cannot be separated. We will discuss this problem in section 5.2.2.

SPR: Special parametric description

We call this simplified, purely geometrical description, preferably in a principal axes system, the *special parametric representation* (SPR):

$$\mathbf{x}'(u, v, \mathbf{p}) \ .$$

For the description of position/orientation we will always refer to this special parametric representation $\mathbf{x}'$, containing only geometrical but no general position/orientation information, as given, e.g., by (3.3), where $\mathbf{x}'(u, v, \mathbf{p}) = (u, v, 0)$ describes a plane in SPR.

3.1.2 Generalized Parametric Representation

Posit./ orient. parameters

By means of 3 rotations around the x-,y- and z-axis defined by the angles

$$\mathbf{a} = (a, b, c)$$

and a translation vector

$$\mathbf{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \ ,$$

we can move a surface from SPR into any desired position/orientation.

Parametric description for any posit./orient.

Let $\mathbf{x}'$ be the coordinates in the principal axes system. Then

$$\mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t}) = R(\mathbf{a}) \cdot \mathbf{x}'(u, v, \mathbf{p}) + \bar{\mathbf{t}} = R(\mathbf{a}) \cdot (\mathbf{x}'(u, v, \mathbf{p}) + \mathbf{t}) \tag{3.4}$$

with

$$\mathbf{t} = R(\mathbf{a})^T \cdot \bar{\mathbf{t}}$$

Total rotation as combined xyz-rotations

and

$$R(\mathbf{a}) = R_x(a) \cdot R_y(b) \cdot R_z(c) \tag{3.5}$$

where

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & \sin(a) \\ 0 & -\sin(a) & \cos(a) \end{bmatrix} \tag{3.6}$$

$$R_y = \begin{bmatrix} \cos(b) & 0 & -\sin(b) \\ 0 & 1 & 0 \\ \sin(b) & 0 & \cos(b) \end{bmatrix} \tag{3.7}$$

$$R_z = \begin{bmatrix} \cos(c) & \sin(c) & 0 \\ -\sin(c) & \cos(c) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{3.8}$$

Rotation and translation of the plane

Not only single points, but also the surface-function can be transformed as a whole. As an example we transform the plane (3.3) by using (3.4)

$$R(\mathbf{a}) \cdot \left(\begin{pmatrix} u \\ v \\ 0 \end{pmatrix} + \mathbf{t} \right)$$

and we arrive at the usual description (3.2)

$$\mathbf{x}(u, v, \mathbf{P}) = \mathbf{r} + u \cdot \mathbf{n_u} + v \cdot \mathbf{n_v}$$

with

$$\mathbf{r} = R(\mathbf{a}) \cdot \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

and

$$\mathbf{n_u} = \begin{pmatrix} \cos(b)\cos(c) \\ \cos(c)\sin(a)\sin(b) - \cos(a)\sin(c) \\ \cos(a)\cos(c)\sin(b) + \sin(a)\sin(c) \end{pmatrix}$$

and

$$\mathbf{n_v} = \begin{pmatrix} \cos(b)\sin(c) \\ \cos(a)\cos(c) + \sin(a)\sin(b)\sin(c) \\ \cos(a)\sin(b)\sin(c) - \cos(c)\sin(a) \end{pmatrix} .$$

Direction vectors have automatically length 1

Notice: Defining the direction vectors $\mathbf{n_u}$ and $\mathbf{n_v}$ like this, they are automatically unit vectors and orthogonal to each other

$$\| \mathbf{n_u} \| = 1 \quad \text{and} \quad \| \mathbf{n_v} \| = 1 \quad \text{and} \quad \| \mathbf{n_u} \|^T \cdot \| \mathbf{n_v} \| = 0$$

for arbitrary values (a, b, c), because $R(\mathbf{a})$ is orthogonal.

Symmetries: Certain posit./orient. parameters can be eliminated

Furthermore, due to the 2 translation symmetries of the plane, we can prescribe the translations t_x, t_y, and due to the rotation-symmetry, we can also prescribe for c arbitrary constant values, for example

$$t_x = 0 \qquad t_y = 0 \qquad c = 0 .$$

Position and direction vectors expressed by $\mathbf{t}$ and $\mathbf{a}$

By this we have eliminated all redundancies. We get

$$\mathbf{r}(a,b,t_z) = \begin{pmatrix} -\sin(b)\cdot t_z \\ \sin(a)\cos(b)\cdot t_z \\ \cos(a)\cos(b)\cdot t_z \end{pmatrix}$$

and

$$\mathbf{n_u}(a,b) = \begin{pmatrix} \cos(b) \\ \sin(a)\sin(b) \\ \cos(a)\sin(b) \end{pmatrix} \qquad \mathbf{n_v}(a) = \begin{pmatrix} 0 \\ \cos(a) \\ \sin(a) \end{pmatrix} .$$

So we have explicitly expressed the plane by the position/orientation parameters defined above:

$$\mathbf{x}(u,v,\mathbf{P}) = \mathbf{x}(u,v,\mathbf{r}(a,b,t_z),\mathbf{n_u}(a,b),\mathbf{n_v}(a)) = \mathbf{x}(u,v,a,b,t_z) .$$

Not taking in account any symmetries: g+6 unknown parameters

In general we have to define, according to (3.4), the surface $\mathbf{x}$ as a function of 6 position/orientation parameters $\mathbf{a} = (a,b,c)$ and $\mathbf{t} = (t_x,t_y,t_z)$. Fitting the surface also with respect to its dimension (or, more generally, to its geometry) we have to vary additionally the g geometry parameters $\mathbf{p} = (p_1,\ldots,p_g)$. So not taking into account any symmetries, a general surface description $\mathbf{x}(u,v)$ has $g+6$ fixed parameters

$$\mathbf{x} = \mathbf{f}(u,v,\underbrace{p_1,\ldots,p_g,a,b,c,t_x,t_y,t_z}_{fixed\ parameters}) . \tag{3.9}$$

3.2 Nominal Feature and Nominal Points

Nominal geometry

Let the nominal surface geometry be defined by a set of g initial values

$$\mathbf{p_0} = (p_{1_0},\ldots,p_{g_0})$$

of its dimension parameters. Then the nominal surface in SPR $\mathbf{x'}_{nom}(u,v)$ becomes

$$\mathbf{x'}_{nom}(u,v) = \mathbf{x'}(u,v,\mathbf{p_0}) . \tag{3.10}$$

Nominal posit./orient.

According to (3.4) the nominal position/orientation of the nominal surface is given by the 3 initial values

$$\mathbf{t_0} = (t_{x0},t_{y0},t_{z0})$$

for the translation vector and the 3 initial values

$$\mathbf{a_0} = (a_0,b_0,c_0)$$

for the angles by which we achieve the nominal position/orientation, starting from the original position/orientation of the surface description $\mathbf{x'}(u,v,\mathbf{p})$ in SPR.

Inserting these initial values into (3.9), the nominal surface $\mathbf{x}_{nom}(u,v)$ in nominal position/orientation is completely defined

Nominal surface in nominal posit./orient.

$$\mathbf{x}_{nom}(u,v) = \mathbf{x}(u,v,\mathbf{p_0},\mathbf{a_0},\mathbf{t_0}) \ . \tag{3.11}$$

After defining nominal geometry and nominal position/orientation, we can specify the location of the i-th nominal-point $\tilde{\mathbf{x}}_\mathbf{i}$ on the surface by the two surface coordinates

Nominal points

$$(u_{i0}, v_{i0})$$

and we get its 3D-coordinates

$$\tilde{\mathbf{x}}_\mathbf{i} = \mathbf{x}(u_{i0},v_{i0},\mathbf{p_0},\mathbf{a_0},\mathbf{t_0}) \ . \tag{3.12}$$

All these nominal values we know from the beginning. We can use them as initial values for an iterative search for the final, unknown optimum as the result of the bestfit.

Nominal data as starting point for optimization

Apart from these nominal data, we have the measuring points as an additional input for the bestfit procedure. For each nominal point $\tilde{\mathbf{x}}_\mathbf{i}$, we define the measuring point belonging to it as

Measuring points as additional bestfit-input

$$\hat{\mathbf{x}}_\mathbf{i} = (\hat{x}_i, \hat{y}_i, \hat{z}_i) \ .$$

3.3 Substitute Feature and Footpoints

Let $\mathbf{p}^*$ be the unknown values of the parameters $\mathbf{p}$ describing the geometry of the substitute feature (= member of the allowed class of features, which is best substituting the real feature).

Geometry of the substitute feature

Furthermore, let $\mathbf{a}^*, \mathbf{t}^*$ be the values of the parameters $\mathbf{a}, \mathbf{t}$ describing position/orientation of the bestfitted substitute feature.

Posit./orient. after bestfit

All these parameters appear also in an *implicit* representation even though they are normally not explicitly available but hidden in a non-trivial way (e.g. they have to be determined by a principal axes transformation).

In implicit representation the different types of parameters cannot be separated

The *parametric* substitute surface with unknown geometry $\mathbf{p}^*$ is described in its unknown, bestfitted position/orientation $\mathbf{a}^*, \mathbf{t}^*$ with the two additional parameters u, v by:

Substitute surface in bestfitted pos./orient.

$$\mathbf{x}_{sub}(u,v) = \mathbf{x}(u,v,\mathbf{p}^*,\mathbf{a}^*,\mathbf{t}^*) \ . \tag{3.13}$$

With these definitions we can define the bestfit as the task to find the corrections

Corrections to the nominal values

$$\begin{aligned} \mathbf{\Delta p} &= \mathbf{p}^* - \mathbf{p_0} \\ \mathbf{\Delta a} &= \mathbf{a}^* - \mathbf{a_0} \\ \mathbf{\Delta t} &= \mathbf{t}^* - \mathbf{t_0} \end{aligned}$$

to the known nominal values $\mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0}$ such that the obtained surface fits optimally into the given measuring points $\hat{\mathbf{x}_1} \ldots \hat{\mathbf{x}_n}$. The optimum is defined then by a given objective function (Gauss, Chebychev, etc.).

Footpoints

Dealing with parametric representation we can also specify the location of the i-th footpoint $\bar{\mathbf{x}}_\mathbf{i}$ (defined as the surface-point with shortest distance to the respective measuring point) after bestfit belonging to the i-th measuring point $\hat{\mathbf{x}}_\mathbf{i}$ by two surface coordinates

$$(u_i{}^*, v_i{}^*) \ .$$

We get its 3D-coordinates by

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{x}(u_i{}^*, v_i{}^*, \mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*) \ . \tag{3.14}$$

Corrections to the nominal surface coordinates

Thus we can analogously define

$$\begin{aligned} \boldsymbol{\Delta}\mathbf{u}_i &= u_i{}^* - u_{i0} \\ \boldsymbol{\Delta}\mathbf{v}_i &= v_i{}^* - v_{i0} \ . \end{aligned}$$

Small geometrical corrections $\boldsymbol{\Delta}\mathbf{p}$

We know from CM practice that the corrections Δp, representing the geometrical differences between substitute and nominal features, are normally very small. Either the *absolute* differences $\| \boldsymbol{\Delta}\mathbf{p_i} \|$ are small if the $\| \mathbf{p_{0i}} \|$ themselves are small, or if they are large, the *relative* differences are typically smaller than 0.1% to 1%, i.e.:

$$\frac{\| \boldsymbol{\Delta}\mathbf{p_i} \|}{\| \mathbf{p_{0i}} \|} \ \leq \ 0.001 \quad \text{to} \quad 0.01 \ .$$

A reasonable manufacturing process should not deliver parts with larger deviations.

Small posit./orient. corrections $\boldsymbol{\Delta}\mathbf{t}, \boldsymbol{\Delta}\mathbf{a}$

While measuring on a numerically controlled CMM, we know furthermore that the distances which are allowed for the real workpiece surfaces to differ from the defined nominal points typically have a magnitude of not more than $0.5mm$ to $1mm$, i.e.:

$$\| \hat{\mathbf{x}}_\mathbf{i} - \tilde{\mathbf{x}}_\mathbf{i} \| \ \leq \ 0.5mm \quad \text{to} \quad 1mm \ .$$

Otherwise collisions or large security spaces for probing could not be avoided. By this, we know that we are dealing also with small values $\boldsymbol{\Delta}\mathbf{a}$ and $\boldsymbol{\Delta}\mathbf{t}$.

Rough posit./orient. for measurement on a CNC-driven CMM

So we have always to determine first roughly an approximate position/orientation of a workpiece lying on the CMM and to define this position/orientation as the nominal one. Based on this information the CMM can be moved automatically to the individual nominal points.

Linearization around the nominal data

We have to perform small corrections to given nominal parameters. Thus, we can express the unknown values $\mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*$ by the known values $\mathbf{p_0}, \mathbf{t_0}, \mathbf{a_0}$ based on a linearization around the latter ones. As a prerequisite we need the functional dependence which this linearization is based on.

Explicit linear approximation for the bestfitted parametric surface

In the following, the explicit expression for the linearly approximated values $\mathbf{p}^{*\prime}, \mathbf{a}^{*\prime}$ and $\mathbf{t}^{*\prime}$ can be given if a differentiable parametric function in form (3.4) has to be bestfitted with regard to Gaussian (L_2) norm.

Linearization in a function independent way

It is the goal of this thesis to perform this in a function-*independent* way for any given $\mathbf{x}'(u, v, \mathbf{p})$. As shown in section 6.1.2, we are able to do this in an explicit way, even when we are taking into account any probe radius corrections.

The surface coordinates (u_{i0}, v_{i0}) of the nominal points are also involved in this linearized expression, i.e., we have to compute $\mathbf{p}^{*\prime}$, $\mathbf{a}^{*\prime}$ and $\mathbf{t}^{*\prime}$ based on all of the following values

Surface coordinates u, v additionally involved

$$\mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0}, u_{10}, v_{10} \ldots u_{n0}, v_{n0}, \hat{\mathbf{x_1}} \ldots \hat{\mathbf{x_n}} \ .$$

Because $\mathbf{\Delta a}$ and $\mathbf{\Delta t}$ are small, the surface coordinates of the i-th footpoint $({u_i}^*, {v_i}^*)$ are also very close to the surfaces coordinates of the i-th nominal point (u_{i0}, v_{i0}), assuming we are measuring with a CNC-controlled CMM.

CNC-CMM: Small corrections to the surface coordinates

Thus the linearization point is located close to the final solution and this guarantees a good approximation (and a good convergence of the iterated search). Of course, in CM-practice we should not be satisfied with such a linearized solution, even the results of a linearized, *surface*-based model are still superior to those of any *point*wise fit (which are common practice for sculptured surfaces today).

Linear approximation still better than a point wise fit

However, by iteratively solving the linearized model (Gauss-Newton method) we can improve this first approximation. So we get the true bestfit even if the starting values are not so close to the final solution.

Linearization extended to Gauss-Newton method

This could appear, e.g., if we are measuring with a manually operated CMM where we are not able to move exactly to given nominal points. In this case we do not dispose of good approximated starting values u_{i_0}, v_{i_0}.

Evaluating data of a manually operated CMM

Chapter 4

Function Independent Bestfit

4.1 Preliminary Remarks

Parametric representation convenient for function independent bestfit

In the preceding chapters, we have shown that parametric representation is convenient for function independent bestfit. First, we can define a standardized interface for all types of parametrically defined surfaces. Second, we can properly separate geometry from position/orientation based on this representation form.

Missing distance function

On the other hand, with parametric representation, it is not possible to fit the surface as a whole to the measuring points in the same way as we can do with implicit representation. First, we have 3 equations instead of 1, second, we have the two surface coordinates u and v as additional unknown parameters. So we cannot define an explicit distance function which gives the actual distance to the surface by inserting any measuring point into it.

Point-to-point fit is also surface-independent

So the most simple way to bestfit independently of the actual surface function is to use a point-to-point fit (see e.g. [RMM94]): Each point in the cloud of measured points has a corresponding point in the cloud of nominal points. The first cloud has to be fitted into the second cloud obeying a given optimization criterion (Gauss, Chebychev, etc.). Most of the SW-tools commercially available today for sculptured surface measurement are using such types of algorithms.

No satisfactory results with pointwise fit

But a pointwise fit cannot give satisfactory results, for example, it delivers results strongly dependent on the arbitrary choice of the measuring base. This is in absolute contradiction to the basic concepts of CM, as explained in section 2.1.

Point-to-point fit is covered by FUNKE as a special case

FUNKE is a surface-oriented algorithm. Because there is an implemented mechanism to 'freeze' (i.e. to eliminate) all unknowns individually, point-to-point fit is contained in FUNKE as a special case. We can achieve this by freezing the surface coordinates u_{0i}, v_{0i} of the nominal points $\tilde{\mathbf{x}}_\mathbf{i}$. By this we are able to simulate the results of a point-to-point fit [SB92], [SB95].

Point-to-point fit with SVD

Normally point-to-point fit is an iterative searching process as well as a surface-oriented fit. For starting values far away from the final solution convergence is not guaranteed. In the following section 4.2 we present a (as far as we know) new method to fit a cloud of points into another cloud of points in the least squares

sense. By applying singular value decomposition (SVD) to this well-known problem, we avoid any convergence problems which could appear by conventionally searching for optimum (e.g. using Newton's method).

4.2 Pointwise Bestfit

Expression for the distances in bestfitted state

For fitting the cloud of nominal points to the cloud of measuring points, we have first to express the distance between the i-th measuring point $\hat{\mathbf{x}}_\mathbf{i}$ belonging to the i-th nominal point $\tilde{\mathbf{x}}_\mathbf{i}$ as a function of the translational and rotational corrections $\mathbf{\Delta t}$ and $\mathbf{\Delta a}$ which will perform the point-to-point bestfit:

$$d_i(\mathbf{\Delta a}, \mathbf{\Delta t}) = \parallel \hat{\mathbf{x}}_\mathbf{i} - R(\mathbf{\Delta a}) \cdot \tilde{\mathbf{x}}_\mathbf{i} - \mathbf{\Delta t} \parallel . \tag{4.1}$$

Objective function

Then we minimize the resulting objective function in dependence of these corrections $\mathbf{\Delta a}$ and $\mathbf{\Delta t}$

$$\min_{\mathbf{\Delta a}, \mathbf{\Delta t}} Q_2(\mathbf{\Delta a}, \mathbf{\Delta t}) = \min_{\mathbf{\Delta a}, \mathbf{\Delta t}} \sum_{i=1}^{n} d_i(\mathbf{\Delta a}, \mathbf{\Delta t})^2 . \tag{4.2}$$

Separation in 2 sub-problems

We separate the problem into two sub-problems; in a least squares problem as a function of *translation* and in a least squares problem as a function of *rotation*:

$$\min_{\mathbf{\Delta a}, \mathbf{\Delta t}} Q_2(\mathbf{\Delta a}, \mathbf{\Delta t}) = \min_{\mathbf{\Delta a}} \left(\underbrace{\min_{\mathbf{\Delta t(\Delta a)}} Q_2\left(\mathbf{\Delta a}, \mathbf{\Delta t}(\mathbf{\Delta a})\right)}_{translational\ \ problem} \right) = \underbrace{\min_{\mathbf{\Delta a}} Q_2^\star(\mathbf{\Delta a})}_{rotational\ \ problem} . \tag{4.3}$$

Known solutions for the 2 sub-problems

These two sub-problems can be solved by well-known methods. The translational problem is solved by merging the two *centers of gravity* of the two point clouds and the rotational problem is the so-called orthogonal *Procrustes problem* [GvL89].

Translational problem

First we solve the translational problem:

$$\min_{\mathbf{\Delta t(\Delta a)}} Q_2(\mathbf{\Delta a}, \mathbf{\Delta t}(\mathbf{\Delta a})) = \sum_{i=1}^{n} \parallel \hat{\mathbf{x}}_\mathbf{i} - R(\mathbf{\Delta a}) \cdot \tilde{\mathbf{x}} - \mathbf{\Delta t} \parallel^2 . \tag{4.4}$$

Least squares problem

This leads to the overdetermined system

$$\begin{bmatrix} I_3 \\ \vdots \\ I_3 \end{bmatrix} \cdot \mathbf{\Delta t} \approx \begin{pmatrix} \hat{\mathbf{x}}_\mathbf{1} - R(\mathbf{\Delta a}) \cdot \tilde{\mathbf{x}}_\mathbf{1} \\ \vdots \\ \hat{\mathbf{x}}_\mathbf{n} - R(\mathbf{\Delta a}) \cdot \tilde{\mathbf{x}}_\mathbf{n} \end{pmatrix} . \tag{4.5}$$

Normal equations

The normal equations belonging to the above system

$$n \cdot I_3 \cdot \mathbf{\Delta t} = \sum_{i=1}^{n} \hat{\mathbf{x}}_\mathbf{i} - R(\mathbf{\Delta a}) \cdot \tilde{\mathbf{x}}_\mathbf{i}$$

Difference-vector between centers of gravity

can be solved by the difference vector between the centers of gravity of measuring points and rotated nominal points:

$$\mathbf{\Delta t_{min}}(\mathbf{\Delta a}) = \frac{\sum_{i=1}^{n} \hat{\mathbf{x}}_\mathbf{i}}{n} - R(\mathbf{\Delta a}) \cdot \frac{\sum_{i=1}^{n} \tilde{\mathbf{x}}_\mathbf{i}}{n}$$

$$= < \hat{\mathbf{x}} > - R(\mathbf{\Delta a}) < \tilde{\mathbf{x}} > \ .$$

Rotational problem

By inserting this expression into the objective function Q_2 according to (4.3), we obtain the sub-problem for rotation

$$\min_{\mathbf{\Delta a}} Q_2 \left(\mathbf{\Delta a}, \mathbf{\Delta t_{min}}(\mathbf{\Delta a})\right) = \min_{\mathbf{\Delta a}} Q_2^\star(\mathbf{\Delta a})$$

$$= \sum_{i=1}^{n} \| \ (\hat{\mathbf{x}}_\mathbf{i} - < \hat{\mathbf{x}} >) - R(\mathbf{\Delta a}) \cdot (\tilde{\mathbf{x}}_\mathbf{i} - < \tilde{\mathbf{x}} >) \ \|^2 \ .$$

Orthogonal Procrustes problem

It is equivalent with the orthogonal Procrustes problem

$$\| \ A - BQ \ \|_F = \| \ A^T - Q^T B^T \ \|_F = \min \qquad \text{subject to} \qquad Q^T Q = I \ . \tag{4.6}$$

Transformed measuring points

Identifying A^T by the matrix formed by all *measuring* point coordinates transformed into its center of gravity system yields

$$\begin{bmatrix} (\hat{\mathbf{x}}_\mathbf{1} - < \hat{\mathbf{x}} >)^T \\ \vdots \\ (\hat{\mathbf{x}}_\mathbf{n} - < \hat{\mathbf{x}} >)^T \end{bmatrix} ,$$

Transformed nominal points

B^T by the matrix formed by all *nominal* point coordinates transformed into its center of gravity system yields

$$\begin{bmatrix} (\tilde{\mathbf{x}}_\mathbf{1} - < \tilde{\mathbf{x}} >)^T \\ \vdots \\ (\tilde{\mathbf{x}}_\mathbf{n} - < \tilde{\mathbf{x}} >)^T \end{bmatrix}$$

and Q^T with $R(\mathbf{a})$.

Solution by SVD

In [GvL89] it is shown that this can be solved by putting

$$Q = UV^T,$$

where U and V are the matrices obtained by the singular value decomposition

$$B^T A = U \Sigma V^T.$$

Polar decomposition

Q is also the orthogonal factor of the polar decomposition of the matrix $B^T A$, see [Gan90]:

$$B^T A = U\Sigma V^T = \underbrace{(UV^T)}_{orthogonal} \cdot \underbrace{(V\Sigma V^T)}_{pos.semidefinite} = \underbrace{Q \cdot P}_{polar\ decomposition} ,$$

where Q is orthogonal and P is semi-definite. The polar decomposition can be interpreted as a generalization of the polar representation of a complex value.

So wet get the following algorithm:

SVD-based point-to-point-fit algorithm

1. Calculate centers of gravity:

$$< \hat{\mathbf{x}} >= \frac{\sum_{i=1}^{n} \hat{\mathbf{x}}_\mathbf{i}}{n} \qquad < \tilde{\mathbf{x}} >= \frac{\sum_{i=1}^{n} \tilde{\mathbf{x}}_\mathbf{i}}{n}$$

2. Build matrices with the relative coordinates:

$$C = \begin{bmatrix} (\hat{\mathbf{x}_1} - < \hat{\mathbf{x}} >)^T \\ \vdots \\ (\hat{\mathbf{x}_n} - < \hat{\mathbf{x}} >)^T \end{bmatrix}$$

$$D = \begin{bmatrix} (\tilde{\mathbf{x}_1} - < \tilde{\mathbf{x}} >)^T \\ \vdots \\ (\tilde{\mathbf{x}_n} - < \tilde{\mathbf{x}} >)^T \end{bmatrix}$$

3. Perform the singular value decomposition of DC^T:

$$DC^T = U\Sigma V^T$$

4. Calculate rotation matrix $R(\mathbf{\Delta a})$ by:

$$R(\mathbf{\Delta a}) = VU^T$$

5. Calculate translation vector $\mathbf{\Delta t}$ by:

$$\mathbf{\Delta t} =< \hat{\mathbf{x}} > -R(\mathbf{\Delta a}) < \tilde{\mathbf{x}} >$$

We can also obtain the factorized form of the rotation matrix $R(\mathbf{\Delta a})$ according to (3.5). We apply 3 single Givens rotations G_1,G_2 and G_3 to transform the rotation matrix into an identity matrix. Factorizing $R(\mathbf{\Delta a})$

$$G_3 \cdot G_2 \cdot G_1 \cdot R = I_3$$

Thus we get the factorization:

$$R(\mathbf{\Delta a}) = R_x(\Delta a) \cdot R_y(\Delta b) \cdot R_z(\Delta c) = G_3^T \cdot G_2^T \cdot G_1^T \ . \qquad (4.7)$$

4.3 Parametric Function-Based Bestfit Model

4.3.1 'Distance Function' of a Parametric Surface

Point-to point fit compared to point-to-surface fit

Of course such a fit of a point cloud to another point cloud as discussed in the previous section does not lead to the same good results as the classical fit of a surface to a point cloud. It does not allow us to adapt geometry (dimension) parameters and it depends to a high degree on the initial position/orientation.

Problems caused by pointwise description

The special characteristic of the parametric representation to describe a surface pointwise, useful for CAD and for 3-axes control of a manufacturing process, turns out to be a vital drawback here.

Orthogonal surface-distance as a direct function of a 3D coordinate

As discussed in section 1.2.3, in classical CM we are dealing with an implicit surface description $d(\mathbf{x}) = 0$, called *distance function*, which delivers explicitly the orthogonal surface-distance $d_{\hat{\mathbf{x}}}$ of any measuring point $\hat{\mathbf{x}}$ inserted into it:

$$d_{\hat{\mathbf{x}}} = d(\hat{\mathbf{x}}) \ .$$

Only distance to a single surface point available

For parametric surfaces we cannot compute the point-to-surface-distance with an analytical expression. An *analytical* expression we can give only for the distance between a measured point and a well-defined surface point.

(u^M, v^M) belonging specifically to a certain measuring point

So the point-to-surface distance, delivered explicitly by a distance-function based on implicit representation, is not easy to obtain by a pointwise 'distance function'. It can be obtained only by adding a pair of surface coordinates (u^M, v^M) belonging specifically to the actual measuring point.

$$d(\underbrace{\hat{\mathbf{x}}, u^M, v^M}_{MP \ \ input}, \mathbf{a}, \mathbf{t}, \mathbf{p}) = \| \ \hat{\mathbf{x}} - R(\mathbf{a}) \cdot \Big(\mathbf{x}'(u^M, v^M, \mathbf{p}) - \mathbf{t} \Big) \ \| \tag{4.8}$$

Minimizing delivers point-to-surface distance

These coordinates are not known from the beginning. They are the surface coordinates of the corresponding footpoint, i.e., of the surface-point whose surface normal meets exactly the chosen measuring point. However, we know that the function $d(\hat{\mathbf{x}}, \mathbf{a}, \mathbf{t}, \mathbf{p}, u, v)$ is minimal at this location (u^M, v^M). Thus we cannot give an explicit expression for $d(\hat{\mathbf{x}}, \mathbf{a}, \mathbf{t}, \mathbf{p})$ (like for implicitly defined standard surfaces), but we can write in a formally correct way:

$$d_{\hat{\mathbf{x}}} = d(\hat{\mathbf{x}}, \mathbf{a}, \mathbf{t}, \mathbf{p}) = \min_{u,v} \| \ \hat{\mathbf{x}} - \mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t}) \ \| \ \ . \tag{4.9}$$

Objective function

So we would obtain the following objective function:

$$Q_2(\mathbf{p}, \mathbf{a}, \mathbf{t}) = \sum_{i=1}^{n} d^2_{\hat{\mathbf{x}}_i} = \sum_{i=1}^{n} \min_{u,v} \| \ \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t}) \ \|^2 \ . \tag{4.10}$$

Alternated minimization

Obviously Q_2 could now be minimized directly, after having searched for the (shortest) surface-distances $d_{\hat{\mathbf{x}}_i}$ of all measuring points (e.g. proposed by [Ker87], [Gaw89] or especially for polynomial surfaces by [Hos92], [MYL92]).

$$\min_{\mathbf{p},\mathbf{a},\mathbf{t}} Q_2(\mathbf{p}, \mathbf{a}, \mathbf{t}) = \min_{\mathbf{p},\mathbf{a},\mathbf{t}} \sum_{i=1}^{n} d^2_{\hat{\mathbf{x}}_i} = \min_{\mathbf{p},\mathbf{a},\mathbf{t}} \sum_{i=1}^{n} \min_{u,v} \| \ \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t}) \ \|^2 \tag{4.11}$$

Minimization with fix footpoints

The search for orthogonal distances $\min_{u,v} \| \ldots \|$ yields for each measuring point a specific pair of surface coordinates (u_i^M, v_i^M) designating the actual footpoint on the surface. According to (4.11) Q_2 is minimized then as a function of $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$. The footpoints, given by (u_i^M, v_i^M), stay fixed during this minimization process. If position/orientation $\mathbf{a}$ and $\mathbf{t}$ without any dimension parameters $\mathbf{p}$ are fitted, this corresponds to a point-to-point fit which could be performed, e.g., by the algorithm of section 4.2.

'Distance function' changes by each step

However, the two step minimization (4.11) will not deliver the real minimum. The reason is that after searching for the minimum as a function of the parameters $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$, the distances $d_{\hat{\mathbf{x}}_i}$ as determined before are not valid any more for the new values of $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$. (This is in contrast to an implicit distance function, which gives the explicit surface distance of a measuring point for arbitrary values of these parameters.)

Exact surface coordinates depend on all other parameters

The surface coordinates (u_i^M, v_i^M) of the actual footpoint belonging to the i-th measuring point are strongly linked to position/orientation and geometry of the whole surface:

$$u_i^M = u_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t}) \qquad v_i^M = v_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t}) \; . \tag{4.12}$$

Correlation has to be taken into account

Only by taking into account this correlation while evaluating (4.11) in the form

$$\min_{\mathbf{p},\mathbf{a},\mathbf{t}} \sum_{i=1}^{n} \| \; \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}(u_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t}), v_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t}), \mathbf{p}, \mathbf{a}, \mathbf{t}) \; \|^2 \tag{4.13}$$

would we obtain the real minimum.

'Small-Displacement Screw'

The 'Small-Displacement Screw' of [BC76] and [BC88] could be interpreted as an approach to account for this correlation approximately in the case of small angles $\mathbf{a}$ and without varying the geometry parameters $\mathbf{p}$.

Footpoints on the bestfitted surface

The footpoint-coordinates u_i^*, v_i^*, as defined in section 3.3, are the u, v-values yielding shortest distances while inserting the bestfitted surface-parameters $\mathbf{p}^*, \mathbf{a}^*$ and $\mathbf{t}^*$

$$u_i^* = u_i^M(\mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*) \qquad v_i^* = v_i^M(\mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*) \; . \tag{4.14}$$

Multiple nested iteration processes

The crux is now that in general we do not dispose of an explicit expression giving the relation (4.12). So we would have to repeat the normally nonlinear n-times search for the shortest distances between measuring points and the surface and then to minimize $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$ again, etc. (This is analogous to 'Jacobi'-type iteration instead of 'Gauss-Seidel'-type iteration by the iterative solution of a *linear* system of equations: We do not care about the coupling between two parameter sets handled separately.) Here it would lead to multiple nested iteration processes. Each of them should converge to arrive to the correct solution (see numerical example section 6.3.2.4).

4.3.2 Parametric Objective Function by Splitting u and v into 2n Parameters

$2n$ artificially constructed parameters

However, even if we cannot define a real *distance* function which is valid for more than one measuring point and one surface at the same time, we are able to construct a real *objective* function valid for a *set* of measuring points actually taken into consideration. The key step towards achieving this is to split each parameter u and v into n different parameters

$$u_1, \ldots, u_n \qquad \text{and} \qquad v_1, \ldots, v_n.$$

(Note: n different *parameters*, not n different *values* u_i^M of the same parameter u!)

and consequently $\mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t})$ into n 'different' functions,

$$\mathbf{x_1}(u_1, v_1, \mathbf{p}, \mathbf{a}, \mathbf{t}) \ldots \mathbf{x_n}(u_n, v_n, \mathbf{p}, \mathbf{a}, \mathbf{t}) \; .$$

Objective function, valid for whole set of measuring points

By introducing these additional 2n function parameters, we are able to express the minimum of the Q_2 objective function correctly. (This does not correspond to the minimization of (4.10) by varying $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$, which we discussed in the preceding section. Remember: $\min\limits_{a}\left(\min\limits_{b}(\ldots)\right) \neq \min\limits_{a,b}(\ldots)$!) Thus,

$$\min\left(Q_2(u_1, v_1 \ldots u_n, v_n, \mathbf{p}, \mathbf{a}, \mathbf{t})\right) =$$

$$\min_{\mathbf{p},\mathbf{a},\mathbf{t},u_1,v_1 \ldots u_n,v_n} \sum_{i=1}^{n} \parallel \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i}(u_i, v_i, \mathbf{p}, \mathbf{a}, \mathbf{t}) \parallel^2 \tag{4.15}$$

$$\neq \min_{\mathbf{p},\mathbf{a},\mathbf{t}} \sum_{i=1}^{n} \min_{u_i,v_i} \parallel \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i}(u_i, v_i, \mathbf{p}, \mathbf{a}, \mathbf{t}) \parallel^2 \; .$$

Objective function with a multitude of variables

Compared to an objective function based on an implicit distance function, the minimum of this objective function has to be found by varying many more parameters (+2n) than with the other way. Even if we are not interested in the unknown exact values of these 2n additional parameters, we have to minimize the objective function depending on all these artificially constructed variables. By doing so the minimum of $\sum_{i=1}^{n} d_i^2$ as well as the values for $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$ are identical to those delivered by an objective function based on implicit representation.

Objective function equivalent to an implicit-based one

Thus, we have introduced – in addition to the few known Q_2 objective functions based on implicit surface representation for the standard elements – a Q_2 objective function for the parametric surface representation which is *generally applicable* and has always the *same form*, for complex/ sculptured surfaces as well as for standard ones.

Others than Gaussian fit would become too cost-intensive

Obviously we could construct any other objective function Q_m based on an arbitrary norm

$$L_m = \sqrt[m]{\sum_{i=1}^{n} d_i^m} \; ,$$

in the same way. However, this would result in a very complicated system of equations.

Zeroing gradient of the objective function

For minimizing the objective function Q_m its gradient would have to be set equal to zero:

$$\mathbf{grad}\left(Q_m((u_1,v_1)\dots(u_n,v_n),\mathbf{p},\mathbf{a},\mathbf{t})\right)=\sum_{i=1}^{n}\mathbf{grad}\left(d_i^m(u_i,v_i,\mathbf{p},\mathbf{a},\mathbf{t})\right)=\mathbf{0}\ . \tag{4.16}$$

Standard solution costs too much

Theoretically we could apply a standard tool to find the solution of this very large system of nonlinear equations, but this would become a rather cost and memory intensive task. On the other hand, for the case $m = 2$ we can find a solution which becomes quite simple and practical.

Automatized solution-method with no additional costs

Furthermore, for a norm other than L_2, it would not be easy to generate the resulting system of equations for arbitrary surfaces and bestfit problems automatically. But considering the case $m = 2$ (Gaussian least squares) again, it will be shown in the next section 4.4, how this can be performed quite easily in a *function independent* way with nearly no additional costs compared to the conventional, specialized methods. In addition, it can also be used for surfaces not yet predefined, simply by adding the new surface functions.

4.4 L_2-Norm Bestfit

4.4.1 'Gauss-Newton' and Single-Coordinate Residuum

2 advantages for m=2

The 'Gaussian' bestfit ($m = 2$) offers the two following advantages:

Gauss-Newton instead of Newton

First, we can approximate the bestfit problem by a linear least squares problem. Using the 'Gauss-Newton'-method instead of the 'Newton'-method, this allows us to bypass the rather complicated system of equations which results from zeroing the gradient. We construct an overdetermined system of equations which has to be solved in the least squares sense. This system of equations is linearized around the actually known values. At the beginning these are the known nominal values which we can use as a starting point for the iteration process which improves them step by step, converging to the correct least squares solution.

Selecting an other residuum vector

Second, dealing with the L_2-norm, we are able to substitute the original residuum vector consisting of the individual distances between measuring points and surface

$$\mathbf{D}=(d_1,\dots,d_n)$$

by a more appropriate residuum vector.

More suitable residuals thanks to 'Pythagoras'

As we are dealing with orthogonal coordinates, we know that the Pythagoras theorem holds and thus the squares of distances can be expressed by the sum of the squares of the individual coordinate differences

$$d_i^2=\|\ \mathbf{\Delta x_i}\ \|^2=\Delta x_i^2+\Delta y_i^2+\Delta z_i^2\ . \tag{4.17}$$

Thus we have

$$Q_2 = \sum_{i=1}^{n} d_i^2 = \mathbf{\Delta X}^T \mathbf{\Delta X} \tag{4.18}$$

with

$$\mathbf{\Delta X} = (\Delta x_1, \Delta y_1, \Delta z_1, \ldots, \Delta x_n, \Delta y_n, \Delta z_n)$$

representing the vector, which contains the $3n$ coordinate differences $\Delta x, \Delta y, \Delta z$ between the n measuring points

$$\hat{\mathbf{X}} = (\hat{\mathbf{x_1}}, \ldots, \hat{\mathbf{x_n}})$$

and the n points lying on the surface.

Nominal points mutate to final footpoints during bestfit procedure

At the beginning these are the n nominal points of the nominal feature in nominal position/orientation

$$\tilde{\mathbf{X}} = (\tilde{\mathbf{x_1}}, \ldots, \tilde{\mathbf{x_n}}) \ ,$$

while after the bestfit these are the n unknown footpoints

$$\bar{\mathbf{X}} = (\bar{\mathbf{x}}_\mathbf{1}, \ldots, \bar{\mathbf{x}}_\mathbf{n})$$

on the unknown substitute feature in the unknown final position/orientation.

4.4.2 Matrix Equation for Bestfit Solution

N-point related vectors

Analogously to the abbreviations $\hat{\mathbf{X}}$ for all measuring points and $\tilde{\mathbf{X}}$ for all nominal points, we use in the following capital boldface letters $\mathbf{L}$ to indicate a vector comprising all parameters referring to all n points. So we use the abbreviation

$$\mathbf{U} = (u_1, v_1, \ldots, u_n, v_n)$$

for the total of all $2n$ surfaces coordinates. Furthermore we define the vector

$$\mathbf{T} = (\mathbf{t}, \ldots, \mathbf{t})$$

composed of n identical translation vectors $\mathbf{t}$ and the matrix

$$Q(\mathbf{a}) = \bigoplus_{i=1}^{n} R(\mathbf{a}) \ .$$

This is a block diagonal matrix with n identical blocks $R(\mathbf{a})$.

Bestfit matrix equation

Now we can formulate the bestfit problem by the following simple, overdetermined matrix equation which has to be solved in the least squares sense:

$$Q(\mathbf{a}) \cdot (\mathbf{X}'(\mathbf{U}, \mathbf{p}) + \mathbf{T}) \approx \hat{\mathbf{X}} \ . \tag{4.19}$$

4.4.3 Preserving the Structure of Parametric Equations

1 equation for 1 coordinate

The above system contains exactly one equation for each coordinate. This helps to keep the optimization as simple as possible because we can carry out the position/orientation transformation (3.4) line by line.

Inviolate triple-structure

The original residuum

$$\mathbf{D} = (d_1, \ldots, d_n)$$

which involves the 3 parametric equations in 1 equation has been replaced by the residuum

$$\mathbf{\Delta X} = (\Delta x_1, \Delta y_1, \Delta z_1, \ldots, \Delta x_n, \Delta y_n, \Delta z_n) \ ,$$

and this leaves the useful triple structure of the parametric description untouched. Note: This is not possible when dealing with norms other than the L_2-norm.

4.4.4 Optimizing all Unknowns in Parallel

Vector of all unknowns

Collecting all unknowns in one vector

$$\mathbf{W} := (\mathbf{U}, \mathbf{p}, \mathbf{a}, \mathbf{t})$$

we can express the total of n arbitrary points on the surface as

$$\mathbf{X}(\mathbf{W}) := (\mathbf{x_1}(\mathbf{W}) \ldots \mathbf{x_n}(\mathbf{W}))$$

and thus (4.19) becomes a nonlinear least squares problem:

Find $\mathbf{W}$ such that

$$\| \ \mathbf{X}(\mathbf{W}) - \hat{\mathbf{X}} \ \| = \min \ . \tag{4.20}$$

Special structure of Jacobian matrix

Note: We can write $\mathbf{x_i}(\mathbf{W})$, even the individual functions $\mathbf{x_i}$ are not really dependent on all unknowns appearing in the vector W. This will result in a special structure of the Jacobian Matrix: Partial derivatives with respect to unappearing unknowns will vanish!

Each measuring point could belong to an other surface

For the time being we assume that $\mathbf{x_1} \ldots \mathbf{x_n}$ all refer to the same surface function. The concept that they can refer to different surface functions will prove quite useful when fitting compound features (see section 6.3).

Starting vector constituted by all nominal values

We can start the iteration with the $2n$ values of the surface coordinates of the n nominal points $\mathbf{U_0}$ together with the nominal dimensions $\mathbf{p_0}$ and the values $\mathbf{a_0}$ and $\mathbf{t_0}$ derived from nominal surface-position/orientation

$$\mathbf{W_0} := (\mathbf{U_0}, \mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0}). \tag{4.21}$$

Gauss-Newton iteration

By using Gauss-Newton iteration to solve (4.20) we get

$$J(\mathbf{W})\mathbf{\Delta W} \approx \mathbf{\Delta X}(\mathbf{W}) = \hat{\mathbf{X}} - \mathbf{X}(\mathbf{W}), \tag{4.22}$$

where $J(\mathbf{W})$ is the Jacobian of the function $\mathbf{X}(\mathbf{W})$.

$\mathbf{W}$ is refined by increments of $\mathbf{\Delta W}$ to find the final correction $\mathbf{\Delta W}^\star$ by which the orthogonality condition for local extremum

$$J^T(\mathbf{W}+\mathbf{\Delta W}^\star)\mathbf{\Delta X}(\mathbf{W}+\mathbf{\Delta W}^\star)=\mathbf{0} \tag{4.23}$$

is satisfied. ($\mathbf{\Delta X}(\mathbf{W}+\mathbf{\Delta W}^\star)$ is the final residual vector.)

4.5 Separating Geometry from Pos./Orient.

4.5.1 Surface Independence by Splitting the Jacobian

Splitting off the surface dependent part of the Jacobian

First we analyze the computation of the Jacobian matrix J. Not losing sight of surface function independence and separation of geometry from position/orientation, we try to split up the Jacobian matrix into a surface *dependent* part and a *general* part.

$\mathbf{X}$ depends indirectly on $\mathbf{U}$ and $\mathbf{p}$

From (3.4) we know that a point on a arbitrary surface is not a direct function of all its unknowns, i.e.:

$$\mathbf{x}(u,v,\mathbf{p},\mathbf{a},\mathbf{t})=\mathbf{x}(\mathbf{x}'(u,v,\mathbf{p}),\mathbf{a},\mathbf{t})\ . \tag{4.24}$$

Similarly we have

$$\mathbf{X}(\mathbf{W})=\mathbf{X}(\mathbf{U},\mathbf{p},\mathbf{a},\mathbf{t})=\mathbf{X}(\mathbf{X}'(\mathbf{U},\mathbf{p}),\mathbf{a},\mathbf{t}), \tag{4.25}$$

using $\mathbf{X}'$ as an abbreviation for the total of all untransformed coordinates in SPR

$$\mathbf{X}'=(\mathbf{x}'_\mathbf{1},\ldots,\mathbf{x}'_\mathbf{n}).$$

Formula for Jacobian computation

Knowing about this functional dependence, we can write for the Jacobian J

$$J(\mathbf{W})=\frac{\partial\mathbf{X}}{\partial\mathbf{W}}=\left[\ \frac{\partial\mathbf{X}}{\partial\mathbf{X}'}\cdot\left[\ \frac{\partial\mathbf{X}'}{\partial\mathbf{U}}\ ,\ \frac{\partial\mathbf{X}'}{\partial\mathbf{p}}\ \right]\ ,\ \frac{\partial\mathbf{X}}{\partial\mathbf{a}}\ ,\ \frac{\partial\mathbf{X}}{\partial\mathbf{t}}\ \right] \tag{4.26}$$

Definition of the partial Jacobian

if we define $\frac{\partial\mathbf{f}}{\partial\mathbf{e}}$ as the *partial* Jacobian matrix

$$\frac{\partial\mathbf{f}}{\partial\mathbf{e}}:=(J_{i,j})=(\frac{\partial f_i}{\partial e_j})$$

of a function

$$\mathbf{f}(\mathbf{g},\mathbf{e},\mathbf{h})$$

Jacobian of successive mappings

and the basic rule

$$\frac{\partial\mathbf{b}\left(\mathbf{c}\left(\mathbf{a}\right)\right)}{\partial\mathbf{a}}=\frac{\partial\mathbf{b}}{\partial\mathbf{c}}\cdot\frac{\partial\mathbf{c}}{\partial\mathbf{a}}$$

is applied accordingly.

Surface dependent parts of the Jacobian

In the next section, we will see that only the two partial Jacobians

$$\frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \quad \text{and} \quad \frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$$

are fully *dependent* of the actual surface geometry.

Surface-independent parts of the Jacobian

The two partial Jacobians

$$\frac{\partial \mathbf{X}}{\partial \mathbf{X}'} \quad \text{and} \quad \frac{\partial \mathbf{X}}{\partial \mathbf{t}}$$

are *independent* of the actual surface geometry, while

Mixed part of the Jacobian

$$\frac{\partial \mathbf{X}}{\partial \mathbf{a}}$$

can be *split* further into a surface dependent and a surface independent part.

$\mathbf{x}'$ as a 'generic' function

We assume $\mathbf{x}'(u, v, \mathbf{p})$ to be a 'generic' (or, expressed by an object-oriented term, to be a 'virtual') function, which can be substituted – while dealing with the actual geometry – by the real surface function. So we can define the full Jacobian $\frac{\partial \mathbf{X}}{\partial \mathbf{W}}$ for the general case.

4.5.2 Automated Generation of the Jacobian

Identifying 2 partial Jacobians as $Q(\mathbf{a})$

Considering again (4.19)

$$\mathbf{X}(\mathbf{W}) = Q(\mathbf{a}) \cdot (\mathbf{X}'(\mathbf{U}, \mathbf{p}) + \mathbf{T}) \approx \hat{\mathbf{X}} \,, \tag{4.27}$$

we recognize that the partial Jacobian $\frac{\partial \mathbf{X}}{\partial \mathbf{X}'}$ as well as $\frac{\partial \mathbf{X}}{\partial \mathbf{T}}$ are identical with the matrix $Q(\mathbf{a})$:

$$\frac{\partial \mathbf{X}}{\partial \mathbf{T}} = Q(\mathbf{a}) \qquad \text{and} \qquad \frac{\partial \mathbf{X}}{\partial \mathbf{X}'} = Q(\mathbf{a}) \,.$$

Thus

$$\frac{\partial \mathbf{X}}{\partial \mathbf{t}} = \frac{\partial \mathbf{X}}{\partial \mathbf{T}} \cdot \frac{\partial \mathbf{T}}{\partial \mathbf{t}} = Q(\mathbf{a}) \cdot \frac{\partial \mathbf{T}}{\partial \mathbf{t}} \,.$$

Jacobian written with $Q(\mathbf{a})$

This allows us to write the complete Jacobian matrix as

$$J(\mathbf{W}) = \frac{\partial \mathbf{X}}{\partial \mathbf{W}} = \left[\, Q(\mathbf{a}) \cdot \left[\, \frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \,,\, \frac{\partial \mathbf{X}'}{\partial \mathbf{p}} \, \right] \,,\, \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \,,\, Q(\mathbf{a}) \cdot \frac{\partial \mathbf{T}}{\partial \mathbf{t}} \, \right] \,. \tag{4.28}$$

Multiplication with the orthogonal matrix Q^T

By inserting this decomposition (4.28) and (4.27) into (4.22), equation (4.22) can be multiplied – without changing the norm of the residual – from the left side with the orthogonal matrix $Q^T(\mathbf{a})$ which results in

$$\left[\, \frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \,,\, \frac{\partial \mathbf{X}'}{\partial \mathbf{p}} \,,\, Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \,,\, \frac{\partial \mathbf{T}}{\partial \mathbf{t}} \, \right] \cdot \mathbf{\Delta W} \approx Q^T(\mathbf{a}) \hat{\mathbf{X}} - \mathbf{X}'(\mathbf{U}, \mathbf{p}) - \mathbf{T}(\mathbf{t}) \,. \tag{4.29}$$

Automated matrix generation

Now we would like to generate the actual matrix $Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{W}}(\mathbf{W})$ in an automated way based only on the calculation of the surface-function and its partial derivatives in SPR, independent of any position/orientation information

$$\mathbf{x}' \quad ; \quad \frac{\partial \mathbf{x}'}{\partial u}, \frac{\partial \mathbf{x}'}{\partial v} \quad ; \quad \frac{\partial \mathbf{x}'}{\partial p_1} \ \ldots \ \frac{\partial \mathbf{x}'}{\partial p_g} \ .$$

Iteration starts with nominal values

Let us consider starting with the nominal values $\mathbf{W_0}$ as defined by (4.21):

$$\mathbf{W_0} = (u_{1_0}, v_{1_0} \ldots u_{n_0}, v_{n_0}, \mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0}) \ .$$

Derivatives to dummy variables

By differentiating the components $\mathbf{x'_i}$ of $\mathbf{X}'$ with respect to the components u_j, v_j of $\mathbf{U}$ we get

$$\frac{\partial \mathbf{x'_i}}{\partial u_j} = \frac{\partial \mathbf{x'_i}}{\partial v_j} = 0 \qquad \text{if} \qquad i \neq j \ .$$

All points on the same surface

In section 4.4.2 we have assumed that all points belong to the same surface $\mathbf{x}'$, i.e.,

$$\mathbf{x'_i}(u_i, v_i, \mathbf{p}) = \mathbf{x}'(u, v, \mathbf{p})$$

Derivatives to all surface coordinates $\mathbf{U}$

so we get

$$\frac{\partial \mathbf{x'_i}}{\partial u_i}(\mathbf{W_0}) = \frac{\partial \mathbf{x}'}{\partial u}(u_{i_0}, v_{i_0}, \mathbf{p_0}) = \begin{pmatrix} \frac{\partial x'}{\partial u}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial y'}{\partial u}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial z'}{\partial u}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \end{pmatrix}$$

and

$$\frac{\partial \mathbf{x'_i}}{\partial v_i}(\mathbf{W_0}) = \frac{\partial \mathbf{x}'}{\partial v}(u_{i_0}, v_{i_0}, \mathbf{p_0}) = \begin{pmatrix} \frac{\partial x'}{\partial v}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial y'}{\partial v}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial z'}{\partial v}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \end{pmatrix} \ .$$

The partial Jacobian $\frac{\partial \mathbf{X}'}{\partial \mathbf{U}}$ then becomes

$$\frac{\partial \mathbf{X}'}{\partial \mathbf{U}}(\mathbf{W_0}) = \bigoplus_{i=1}^{n} \begin{bmatrix} \frac{\partial \mathbf{x}'}{\partial u}(u_{i_0}, v_{i_0}, \mathbf{p_0}) & \frac{\partial \mathbf{x}'}{\partial v}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \end{bmatrix} \ . \tag{4.30}$$

Derivatives to all geometry-parameters $\mathbf{p}$

By differentiating the components $\mathbf{x'_i}$ of $\mathbf{X}'$ with respect to the components $p_1 \ldots p_g$ of $\mathbf{p}$ we get

$$\frac{\partial \mathbf{x'_i}}{\partial p_j}(\mathbf{W_0}) = \frac{\partial \mathbf{x}'}{\partial p_j}(u_{i_0}, v_{i_0}, \mathbf{p_0}) = \begin{pmatrix} \frac{\partial x'}{\partial p_j}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial y'}{\partial p_j}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \\ \frac{\partial z'}{\partial p_j}(u_{i_0}, v_{i_0}, \mathbf{p_0}) \end{pmatrix} \ .$$

So we get for the partial Jacobian $\frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$

$$\frac{\partial \mathbf{X}'}{\partial \mathbf{p}}(\mathbf{W_0}) = \begin{bmatrix} \frac{\partial \mathbf{x}'}{\partial p_1}(u_{1_0}, v_{1_0}, \mathbf{p_0}) & \dots & \frac{\partial \mathbf{x}'}{\partial p_g}(u_{1_0}, v_{1_0}, \mathbf{p_0}) \\ \vdots & \vdots & \vdots \\ \frac{\partial \mathbf{x}'}{\partial p_1}(u_{n_0}, v_{n_0}, \mathbf{p_0}) & \dots & \frac{\partial \mathbf{x}'}{\partial p_g}(u_{n_0}, v_{n_0}, \mathbf{p_0}) \end{bmatrix} . \tag{4.31}$$

The partial Jacobian $\frac{\partial \mathbf{T}}{\partial \mathbf{t}}$ is simply

Derivatives to translation parameters $\mathbf{t}$

$$\frac{\partial \mathbf{T}}{\partial \mathbf{t}}(\mathbf{W_0}) = \begin{bmatrix} I_3 \\ \vdots \\ I_3 \end{bmatrix} . \tag{4.32}$$

We can evaluate the last part of the Jacobian $Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}}$ for the value $\mathbf{W_0}$ by

Derivatives to rotation parameters $\mathbf{a}$

$$\left(Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}}\right)(\mathbf{W_0}) = \tag{4.33}$$

$$Q^T(\mathbf{a_0}) \cdot \begin{bmatrix} Q_a & Q_b & Q_c \end{bmatrix} \cdot \begin{bmatrix} \mathbf{X}'(\mathbf{U_0}, \mathbf{p_0}) & 0 & 0 \\ 0 & \mathbf{X}'(\mathbf{U_0}, \mathbf{p_0}) & 0 \\ 0 & 0 & \mathbf{X}'(\mathbf{U_0}, \mathbf{p_0}) \end{bmatrix}$$

where

$$Q_a := \frac{\partial Q}{\partial a}(\mathbf{a_0}) = \bigoplus_{i=1}^{n} R_a(\mathbf{a_0})$$

$$Q_b := \frac{\partial Q}{\partial b}(\mathbf{a_0}) = \bigoplus_{i=1}^{n} R_b(\mathbf{a_0})$$

$$Q_c := \frac{\partial Q}{\partial c}(\mathbf{a_0}) = \bigoplus_{i=1}^{n} R_c(\mathbf{a_0}) ,$$

with

$$R_a = \begin{bmatrix} 0 & 0 & 0 \\ x & w & l \\ -v & -u & -k \end{bmatrix} \quad R_b = \begin{bmatrix} -bf & -bc & -e \\ kf & kc & -y \\ lf & lc & -z \end{bmatrix} \quad R_c = \begin{bmatrix} -m & n & 0 \\ -u & v & 0 \\ -w & x & 0 \end{bmatrix}$$

where we used the following abbreviations

Substitutions

$$a := \sin(a) , \quad b := \sin(b) , \quad c := \sin(c) , \quad d := \cos(a) , \quad e := \cos(b) , \quad f := \cos(c)$$

$$\begin{bmatrix} y & k \\ z & l \end{bmatrix} := \begin{bmatrix} a \\ d \end{bmatrix} \cdot \begin{bmatrix} b & e \end{bmatrix} \qquad \begin{bmatrix} m \\ n \end{bmatrix} := e \cdot \begin{bmatrix} c \\ f \end{bmatrix}$$

$$\begin{bmatrix} u & v \\ w & x \end{bmatrix} := \begin{bmatrix} y & d \\ z & -a \end{bmatrix} \cdot \begin{bmatrix} c & f \\ f & -c \end{bmatrix} .$$

Multiplication by R^T simplifies the R_x-matrices

We get some simplification by explicitly carrying out the multiplication

$$Q^T(\mathbf{a_0}) \cdot \left[\begin{array}{ccc} Q_a & Q_b & Q_c \end{array} \right]$$

with

$$Q^T \cdot Q_x = \bigoplus_{i=1}^{n} \left(R^T \cdot R_x \right)$$

where we get for

$$R^T \cdot R_a = \left[\begin{array}{ccc} 0 & -\sin(b) & -\cos(b)\sin(c) \\ \sin(b) & 0 & \cos(b)\cos(c) \\ \cos(b)\sin(c) & -\cos(b)\cos(c) & 0 \end{array} \right] \tag{4.34}$$

$$R^T \cdot R_b = \left[\begin{array}{ccc} 0 & 0 & -\cos(c) \\ 0 & 0 & -\sin(c) \\ \cos(c) & \sin(c) & 0 \end{array} \right] \tag{4.35}$$

$$R^T \cdot R_c = \left[\begin{array}{ccc} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{array} \right] . \tag{4.36}$$

Surface dependent part on the right hand side of the system of equations

The matrix additionally used here

$$\mathbf{X}'(\mathbf{U_0}, \mathbf{p_0}) = \left[\begin{array}{c} \mathbf{x}'(u_{1_0}, v_{1_0}, \mathbf{p_0}) \\ \\ \vdots \\ \\ \mathbf{x}'(u_{n_0}, v_{n_0}, \mathbf{p_0}) \end{array} \right]$$

is the only surface dependent matrix which appears also on the *right hand* side of the overdetermined system of equations

$$\left(Q^T(\mathbf{a_0}) \cdot \hat{\mathbf{X}} - \mathbf{T}(\mathbf{t_0}) \right) - \mathbf{X}'(\mathbf{U_0}, \mathbf{p_0}) \, .$$

Automatized construction-process of 3 surface dependent matrices

We see that all 3 remaining, surface *dependent* matrices

$$\mathbf{X}'(\mathbf{U}, \mathbf{p}) \, , \qquad \frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \qquad \text{and} \qquad \frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$$

can be constructed by evaluating n times the surface function in the simple SPR $\mathbf{x}'(u, v, \mathbf{p})$ plus its partial derivatives $\frac{\partial \mathbf{x}'}{\partial u}$, $\frac{\partial \mathbf{x}'}{\partial v}$ and $\frac{\partial \mathbf{x}'}{\partial p_j}$ at the n locations $(u_1, v_1) \ldots (u_n, v_n)$.

Jacobian has always same structure

The *structure* of this surface dependent part of the full Jacobian always remains the same, no matter which surface function $\mathbf{x}'$ is actually involved in the bestfit process. This allows us to use always the same procedure for any further solutions of the especially structured system of equations.

Chapter 5

Solutions

5.1 The Overdetermined Linear System

5.1.1 Solution Methods

Two approaches

Basically there are two possibilities to solve (4.22):

Normal equations

First, we can insert equation (4.22) into equation (4.23). Thus we get the well-known Gaussian normal equations. This way we have transformed an overdetermined system of equations into a system with an equal number of equations and unknowns. Its solution satisfies the overdetermined system of equations in the least squares sense.

Orthogonal transformations

The same result can be obtained by transforming $\mathbf{\Delta X}$ (given by (4.22)) with successive orthogonal transformations (Givens or Householder) which have the property of leaving the L_2-norm (of $\mathbf{\Delta X}$) unchanged. We will continue doing this until we arrive at a coordinate system where the solution can be computed in a trivial way.

Drawbacks and advantages of both methods

From the numerical point of view we prefer the second method due to its increased numerical stability. From the practical point of view, there are some reasons to prefer the first method, e.g., to economize memory space. Because we solve the resulting system of equations iteratively, numerical inaccuracies are self-correcting to a certain extent.

Exploiting structure

For the two possibilities we will provide solution methods that exploit the special structure of the resulting system of equations.

Combined procedure

Searching for the most time efficient solution method, we will recognize that a mixed procedure (Givens combined with normal equations) will give the best results.

Automated handling of redundancies in the parametric representation

Finally we are considering an alternative method based on singular value decomposition (SVD) which is more cost-intensive but which allows for the automatic handling of redundancies in the parametric representation. This is especially

useful while bestfitting standard surfaces or modifying the shape of polynomial surfaces by simultaneously varying coefficients and surface coordinates.

Summary

In short: *Givens transformations* preserve numerical *stability* best, *normal equations* are most economic regarding *memory* space, the here proposed *mixed* procedure (Givens/ normal equations) is most *time* efficient, while a combination with *SVD* (Givens/ SVD) allows us to treat *redundancies* in a more general and automatized manner.

5.1.2 Structure

Structure stays fixed

The structure of the $(3n) \times (2n + g + 6)$ matrix $Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{W}}$, i.e., the structure of the overdetermined system of linear equations to be solved during the solution process of the nonlinear least squares problem, is always the same ('x' denotes a nonzero element):

$$Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{W}} = \left[\frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \, , \, \frac{\partial \mathbf{X}'}{\partial \mathbf{p}} \, , \, Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \, , \, \frac{\partial \mathbf{T}}{\partial \mathbf{t}} \right] =$$

$$\begin{bmatrix}
x\,x & 0\,0 & \cdots & 0\,0 & x \cdots x & 1 & 0 & 0 \\
x\,x & 0\,0 & \cdots & 0\,0 & x \cdots x & 0 & 1 & 0 \\
x\,x & 0\,0 & \cdots & 0\,0 & x \cdots x & 0 & 0 & 1 \\
 & & & & & & & \\
0\,0 & x\,x & \cdots & 0\,0 & x \cdots x & 1 & 0 & 0 \\
0\,0 & x\,x & \cdots & 0\,0 & x \cdots x & 0 & 1 & 0 \\
0\,0 & x\,x & \cdots & 0\,0 & x \cdots x & 0 & 0 & 1 \\
 & & & & & & & \\
\vdots\,\vdots & \vdots\,\vdots & \cdots & \vdots\,\vdots & \vdots \cdots \vdots & \vdots & \vdots & \vdots \\
 & & & & & & & \\
0\,0 & 0\,0 & \cdots & x\,x & x \cdots x & 1 & 0 & 0 \\
0\,0 & 0\,0 & \cdots & x\,x & x \cdots x & 0 & 1 & 0 \\
0\,0 & 0\,0 & \cdots & x\,x & x \cdots x & 0 & 0 & 1
\end{bmatrix} . \qquad (5.1)$$

Known sparse structure

We can see that first the system is extremely sparse (especially for a large number of measuring points n, when $\frac{\partial \mathbf{X}'}{\partial \mathbf{U}}$ becomes large), second we know the locations of the zeros in advance. So we can omit them from the beginning, i.e., we do not have to store them, and we are using an algorithm not involving them in any calculations.

Efficient solution method

This will give us a very efficient solution method with computational costs and memory demands growing no more than *linearly* with the number of measuring points. This fact makes FUNKE competitive also in the field of standard surfaces compared with conventional, implicit distance-function based algorithms.

5.2 Solution by Orthogonal Transformations

5.2.1 Givens Rotations Interpreted as Geometrical Transformations

Permutation of equations

Permuting the equations, by separating the x, y and z coordinate blocks according to $(x_1, y_1, z_1, \ldots, x_n, y_n, z_n) \mapsto (x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n)$, yields

$$\left[\begin{array}{cccccccccccc}
x & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & 1 & 0 & 0 \\
0 & 0 & x & x & \cdots & 0 & 0 & x & \cdots & x & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & x & x & x & \cdots & x & 1 & 0 & 0 \\
 & & & & & & & & & & & & \\
x & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & 0 & 1 & 0 \\
0 & 0 & x & x & \cdots & 0 & 0 & x & \cdots & x & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & x & x & x & \cdots & x & 0 & 1 & 0 \\
 & & & & & & & & & & & & \\
x & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & 0 & 0 & 1 \\
0 & 0 & x & x & \cdots & 0 & 0 & x & \cdots & x & 0 & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & x & x & x & \cdots & x & 0 & 0 & 1
\end{array}\right].$$

Permutation of unknowns

Permuting the unknowns $(u_1, v_1, \ldots, u_n, v_n) \mapsto (u_1, \ldots, u_n, v_1, \ldots, v_n)$ yields

$$\left[\begin{array}{cccccccccccc}
x & 0 & \cdots & 0 & x & 0 & \cdots & 0 & x & \cdots & x & 1 & 0 & 0 \\
0 & x & \cdots & 0 & 0 & x & \cdots & 0 & x & \cdots & x & 1 & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & x & 0 & 0 & \cdots & x & x & \cdots & x & 1 & 0 & 0 \\
 & & & & & & & & & & & & & \\
x & 0 & \cdots & 0 & x & 0 & \cdots & 0 & x & \cdots & x & 0 & 1 & 0 \\
0 & x & \cdots & 0 & 0 & x & \cdots & 0 & x & \cdots & x & 0 & 1 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & x & 0 & 0 & \cdots & x & x & \cdots & x & 0 & 1 & 0 \\
 & & & & & & & & & & & & & \\
x & 0 & \cdots & 0 & x & 0 & \cdots & 0 & x & \cdots & x & 0 & 0 & 1 \\
0 & x & \cdots & 0 & 0 & x & \cdots & 0 & x & \cdots & x & 0 & 0 & 1 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & x & 0 & 0 & \cdots & x & x & \cdots & x & 0 & 0 & 1
\end{array}\right].$$

Triagonalization by Givens rotations

With $3n$ Givens rotations we can triagonalize the sparse part of the matrix. (The

symbol '**o**' stands for an eliminated element.)

$$\left[\begin{array}{cccccccccccc}
x & 0 & \cdots & 0 & x & 0 & \cdots & 0 & x & \cdots\cdots & x \\
0 & x & \cdots & 0 & 0 & x & \cdots & 0 & x & \cdots\cdots & x \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots\cdots & \vdots \\
0 & 0 & \cdots & x & 0 & 0 & \cdots & x & x & \cdots\cdots & x \\
 & & & & & & & & & & \\
\mathbf{o} & 0 & \cdots & 0 & x & 0 & \cdots & 0 & x & \cdots\cdots & x \\
0 & \mathbf{o} & \cdots & 0 & 0 & x & \cdots & 0 & x & \cdots\cdots & x \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots\cdots & \vdots \\
0 & 0 & \cdots & \mathbf{o} & 0 & 0 & \cdots & x & x & \cdots\cdots & x \\
 & & & & & & & & & & \\
\mathbf{o} & 0 & \cdots & 0 & \mathbf{o} & 0 & \cdots & 0 & x & \cdots\cdots & x \\
0 & \mathbf{o} & \cdots & 0 & 0 & \mathbf{o} & \cdots & 0 & x & \cdots\cdots & x \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots\cdots & \vdots \\
0 & 0 & \cdots & \mathbf{o} & 0 & 0 & \cdots & \mathbf{o} & x & \cdots\cdots & x
\end{array}\right] \tag{5.2}$$

Correspondence between a Givens rotation and a geometrical rotation

We obtain this by constructing a single Givens rotation by selecting the two corresponding rows from 2 of the 3 x-, y-, z-coordinate blocks. So the resulting Givens rotation is always from one of the 3 forms, (3.6),(3.7) or (3.8). Thus, it can be interpreted as a *real* geometrical rotation. First, to zero the leading element of the i-th row of the y-coordinate block, we rotate this row in combination with the i-th row of the x-coordinate block. Second, to zero the leading element of the i-th row of the z-coordinate block, we rotate this row in combination with the i-th row of the x-coordinate block. Third, to zero again the leading element of the i-th row of the z-coordinate block, we rotate this row in combination with the i-th row of the y-coordinate block. By this, we see that these three consecutive Givens rotations represent the geometrical 3D-rotation

$$R(\mathbf{a_i}) = R_x(a_i) \cdot R_y(b_i) \cdot R_z(c_i) \ ,$$

In total we get as many 3D-rotations as we have measuring points.

Transformed normals have only z-component

Remembering that the first 2n columns represent the $2n$ partial derivatives of the actual surface points with respect to their surface coordinates u and v, we see that all n transformed $\frac{\partial \mathbf{x}}{\partial u}$ have an x-component only. Furthermore all n transformed cross products between each pair of partial derivatives $\frac{\partial \mathbf{x}}{\partial u}$ and $\frac{\partial \mathbf{x}}{\partial v}$, $(\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v})$ have a z-component only

$$\left(\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v}\right)^{\star} = \begin{pmatrix} 0 \\ 0 \\ x \end{pmatrix}$$

because the transformed $\frac{\partial \mathbf{x}}{\partial u}$ and $\frac{\partial \mathbf{x}}{\partial v}$ are of the form

$$\frac{\partial \mathbf{x}^{\star}}{\partial u} = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad \frac{\partial \mathbf{x}^{\star}}{\partial v} = \begin{pmatrix} x \\ x \\ 0 \end{pmatrix} \ .$$

Actual surface normal

The cross-product $\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v}$ points always into the direction of the actual surface normal $\mathbf{n}(u, v)$

$$\left(\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v}\right)(u, v) \sim \mathbf{n}(u, v) \; . \tag{5.3}$$

Rotation of difference vectors into a special tangential plane coordinate system

Geometrically this means, that each difference vector $\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}(u_i, v_i, \mathbf{p})$ is transformed by a (triple) rotation $R_x(a) \cdot R_y(b) \cdot R_z(c)$ in a coordinate system which consists of the actual $\frac{\partial \mathbf{x}}{\partial u}$ as the new x-vector, a second vector of the actual tangential plane as the new y-vector and finally a new z-vector having the same direction as the actual surface normal vector.

Matrix G performs pre-triagonalization

It follows that multiplying the original matrix (5.1) from the left side by an orthogonal matrix G of the form

$$G := \bigoplus_{i=1}^{n} R(\mathbf{a_i}) \tag{5.4}$$

with the geometrical 3D rotations $R(\mathbf{a_1})$, $R(\mathbf{a_2})$, ... , $R(\mathbf{a_n})$ has the same effect as the above operations (apart from the permutations of equations and unknowns).

Merging 3 successive rotations

Once we can determine these 3D-rotations by a purely algebraic method (which is numerically most stable) by *determining* 3 successive Givens rotations without executing them immediately.

R(u,v) as an analytic function of u, v

Based on the above geometrical considerations we can furthermore express R as an *analytic* function of u, v with the help of the following geometric algorithm:

Algorithm for calculating R directly

1. Calculate
$$\mathbf{r}_1(u, v) = \frac{\frac{\partial \mathbf{x}}{\partial u}}{\| \frac{\partial \mathbf{x}}{\partial u} \|} \tag{5.5}$$
2. Calculate
$$\mathbf{r}_2^\star = \frac{\partial \mathbf{x}}{\partial v} - (\mathbf{r}_1, \frac{\partial \mathbf{x}}{\partial v}) \cdot \mathbf{r}_1 \tag{5.6}$$
3. Calculate
$$\mathbf{r}_2(u, v) = \frac{\mathbf{r}_2^\star}{\| \mathbf{r}_2^\star \|} \tag{5.7}$$
4. Calculate
$$\mathbf{r}_3(u, v) = \mathbf{r}_1 \times \mathbf{r}_2 \tag{5.8}$$
5. Form
$$R(u, v) = \begin{bmatrix} \mathbf{r}_1^T(u, v) \\ \mathbf{r}_2^T(u, v) \\ \mathbf{r}_3^T(u, v) \end{bmatrix} \tag{5.9}$$

Geometrical explanation of the algorithm

From

$$\begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} = R \cdot \frac{\partial \mathbf{x}}{\partial u} \quad \text{and} \quad \begin{pmatrix} 0 \\ 0 \\ x \end{pmatrix} = R \cdot \left(\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v} \right)$$

it follows that

$$R^T \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \frac{\frac{\partial \mathbf{x}}{\partial u}}{\| \frac{\partial \mathbf{x}}{\partial u} \|} = \mathbf{r_1} \quad \text{and} \quad R^T \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \frac{\mathbf{n}}{\| \mathbf{n} \|} .$$

From the construction of $\mathbf{r}_2$ (Gram-Schmidt-step) it follows that $\| r_2 \| = 1$, that it is orthogonal to $\mathbf{r}_1$, and that it lies in the tangential plane. From the construction of $\mathbf{r}_3$ it follows that $\mathbf{r}_3$ is the normalized direction vector $\frac{\mathbf{n}}{\|\mathbf{n}\|}$ of the tangential plane and that $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ establish a right hand coordinate system. Thus we get

$$R^T \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \mathbf{r}_2$$

and it is shown that R can be constructed by step 5.

Improving step 2

Step two of the above algorithm (constructing an orthogonal vector to r_1 by Gram-Schmidt) could be unstable. Practically this would occur if the u, v-isoparametric lines would nearly coincide. A usual method to avoid numerical instability in this case is to re-orthogonalize $r_2^\star$ by repeating step 2, substituting $\frac{\partial \mathbf{x}}{\partial v}$ by the still inaccurate $r_2^\star$.

Triple rotation $R(\mathbf{a_i})$ costs less

Calculating directly the matrices $R(\mathbf{a_i})$ is somewhat less cost-intensive than successive single Givens rotations. We substitute 3 single rotations, each taking 6 floating point operations (flops) (=total of 18 flops), by 1 3D-rotation taking 15 flops. Furthermore we save the transformation of the submatrix $\frac{\partial \mathbf{X}}{\partial \mathbf{T}}$ because we know that we can directly write

$$G \cdot Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{W}} = \begin{bmatrix} x & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & \\ o & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & R(\mathbf{a_1}) \\ o & o & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x & \\ 0 & 0 & x & x & \cdots & 0 & 0 & x & \cdots & x & \\ 0 & 0 & o & x & \cdots & 0 & 0 & x & \cdots & x & R(\mathbf{a_2}) \\ 0 & 0 & o & o & \cdots & 0 & 0 & x & \cdots & x & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & x & x & x & \cdots & x & \\ 0 & 0 & 0 & 0 & \cdots & o & x & x & \cdots & x & R(\mathbf{a_n}) \\ 0 & 0 & 0 & 0 & \cdots & o & o & x & \cdots & x & \end{bmatrix} . \qquad (5.10)$$

Equation-permutation matrix

Matrix (5.10) can be brought into the triagonalized form (5.2) by permutations of equations and unknowns as described above. We define a matrix P_e with

dimension 3n which performs the permutation of equations

$$P_e := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} . \tag{5.11}$$

To arrange the equations, P_e has to be applied from the left side.

Unknown-permutation matrix

For the permutation of unknowns we define the matrix P_u with dimension 2n+g+6

$$P_u := \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} . \tag{5.12}$$

To arrange the unknowns, P_u has to be applied from the right side.

Explicit expression for the pre-triangular form

By transforming the system of equations (4.22) by all these matrices we get an *explicit* expression, which the pre-triangular form (5.2) of the Jacobian is part of:

$$(P_e \cdot G \cdot Q^T \cdot J(\mathbf{W}) \cdot P_u) \cdot ({P_u}^T \cdot \mathbf{\Delta W}) \approx P_e \cdot G \cdot Q^T \cdot (\hat{\mathbf{X}} - \mathbf{X}(\mathbf{W})) . \tag{5.13}$$

Orthogonal transformations do not change solution

Because all these transformations are orthogonal we do not change the solution of the original, overdetermined system of equations (4.22). Starting from (5.13) we can triagonalize the matrix conventionally by further Givens rotations.

Exploiting sparse structure

Using the method described above, we have fully taken advantage of the fact that the matrix is sparse for a large number of measuring points. For backward substitution, we can also take advantage of this by a specialized implementation.

As we exactly know the locations of the zeros in the matrix we can skip them during backward substitution. Additionally we can save memory space by not working with the standard matrix representation but implementing the special matrix structure shown above.

So with Givens Rotations we obtain the following computational cost, which depends *linearly* on the number of measuring points n: Costs of 'Givens'

	Multiplic.		Additions		Flops
Pre-Triag.	$9(g+5)n$	+	$6(g+5)n$	=	$(15g+75)n$
Nrm.Triag.	$2(g+6)^2n$	+	$(g+6)^2n$	=	$(3g^2+36g+108)n$
Back.Subst.	$(2g+15)n$ $+\frac{(g+6)^2}{2}$	+	# mult.	=	$(4g+30)n$ $+(g+6)^2$
Total					$(3g^2+55g+213)n+(g+6)^2$

Note: $n \gg g$!

('# mult.' in the third column indicates that the number of additions is the same as the number of multiplications.)

5.2.2 Polynomial Surfaces

The special case of a polynomial surface is a very important one, especially in the CAD-field. So, while dealing with data coming from a CAD-interface, it is important to have an algorithm which performs exact bestfit on this kind of surface representation in an efficient way. Today these algorithms are not available in state-of-the-art software systems. Polynomial surface: Important special case

Different types of *base polynomials* $p_{kl}(u,v)$ exist: Standard polynomials, Bézier polynomials, b-splines, etc. They can be converted to each other by linear transformations because they cover the same linear space as long as they have the same degree in u and v direction. Different types of base polynomials

We can define a general polynomial surface $\mathbf{x}'(u,v,\mathbf{p})$ of degree s/t by Definition of a polynomial surface

$$x'(u,v,\mathbf{p}) = \sum_{k=0}^{s}\sum_{l=0}^{t} a_{x_{kl}} \cdot p_{kl}(u,v)$$

$$y'(u,v,\mathbf{p}) = \sum_{k=0}^{s}\sum_{l=0}^{t} a_{y_{kl}} \cdot p_{kl}(u,v)$$

$$z'(u,v,\mathbf{p}) = \sum_{k=0}^{s}\sum_{l=0}^{t} a_{z_{kl}} \cdot p_{kl}(u,v)\ .$$

From the mathematical point of view, especially with regard to FUNKE, there are some specialities of a polynomial surface function to be taken into consideration. First a triple polynomial as defined above has a much larger vector $\mathbf{p}$ Specialities

than a standard surface. The components are identical to the polynomial coefficients. Second, the partial derivatives with respect to these geometry parameters (polynomial coefficients) $\mathbf{p}$ are represented by the base polynomials. Finally, we cannot describe position/orientation and geometry by really independent parameters. They depend on each other.

Roto-translation of a polynomial

That is to say that a polynomial transformed by an arbitrary roto-translation (1.13) remains a polynomial of the same degree but with different coefficients. They describe the *identical* geometry but in another *position/orientation*. The individual coefficients transform like x, y, z-coordinates:

$$R \cdot (\mathbf{x}'(u, v, \mathbf{p}) + \mathbf{t}) =$$

$$R \cdot \left(\sum_{k=0}^{s} \sum_{l=0}^{t} \mathbf{a_{kl}} \cdot p_{kl}(u, v) + \mathbf{t} \right) = \sum_{k=0}^{s} \sum_{l=0}^{t} \mathbf{a^{\star}_{kl}} \cdot p_{kl}(u, v) \tag{5.14}$$

with

$$\mathbf{a^{\star}_{kl}} = R \cdot (\mathbf{a_{kl}} + \mathbf{t}) \quad \text{for} \quad k = l = 0 \quad (\text{if} \quad p_{00}(u, v) \equiv 1)$$

and

$$\mathbf{a^{\star}_{kl}} = R \cdot \mathbf{a_{kl}} \quad \text{for} \quad k, l > 0$$

No simultaneous variation of all parameters

Thus position and orientation are implicitly contained in the polynomial coefficients. As a consequence, we should not vary $\mathbf{a}$ and $\mathbf{t}$ simultaneously with $\mathbf{p}$, as we would get a singular system of equations otherwise.

Two different problems

Considering polynomial fit, two different problems are distinguished in section 1.2.5 (see also [GR89]):

Approximation

One of these – especially important in the CAD-field – is the approximation problem (geometrical fit) of a sculptured surface (surface not describable by analytical functions). Normally this problem leads to a system of *linear* equations for the polynomial coefficients. A more sophisticated approach ([Hen74], [SH76], [BCF92]), where the surface coordinates u and v are varied additionally, leads to a *nonlinear* optimization process. Dealing with a sculptured surface (constructed by a multitude of polynomials) instead of dealing with a single polynomial, the approximation problem can be solved in a *hundred* different ways, for better or worse. It cannot even be decided which way is really the best. Research on this topic increased especially in the last few years [Sch94], when powerful optical measuring devices which can deliver a lot of measuring points ($\geq 250'000$) in a few seconds became available [Bre93], [Wil93].

Position/orientation bestfit

The second problem, especially important in CM and in the CAQ/CAM-field, is the *position/orientation* fit of a sculptured surface. In contrast to the first problem, this can be solved only in *one* single way for any given objective function. Taking into account what we said above, we can correctly solve this problem only by separating position/orientation from geometry, the concept FUNKE is based on. We have to freeze the geometry parameters $\mathbf{p}$ and to vary the position/orientation parameters $\mathbf{a}$ and $\mathbf{t}$ only. So, we can form the system of equations

for bestfit in the same way as we would fit a surface containing no geometry parameters ($g = 0$) (e.g. a plane).

The former problem, i.e., the approximation problem, is covered by FUNKE as well. We can choose an approximation with (the more sophisticated way) or without simultaneous variation of the surface coordinates u and v. Because orthogonal transformations give more numerical stability, it is reasonable to use them for solving the bestfit problem for high degree surfaces. Fitting polynomial coefficients with simultaneous variation of the surface coordinates u and v is rather cost-intensive. We should therefore exploit all possible simplifications for this case.

Both covered by FUNKE

First, by varying the coefficients of a polynomial surface the terms including the partial derivatives to rotation and translation can be ignored because $\mathbf{a}$ and $\mathbf{t}$ are frozen. Second, if we use the same base polynomials for all 3 coordinates we have the same derivatives to the geometry parameters for all 3 coordinates.

No rotat./transl.-derivatives

Instead of (5.1) we have in this case:

Special 'polynomial' structure

$$\begin{bmatrix} \mathbf{x}_{u_{(1)}} & \mathbf{x}_{v_{(1)}} & 0 & 0 & \cdots & 0 & p_{00_{(1)}} \cdot I & \cdots & p_{st_{(1)}} \cdot I \\ 0 & 0 & \mathbf{x}_{u_{(2)}} & \mathbf{x}_{v_{(2)}} & \cdots & 0 & p_{00_{(2)}} \cdot I & \cdots & p_{st_{(2)}} \cdot I \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \mathbf{x}_{v_{(n)}} & p_{00_{(n)}} \cdot I & \cdots & p_{st_{(n)}} \cdot I \end{bmatrix}, \qquad (5.15)$$

where the abbreviations $\mathbf{x}_{u_{(i)}}$, $\mathbf{x}_{v_{(i)}}$ mean the vectors

$$\mathbf{x}_{u_{(i)}} := \frac{\partial \mathbf{x}'}{\partial u}(u_i, v_i, \mathbf{p}) \quad \text{and} \quad \mathbf{x}_{v_{(i)}} := \frac{\partial \mathbf{x}'}{\partial v}(u_i, v_i, \mathbf{p})$$

and the abbreviations $p_{00_{(i)}} \ldots p_{st_{(i)}}$ mean the scalar values

$$p_{00_{(i)}} := p_{00}(u_i, v_i), \quad \ldots \quad , p_{st_{(i)}} := p_{st}(u_i, v_i)$$

of the $s \cdot t$ scalar basic polynomials p_{kl}.

Furthermore we can compute the pre-triagonalized form of the resulting system of equations as an explicit function of the surface coordinates (u, v) by using (5.5), (5.6), (5.7), (5.8) and (5.9). This is especially advantageous for polynomials.

Pre-triag.form as a function of u, v

So $G \cdot Q^T \cdot \frac{\partial \mathbf{J}}{\partial \mathbf{W}}$ simplifies to:

Simplified form

$$\begin{bmatrix}
x^\star_{u_{(1)}} & x^\star_{v_{(1)}} & 0 & \cdots & 0 & 0 & & & \\
o & y^\star_{v_{(1)}} & 0 & \cdots & 0 & 0 & p_{00_{(1)}} \cdot R(u_1, v_1) & \cdots & p_{st_{(1)}} \cdot R(u_1, v_1) \\
o & o & 0 & \cdots & 0 & 0 & & & \\
0 & 0 & x^\star_{u_{(2)}} & \cdots & 0 & 0 & & & \\
0 & 0 & o & \cdots & 0 & 0 & p_{00_{(2)}} \cdot R(u_2, v_2) & \cdots & p_{st_{(2)}} \cdot R(u_2, v_2) \\
0 & 0 & o & \cdots & 0 & 0 & & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & x^\star_{u_{(n)}} & x^\star_{v_{(n)}} & & & \\
0 & 0 & 0 & \cdots & o & y^\star_{v_{(n)}} & p_{00_{(n)}} \cdot R(u_n, v_n) & \cdots & p_{st_{(n)}} \cdot R(u_n, v_n) \\
0 & 0 & 0 & \cdots & o & o & & &
\end{bmatrix} \tag{5.16}$$

where

$$\mathbf{x}^\star_{u_{(i)}} \quad := \quad R(u_i, v_i) \cdot \mathbf{x}_{u_{(i)}} = \begin{pmatrix} \| \mathbf{x}_{u_{(i)}} \| \\ 0 \\ 0 \end{pmatrix} \tag{5.17}$$

and

$$\mathbf{x}^\star_{v_{(i)}} \quad := \quad R(u_i, v_i) \cdot \mathbf{x}_{v_{(i)}} = \begin{pmatrix} \mathbf{r}_1^T(u_i, v_i) \cdot \mathbf{x}_{v_{(i)}} \\ \| \mathbf{r}^\star_2(u_i, v_i) \| \\ 0 \end{pmatrix} . \tag{5.18}$$

Simplifications using (5.5),(5.6),(5.7)

In (5.17) we have taken into account that $\mathbf{r}_1^T \cdot \mathbf{x}_{u_{(i)}} = \| \mathbf{x}_{u_{(i)}} \|$ based on the definition (5.5) of $\mathbf{r}_1$. In (5.18) we have taken into account that $\mathbf{r}_2^T \cdot \mathbf{x}_{v_{(i)}} = \| \mathbf{r}^\star_2 \|$ based on the definition (5.7) of $\mathbf{r}_2$ and the definition (5.6) of $\mathbf{r}^\star_2$. If the u, v-isoparametric lines intersect at right angles, i.e, $\mathbf{x}^T_{u_{(i)}} \cdot \mathbf{x}_{v_{(i)}} = 0$ (see section 5.3.3) we can replace (5.17) and (5.18) by

$$\mathbf{x}^\star_{u_{(i)}} = \begin{pmatrix} \| \mathbf{x}_{u_{(i)}} \| \\ 0 \\ 0 \end{pmatrix} \tag{5.19}$$

and

$$\mathbf{x}^\star_{v_{(i)}} = \begin{pmatrix} 0 \\ \| \mathbf{x}_{v_{(i)}} \| \\ 0 \end{pmatrix} . \tag{5.20}$$

Thus we have discussed all possible simplifications for polynomials.

5.3 Solution by Normal Equations

5.3.1 Automated Generation of Normal Equations

Saving memory space

Using normal equations is a numerically less stable way to solve (4.22) than solv-

ing it by orthogonal transformations as discussed in the previous section. On the other hand, there is some saving of memory space. We achieve this by calculating directly the normal equations without explicitly forming the overdetermined system of equations. This could be of some importance when dealing with a large number of measuring points. Furthermore it is useful to consider the normal equations to obtain some theoretical insight and for comparisons.

The system of normal equations can be split up into subparts Sub-matrices

$$\begin{bmatrix} U^TU & U^TP & U^TA & U^TT \\ P^TU & P^TP & P^TA & P^TT \\ A^TU & A^TP & A^TA & A^TT \\ T^TU & T^TP & T^TA & T^TT \end{bmatrix} \cdot \mathbf{\Delta W} = \begin{bmatrix} U^TD \\ P^TD \\ A^TD \\ T^TD \end{bmatrix} \tag{5.21}$$

where

$$U := \frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \qquad P := \frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$$

$$A := Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \qquad T := \frac{\partial \mathbf{T}}{\partial \mathbf{t}}$$

$$D := Q^T(\mathbf{a})\hat{\mathbf{X}} - \mathbf{X}'(\mathbf{U}, \mathbf{p}) - \mathbf{T} \; .$$

First, we define the i-th actual surface point as Function-value in SPR

$$\mathbf{x}_{(i)} = \begin{pmatrix} x_{(i)} \\ y_{(i)} \\ z_{(i)} \end{pmatrix} \quad := \quad \mathbf{x}'(u_i, v_i, \mathbf{p}) = \begin{pmatrix} x'(u_i, v_i, \mathbf{p}) \\ y'(u_i, v_i, \mathbf{p}) \\ z'(u_i, v_i, \mathbf{p}) \end{pmatrix} , \tag{5.22}$$

and further the partial derivatives at the i-th footpoint as Partial derivatives in SPR

$$\mathbf{x}_{u_{(i)}} = \begin{pmatrix} x_{u_{(i)}} \\ y_{u_{(i)}} \\ z_{u_{(i)}} \end{pmatrix} \quad := \quad \frac{\partial \mathbf{x}'}{\partial u}(u_i, v_i, \mathbf{p}) = \begin{pmatrix} \frac{\partial x'}{\partial u}(u_i, v_i, \mathbf{p}) \\ \frac{\partial y'}{\partial u}(u_i, v_i, \mathbf{p}) \\ \frac{\partial z'}{\partial u}(u_i, v_i, \mathbf{p}) \end{pmatrix} \tag{5.23}$$

$$\mathbf{x}_{v_{(i)}} = \begin{pmatrix} x_{v_{(i)}} \\ y_{v_{(i)}} \\ z_{v_{(i)}} \end{pmatrix} \quad := \quad \frac{\partial \mathbf{x}'}{\partial v}(u_i, v_i, \mathbf{p}) = \begin{pmatrix} \frac{\partial x'}{\partial v}(u_i, v_i, \mathbf{p}) \\ \frac{\partial y'}{\partial v}(u_i, v_i, \mathbf{p}) \\ \frac{\partial z'}{\partial v}(u_i, v_i, \mathbf{p}) \end{pmatrix} \tag{5.24}$$

and

$$\mathbf{x}_{p_{j_{(i)}}} = \begin{pmatrix} x_{p_{j_{(i)}}} \\ y_{p_{j_{(i)}}} \\ z_{p_{j_{(i)}}} \end{pmatrix} \quad := \quad \frac{\partial \mathbf{x}'}{\partial p_j}(u_i, v_i, \mathbf{p}) = \begin{pmatrix} \frac{\partial x'}{\partial p_j}(u_i, v_i, \mathbf{p}) \\ \frac{\partial y'}{\partial p_j}(u_i, v_i, \mathbf{p}) \\ \frac{\partial z'}{\partial p_j}(u_i, v_i, \mathbf{p}) \end{pmatrix} . \tag{5.25}$$

We define the difference vector between the i-th measuring point and its footpoint as Coordinate differences

$$\mathbf{\Delta x}_{(i)} = \begin{pmatrix} \Delta x_{(i)} \\ \Delta y_{(i)} \\ \Delta z_{(i)} \end{pmatrix} \quad := \quad R^T(\mathbf{a})\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}'(u_i, v_i, \mathbf{p}) - \mathbf{t} \; . \tag{5.26}$$

Auxiliary vectors

Additionally we define for each point 3 auxiliary vectors

$$\begin{aligned} \mathbf{x}_{a_{(i)}} &:= R^T(\mathbf{a}) \cdot R_a(\mathbf{a}) \cdot \mathbf{x}'(u_i, v_i, \mathbf{p}) \\ \mathbf{x}_{b_{(i)}} &:= R^T(\mathbf{a}) \cdot R_b(\mathbf{a}) \cdot \mathbf{x}'(u_i, v_i, \mathbf{p}) \\ \mathbf{x}_{c_{(i)}} &:= R^T(\mathbf{a}) \cdot R_c(\mathbf{a}) \cdot \mathbf{x}'(u_i, v_i, \mathbf{p}) \; . \end{aligned}$$

By using (4.34), (4.35) and (4.36) we get

$$\mathbf{x}_{a_{(i)}} = \begin{pmatrix} x_{a_{(i)}} \\ y_{a_{(i)}} \\ z_{a_{(i)}} \end{pmatrix} = \begin{pmatrix} -\sin(b)y_{(i)} - \cos(b)\sin(c)z_{(i)} \\ \sin(b)x_{(i)} + \cos(b)\cos(c)z_{(i)} \\ \cos(b)\sin(c)x_{(i)} - \cos(b)\cos(c)y_{(i)} \end{pmatrix} \tag{5.27}$$

$$\mathbf{x}_{b_{(i)}} = \begin{pmatrix} x_{b_{(i)}} \\ y_{b_{(i)}} \\ z_{b_{(i)}} \end{pmatrix} = \begin{pmatrix} -\cos(c)z_{(i)} \\ -\sin(c)z_{(i)} \\ \cos(c)x_{(i)} + \sin(c)y_{(i)} \end{pmatrix} \tag{5.28}$$

$$\mathbf{x}_{c_{(i)}} = \begin{pmatrix} x_{c_{(i)}} \\ y_{c_{(i)}} \\ z_{c_{(i)}} \end{pmatrix} = \begin{pmatrix} y_{(i)} \\ -x_{(i)} \\ 0 \end{pmatrix} \; . \tag{5.29}$$

Explicit subparts

Thus we can express these subparts explicitly. From the diagonal subparts we have only to calculate the upper part because they are symmetric (the corresponding elements are indicated by (%)) :

$$U^TU = \begin{bmatrix} \mathbf{x}^T_{u_{(1)}} \cdot \mathbf{x}_{u_{(1)}} & \mathbf{x}^T_{u_{(1)}} \cdot \mathbf{x}_{v_{(1)}} & 0 & \cdots & 0 & 0 & 0 \\ \% & \mathbf{x}^T_{v_{(1)}} \cdot \mathbf{x}_{v_{(1)}} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \mathbf{x}^T_{u_{(n)}} \cdot \mathbf{x}_{u_{(n)}} & \mathbf{x}^T_{u_{(n)}} \cdot \mathbf{x}_{v_{(n)}} \\ 0 & 0 & 0 & \cdots & 0 & \% & \mathbf{x}^T_{v_{(n)}} \cdot \mathbf{x}_{v_{(n)}} \end{bmatrix}$$

$$P^TP = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{x}^T_{p1_{(i)}} \cdot \mathbf{x}_{p1_{(i)}} & \sum_{i=1}^{n} \mathbf{x}^T_{p1_{(i)}} \cdot \mathbf{x}_{p2_{(i)}} & \cdots & \sum_{i=1}^{n} \mathbf{x}^T_{p1_{(i)}} \cdot \mathbf{x}_{pg_{(i)}} \\ \% & \sum_{i=1}^{n} \mathbf{x}^T_{p2_{(i)}} \cdot \mathbf{x}_{p2_{(i)}} & \cdots & \sum_{i=1}^{n} \mathbf{x}^T_{p2_{(i)}} \cdot \mathbf{x}_{pg_{(i)}} \\ \vdots & \vdots & \ddots & \vdots \\ \% & \% & \cdots & \sum_{i=1}^{n} \mathbf{x}^T_{pg_{(i)}} \cdot \mathbf{x}_{pg_{(i)}} \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{x}_{a_{(i)}}^T \cdot \mathbf{x}_{a_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{a_{(i)}}^T \cdot \mathbf{x}_{b_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{a_{(i)}}^T \cdot \mathbf{x}_{c_{(i)}} \\ \% & \sum_{i=1}^{n} \mathbf{x}_{b_{(i)}}^T \cdot \mathbf{x}_{b_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{b_{(i)}}^T \cdot \mathbf{x}_{c_{(i)}} \\ \% & \% & \sum_{i=1}^{n} \mathbf{x}_{c_{(i)}}^T \cdot \mathbf{x}_{c_{(i)}} \end{bmatrix}$$

$$T^T T = \begin{bmatrix} n & 0 & 0 \\ \% & n & 0 \\ \% & \% & n \end{bmatrix}$$

$$U^T P = \begin{bmatrix} \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{p1_{(1)}} & \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{p2_{(1)}} & \cdots & \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{pg_{(1)}} \\ \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{p1_{(1)}} & \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{p2_{(1)}} & \cdots & \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{pg_{(1)}} \\ \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{p1_{(2)}} & \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{p2_{(2)}} & \cdots & \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{pg_{(2)}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{p1_{(n)}} & \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{p2_{(n)}} & \cdots & \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{pg_{(n)}} \end{bmatrix}$$

$$U^T A = \begin{bmatrix} \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{a_{(1)}} & \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{b_{(1)}} & \mathbf{x}_{u_{(1)}}^T \cdot \mathbf{x}_{c_{(1)}} \\ \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{a_{(1)}} & \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{b_{(1)}} & \mathbf{x}_{v_{(1)}}^T \cdot \mathbf{x}_{c_{(1)}} \\ \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{a_{(2)}} & \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{b_{(2)}} & \mathbf{x}_{u_{(2)}}^T \cdot \mathbf{x}_{c_{(2)}} \\ \vdots & \vdots & \vdots \\ \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{a_{(n)}} & \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{b_{(n)}} & \mathbf{x}_{v_{(n)}}^T \cdot \mathbf{x}_{c_{(n)}} \end{bmatrix}$$

$$U^T T = \begin{bmatrix} x_{u_{(1)}} & y_{u_{(1)}} & z_{u_{(1)}} \\ x_{v_{(1)}} & y_{v_{(1)}} & z_{v_{(1)}} \\ x_{u_{(2)}} & y_{u_{(2)}} & z_{u_{(2)}} \\ \vdots & \vdots & \vdots \\ x_{v_{(n)}} & y_{v_{(n)}} & z_{v_{(n)}} \end{bmatrix}$$

$$P^T A = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{x}_{p1_{(i)}}^T \mathbf{x}_{a_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{p1_{(i)}}^T \mathbf{x}_{b_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{p1_{(i)}}^T \mathbf{x}_{c_{(i)}} \\ \sum_{i=1}^{n} \mathbf{x}_{p2_{(i)}}^T \mathbf{x}_{a_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{p2_{(i)}}^T \mathbf{x}_{b_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{p2_{(i)}}^T \mathbf{x}_{c_{(i)}} \\ \vdots & \vdots & \vdots \\ \sum_{i=1}^{n} \mathbf{x}_{pg_{(i)}}^T \mathbf{x}_{a_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{pg_{(i)}}^T \mathbf{x}_{b_{(i)}} & \sum_{i=1}^{n} \mathbf{x}_{pg_{(i)}}^T \mathbf{x}_{c_{(i)}} \end{bmatrix}$$

$$P^T T = \begin{bmatrix} \sum_{i=1}^{n} x_{p1_{(i)}} & \sum_{i=1}^{n} y_{p1_{(i)}} & \sum_{i=1}^{n} z_{p1_{(i)}} \\ \sum_{i=1}^{n} x_{p2_{(i)}} & \sum_{i=1}^{n} y_{p2_{(i)}} & \sum_{i=1}^{n} z_{p2_{(i)}} \\ \vdots & \vdots & \vdots \\ \sum_{i=1}^{n} x_{pg_{(i)}} & \sum_{i=1}^{n} y_{pg_{(i)}} & \sum_{i=1}^{n} z_{pg_{(i)}} \end{bmatrix}$$

$$A^T T = \begin{bmatrix} \sum_{i=1}^{n} x_{a_{(i)}} & \sum_{i=1}^{n} y_{a_{(i)}} & \sum_{i=1}^{n} z_{a_{(i)}} \\ \sum_{i=1}^{n} x_{b_{(i)}} & \sum_{i=1}^{n} y_{b_{(i)}} & \sum_{i=1}^{n} z_{b_{(i)}} \\ \sum_{i=1}^{n} x_{c_{(i)}} & \sum_{i=1}^{n} y_{c_{(i)}} & \sum_{i=1}^{n} z_{c_{(i)}} \end{bmatrix}.$$

Constant vector on the right hand side

For the right hand side we get

$$U^T D = \begin{bmatrix} \mathbf{x}^T_{u_{(1)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(1)} \\ \mathbf{x}^T_{v_{(1)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}^T_{u_{(n)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(n)} \\ \mathbf{x}^T_{v_{(n)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(n)} \end{bmatrix} \qquad P^T D = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{x}^T_{p1_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \\ \sum_{i=1}^{n} \mathbf{x}^T_{p2_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \\ \vdots \\ \sum_{i=1}^{n} \mathbf{x}^T_{pg_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \end{bmatrix}$$

$$A^T D = \begin{bmatrix} \sum_{i=1}^{n} \mathbf{x}^T_{a_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \\ \sum_{i=1}^{n} \mathbf{x}^T_{b_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \\ \sum_{i=1}^{n} \mathbf{x}^T_{c_{(i)}} \cdot \boldsymbol{\Delta}\mathbf{x}_{(i)} \end{bmatrix} \qquad T^T D = \begin{bmatrix} \sum_{i=1}^{n} \Delta x_{(i)} \\ \sum_{i=1}^{n} \Delta y_{(i)} \\ \sum_{i=1}^{n} \Delta z_{(i)} \end{bmatrix} .$$

This shows the following: At each point i $(1 \leq i \leq n)$ we can evaluate

Evaluating the surface function and its partial derivatives on n locations

$$\mathbf{x}_{(i)} \quad , \quad \mathbf{x}_{u_{(i)}} \quad , \quad \mathbf{x}_{v_{(i)}} \quad , \quad \mathbf{x}_{p1_{(i)}} \ldots \mathbf{x}_{pg_{(i)}} \tag{5.30}$$

by means of the given surface defining parametric function.

Based on these values we can also calculate for each point i:

Evaluating the auxiliary vectors on n locations

$$\boldsymbol{\Delta}\mathbf{x}_{(i)} \quad , \quad \mathbf{x}_{a_{(i)}} \quad , \quad \mathbf{x}_{b_{(i)}} \quad , \quad \mathbf{x}_{c_{(i)}} \, . \tag{5.31}$$

Thus we have everything to generate the system of equations in a general way and independent of the actual surface.

The computational cost to form the normal equations is

Costs to form the normal equations

	Multiplic.		Additions		Flops
U^TU	$n \cdot 9$	+	$n \cdot 6$	=	$15n$
P^TP	$\frac{3ng^2}{2}$	+	# mult.	=	$3g^2n$
A^TA	$3 \cdot n \cdot 3 + 3 \cdot n \cdot 2$	+	# mult.	=	$30n$
T^TT	0	+	0	=	0
U^TP	$2n \cdot g \cdot 3$	+	$2n \cdot g \cdot 2$	=	$10gn$
U^TA	$2 \cdot 2n \cdot 3 + 1 \cdot 2n \cdot 2$	+	$2 \cdot 2n \cdot 2 + 1 \cdot 2n \cdot 1$	=	$26n$
U^TT	0	+	0	=	0
P^TA	$2 \cdot ng \cdot 3 + 1 \cdot ng \cdot 2$	+	# mult.	=	$16gn$
P^TT	0	+	$3gn$	=	$3gn$
A^TT	0	+	$2 \cdot n \cdot 3 + 1 \cdot n \cdot 2$	=	$8n$
U^TD	$2n \cdot 3$	+	$2n \cdot 2$	=	$10n$
P^TD	$n \cdot g \cdot 3$	+	# mult.	=	$6gn$
A^TD	$2 \cdot n \cdot 3 + 1 \cdot n \cdot 2$	+	# mult.	=	$16n$
T^TD	0	+	$3n$	=	$3n$
Total					$(3g^2 + 35g + 108)n$

which is again a *linear* function of the number of measuring points n!

5.3.2 Solving the Normal Equations

Normal equations structure

The whole system of normal equations (5.21) has the following form (like in domain decomposition [GvL89]):

$$\begin{bmatrix} x & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x \\ \% & x & 0 & 0 & \cdots & 0 & 0 & x & \cdots & x \\ 0 & 0 & x & x & \cdots & 0 & 0 & x & \cdots & x \\ 0 & 0 & \% & x & \cdots & 0 & 0 & x & \cdots & x \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & x & x & x & \cdots & x \\ 0 & 0 & 0 & 0 & \cdots & \% & x & x & \cdots & x \\ & & & & & & & & & \\ \% & \% & \% & \% & \cdots & \% & \% & x & \cdots & x \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \% & \% & \% & \% & \cdots & \% & \% & \% & \cdots & x \end{bmatrix} = \begin{bmatrix} x \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x \end{bmatrix}. \tag{5.32}$$

Exploiting symmetry

The above matrix is symmetric; we do not have to store (nor calculate) the elements marked by (%).

Transformed submatrix stays symmetric

Consider performing a normal Gaussian elimination with diagonal pivoting on a symmetric matrix. Then the quadratic submatrix arising after each column elimination (which consists of the original matrix reduced by the already eliminated

columns and corresponding rows) is again symmetric:

$$\begin{bmatrix} x & x & x & \cdots & x \\ \% & x & x & \cdots & x \\ \% & \% & x & \cdots & x \\ \% & \% & \% & \cdots & x \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \% & \% & \% & \cdots & x \end{bmatrix} \rightarrow \quad \text{1st column elimination} \quad \rightarrow \begin{bmatrix} x & x & x & \cdots & x \\ 0 & x & x & \cdots & x \\ 0 & \% & x & \cdots & x \\ 0 & \% & \% & \cdots & x \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \% & \% & \cdots & x \end{bmatrix} .$$

Proof : Let a_{ik} be the element in the i-th row and k-th column of the equation matrix. Applying the multiplication factor $\frac{a_{i1}}{a_{11}}$ to this row, we get for the transformed element Proof

$$a^{\star}_{ik} = a_{ik} - a_{1k} \cdot \frac{a_{i1}}{a_{11}} .$$

Analogously we could transform the corresponding element in the k-th row and i-th column with

$$a^{\star}_{ki} = a_{ki} - a_{1i} \cdot \frac{a_{k1}}{a_{11}} .$$

The postulated symmetry property of the original matrix guarantees that

$$a_{ik} = a_{ki}; \quad a_{1k} = a_{k1}; \quad a_{i1} = a_{1i} .$$

So, we have

$$a^{\star}_{ik} = a_{ki} - a_{k1} \cdot \frac{a_{1i}}{a_{11}} = a_{ki} - a_{1i} \cdot \frac{a_{k1}}{a_{11}} = a^{\star}_{ki} \quad \text{q.e.d.} \qquad .$$

This fact can be used for a very simple algorithm delivering the LDL^T decomposition of a symmetric matrix. The algorithm is even somewhat simpler to implement than the one proposed in [GvL89] for the same purpose. We modify the normal Gaussian elimination in such a way that we transform only the part of a row which belongs to the part above the diagonal. Because we know that the resulting submatrices are still symmetric we do not have to calculate the lower part explicitly. This saves about half the cost (same as Cholesky). In the lower part we store the scaling factors. LDL^T-decomposition

Algorithm 1 (in MATLAB notation) Simple algorithm for LDL^T decomposition

```
for i = 1 : m − 1                                  {choose pivot row}
  for k = i + 1 : m                                {choose succeeding rows}
    A(k, i) = A(k, i)/A(i, i);                     {calculate scaling factor}
    A(k, k : m) = A(k, k : m) − A(k, i) ∗ A(i, k : m);  {Gaussian elim.step}
  end;
end;
```

L and DL^T

After the execution of the above code, the upper part of the resulting matrix represents the matrix DL^T while the part below the diagonal represents L (without the trivial diagonal). The costs are about

$$\frac{m^3}{3} \underbrace{=}_{\text{actual case: } m=2n+g+6} \frac{(2n+g+6)^3}{3} \quad \text{flops} \quad .$$

Substituting upper part with lower part

A further advantage compared to the algorithm in [GvL89] is that we are able to cut in half the memory demands by a little modification. Because the actually treated submatrix is still symmetric, we can substitute each element below the diagonal which has to be accessed with the corresponding element above the diagonal. Instead of writing the scaling factors into the lower part of the matrix we overwrite the upper part:

LDL^T decomposition with the half of memory demands

Algorithm 2

```
for i = 1 : m − 1                                  {choose pivot row}
  for k = i + 1 : m                                {choose succeeding rows}
    f = A(i, k)/A(i, i);                           {calculate scaling factor}
    A(k, k : m) = A(k, k : m) − f ∗ A(i, k : m);   {Gaussian elimination step}
  end;
  A(i, i + 1 : m) = A(i, i + 1 : m)/A(i, i);       {transform pivot row}
end;
```

D and L^T

After the execution of the above code, the diagonal represents the diagonal of the matrix D while the part above the diagonal represents L^T (without the trivial diagonal). So we need only half the space in memory.

Pivoting

It is highly recommended to pivot when triagonalizing a matrix with normal Gaussian elimination. Using the above algorithm, we cannot freely choose the pivot element. To preserve symmetry we have to swap equations and unknowns simultaneously, i.e., to choose a pivot element along the diagonal. But it is shown in [GvL89] (p.141) that it is absolutely safe not to pivot if we perform the LDL^T factorization on a positive definite matrix. This is the type of matrix we are dealing with in this case.

Comparison to 'Cholesky'

We do not use Cholesky at similar costs because we do not have to compute square roots with the modified Gaussian elimination. Because we have a block diagonal matrix, these operations are significant compared with the total computational effort. Using the above method, we can inhibit the unnecessary steps with the zero elements involved by a few modifications to the code.

LDL^T decomposition on the actual sparse structure

Algorithm 3

```
for i = 1 : m − 1                                  {choose pivot row}
  if rem(i,2)==1,                                  {off-diagonal elements}
    f = A(i, i + 1)/A(i, i);                       {calculate scaling factor}
```

```
    A(i, i+1) = f;                                  {overwrite}
    A(i+1, 2*n+1 : m) = A(i+1, 2*n+1 : m)           {elim.full columns only}
    − f * A(i, 2*n+1 : m);
  end;
  for k = 2*n+1 : m                                 {full rows only}
    f = A(i, k)/A(i, i);                            {calculate scaling factor}
    A(k, k : m) = A(k, k : m) − f * A(i, k : m);    {Gaussian elimination step}
  end;
  A(i, 2*n+1 : m) = A(i, 2*n+1 : m)/A(i, i);        {trsf.full columns only}
end;
```

Corresponding backward substitution

We have fully taken advantage of the fact that the matrix is sparse. Analogously, we can perform the backward substitution.

Useful LDL^T decomposition

During a nonlinear optimization process it is quite useful to have the full decomposition LDL^T available. That is, in case we want to save costs, we shall perform some successive iteration steps with an unchanged Jacobian matrix. The matrix L can be used then to solve efficiently the systems with same equation matrix but different right hand side vectors.

Costs for normal equations elimination

Costs for eliminations:

	Multiplic.		Additions		Flops
Off-diag.El.	$n \cdot (g+6)$	+	# mult.	=	$(2g+12)n$
Band Matrix	$\frac{2n(g+6)^2}{2}$	+	# mult.	=	$(2g^2+24g+72)n$
Triag.Matrix	$\frac{(g+6)^3}{6}$	+	# mult.	=	$\frac{(g+6)^3}{3}$
Total					$(2g^2+26g+84)n + \frac{(g+6)^3}{3}$

Costs for backward substitution

Costs for backward substitution:

	Multiplic.		Additions		Flops
Off-diag.El.	n	+	# mult.	=	$2n$
Band Matrix	$2n \cdot (g+6)$	+	# mult.	=	$(4g+24)n$
Triag.Matrix	$\frac{(g+6)^2}{2}$	+	# mult.	=	$(g+6)^2$
Total					$(4g+26)n + (g+6)^2$

Total costs

We get the total cost using normal equations

$$(5g^2+65g+218) \cdot n + \frac{(g+6)^3}{3} + (g+6)^2 \qquad \text{flops},$$

again *proportional* to n.

5.3.3 Discussion of the Normal Equations

Orthogonal isoparametric lines

There are some elements like planes, cylinders, spheres, toroids, where, in the usual parametric description, the u- and v-isoparametric lines always intersect at right angles.

Disappearing off-diagonal

Right from the beginning we can take advantage of this fact and form the system of equations without any off-diagonal elements, i.e., U^TU as a diagonal-matrix. The off-diagonal elements are calculated by

$$\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial v}$$

and are all equal to zero in these cases. As a consequence the off-diagonal elimination steps can be omitted.

Example: Toroid

As an example consider the toroid. It is described in SPR by

$$\begin{array}{rcl} x(u,v,R,r) & = & (R + r \cdot \cos(v)) \cdot \cos(u) \\ y(u,v,R,r) & = & (R + r \cdot \cos(v)) \cdot \sin(u) \\ z(u,v,R,r) & = & r \cdot \sin(v) \end{array} .$$

We get for the directions of the isoparametric lines at location (u, v)

$$\frac{\partial \mathbf{x}}{\partial u} = \begin{pmatrix} \frac{\partial x}{\partial u}(u,v,R,r) \\ \frac{\partial y}{\partial u}(u,v,R,r) \\ \frac{\partial z}{\partial u}(u,v,R,r) \end{pmatrix} = \begin{pmatrix} -(R + r \cdot \cos(v)) \cdot \sin(u) \\ (R + r \cdot \cos(v)) \cdot \cos(u) \\ 0 \end{pmatrix}$$

and

$$\frac{\partial \mathbf{x}}{\partial v} = \begin{pmatrix} \frac{\partial x}{\partial v}(u,v,R,r) \\ \frac{\partial y}{\partial v}(u,v,R,r) \\ \frac{\partial z}{\partial v}(u,v,R,r) \end{pmatrix} = \begin{pmatrix} -r \cdot \sin(v) \cdot \cos(u) \\ -r \cdot \sin(v) \cdot \sin(u) \\ r \cdot \cos(v) \end{pmatrix}$$

So $\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial v}$ vanishes for arbitrary surface coordinates u and v

$$\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial v} = R\sin(u) r \sin(v)\cos(u) + r\cos(v)\sin(u) r \sin(v)\cos(u)$$

$$-R\cos(u) r \sin(v)\sin(u) + r\cos(v)\cos(u) r \sin(v)\sin(u) + 0 = 0 .$$

Variation of geometry

Furthermore there are some elements (e.g. cylinder, sphere) where

$$\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{p}} = 0 \quad \text{and} \quad \frac{\partial \mathbf{x}}{\partial v}^T \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{p}} = 0 .$$

This can be interpreted geometrically in the sense that a change of a dimension always results in the displacement of the surface points in the direction of the actual surface normal.

Example: sphere

We show this for a sphere. It is described in SPR by

$$\begin{aligned} x(u,v,r) &= r \cdot \cos(v) \cdot \cos(u) \\ y(u,v,r) &= r \cdot \cos(v) \cdot \sin(u) \\ z(u,v,r) &= r \cdot \sin(v) \end{aligned} \quad .$$

The direction of the isoparametric lines at location (u, v) is

$$\frac{\partial \mathbf{x}}{\partial u} = \begin{pmatrix} \frac{\partial x}{\partial u}(u,v,r) \\ \frac{\partial y}{\partial u}(u,v,r) \\ \frac{\partial z}{\partial u}(u,v,r) \end{pmatrix} = \begin{pmatrix} -r \cdot \cos(v) \cdot \sin(u) \\ r \cdot \cos(v) \cdot \cos(u) \\ 0 \end{pmatrix}$$

and

$$\frac{\partial \mathbf{x}}{\partial v} = \begin{pmatrix} \frac{\partial x}{\partial v}(u,v,r) \\ \frac{\partial y}{\partial v}(u,v,r) \\ \frac{\partial z}{\partial v}(u,v,r) \end{pmatrix} = \begin{pmatrix} -r \cdot \sin(v) \cdot \cos(u) \\ -r \cdot \sin(v) \cdot \sin(u) \\ r \cdot \cos(v) \end{pmatrix} .$$

By differentiating with respect to the dimension parameter r we obtain

$$\frac{\partial \mathbf{x}}{\partial r} = \begin{pmatrix} \frac{\partial x}{\partial v}(u,v,r) \\ \frac{\partial y}{\partial v}(u,v,r) \\ \frac{\partial z}{\partial v}(u,v,r) \end{pmatrix} = \begin{pmatrix} \cos(v) \cdot \cos(u) \\ \cos(v) \cdot \sin(u) \\ \sin(v) \end{pmatrix} .$$

So $\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial r}$ becomes

$$\frac{\partial \mathbf{x}}{\partial u}^T \cdot \frac{\partial \mathbf{x}}{\partial r} = -r\cos(v)\sin(u)\cos(v)\cos(u) + r\cos(v)\cos(u)\cos(v)\sin(u) + 0 = 0$$

and $\frac{\partial \mathbf{x}}{\partial v}^T \cdot \frac{\partial \mathbf{x}}{\partial r}$

$$\frac{\partial \mathbf{x}}{\partial v}^T \cdot \frac{\partial \mathbf{x}}{\partial r} = \underbrace{-r\sin(v)\cos(u)\cos(v)\cos(u) - r\sin(v)\sin(u)\cos(v)\sin(u)}_{-r\sin(v)\cos(v)(\cos(u)^2+\sin(u)^2)}$$

$$+r\cos(v)\sin(v) = 0 .$$

As a consequence the two submatrices $U^T P$ and $P^T U$ will always vanish.

Interpretation

How can we interpret the sparse structure of (5.32)? We recognize that the vector of the partial derivatives with respect to the surface coordinates $\frac{\partial \mathbf{X}}{\partial u_i}$ / $\frac{\partial \mathbf{X}}{\partial v_i}$ is orthogonal to all others $\frac{\partial \mathbf{X}}{\partial u_j}$ / $\frac{\partial \mathbf{X}}{\partial v_j}$ with $j \neq i$. But in general it is *not* orthogonal to the partial derivatives with respect to the geometry and position/orientation parameters $\frac{\partial \mathbf{X}}{\partial \mathbf{p}}$ / $\frac{\partial \mathbf{X}}{\partial \mathbf{a}}$ / $\frac{\partial \mathbf{X}}{\partial \mathbf{t}}$. The geometrical interpretation is obvious: Varying the location of a single surface point does not directly influence the locations of other surface points, but varying the position/orientation of the whole surface or its geometry changes the locations of all surface-points. Thus there is an indirect interaction (via pos./ orient. and geometry of the whole surface) between the locations of the individual surface points.

Neglected coupling

If we neglect this correlation between surface coordinates (u_i, v_i) on the one side

and position/orientation plus geometry $\mathbf{p}$, $\mathbf{a}$ and $\mathbf{t}$ on the other side, by zeroing the corresponding submatrices in (5.21):

$$\begin{bmatrix} U^TU & 0 & 0 & 0 \\ 0 & P^TP & P^TA & P^TT \\ 0 & A^TP & A^TA & A^TT \\ 0 & T^TP & T^TA & T^TT \end{bmatrix} \cdot \mathbf{\Delta W} = \begin{bmatrix} U^TD \\ P^TD \\ A^TD \\ T^TD \end{bmatrix}, \tag{5.33}$$

Splits into sub-systems

the linearized system of equations is split up into the subsystem

$$\begin{bmatrix} P^TP & P^TA & P^TT \\ A^TP & A^TA & A^TT \\ T^TP & T^TA & T^TT \end{bmatrix} \cdot \begin{bmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{bmatrix} = \begin{bmatrix} P^TD \\ A^TD \\ T^TD \end{bmatrix} \tag{5.34}$$

n small sub-systems for the n points

and into n small 2×2 subsystems of the form

$$\begin{bmatrix} x & x \\ x & x \end{bmatrix} \cdot \begin{pmatrix} \Delta u_i \\ \Delta v_i \end{pmatrix} = \begin{pmatrix} x \\ x \end{pmatrix}. \tag{5.35}$$

Overdetermined system of nonlinear equations

Iterating these 2×2 sub-systems gives the least squares solutions as functions of u and v of the n overdetermined nonlinear system of equations

$$R \cdot (\mathbf{x}'(u, v, \mathbf{p}) + \mathbf{t}) \approx \hat{\mathbf{x}}_\mathbf{i}\ .$$

Nearest distance search

Solving such a system of equations in the least squares sense corresponds to the geometrical process of searching for nearest distance from a (measuring) point to a given surface. See numerical example in section 6.3.2.4 (second chart).

Alternating parameter-fit

Alternatively solving these n sub-systems with $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$ obtained by (5.34), inserting the results $(u_1, v_1, \ldots, u_n, v_n)$ into (5.34) again, etc., would correspond to the method explained in 1.2.5 (e.g. mentioned in [Ker87], [Gaw89]). The search for orthogonal deviations does not occur *in parallel* with the correction of position/ orientation and geometry. Keeping in mind these mathematical differences we can now compare this simplified method against FUNKE.

When comparable?

The method will become comparable to our method only in the case that these coupling terms are negligible. In the case where the partial derivatives with respect to the surface coordinates are constant over the whole surface (e.g. plane: always the same normal vector), the system of equations can be solved in two steps: First, determine position/orientation, second, determine the orthogonal distances. The tangential plane in any point is equivalent to the whole surface, i.e., we get the correct result after these two steps. The closer a sculptured surface is to a plane, the better this method will work. On the other hand, already a simple surface with a slight curvature (e.g. sphere) or a compound feature (see numerical example in section 6.3.2.4) is absolutely unsuitable for this approach.

5.4 Cost Optimized Solution

5.4.1 Comparison between 'Givens' and normal equations

Generally normal equations less cost-intensive than 'Givens'

We know that Givens rotations have the advantage of being numerically much more stable than normal equations. However, normal equations need less memory space than the factoring by Givens rotations, implementing the normal equations directly (without an overdetermined system) as shown in the last section. This could be of some importance when dealing with large VDA-FS-Files and a big number of measuring points. In general, normal equations are about $4\times$ less cost-intensive than 'Givens' and $2\times$ less cost-intensive than 'Fast-Givens' [GvL89].

Actually Givens less cost-intensive

Comparing now the solution methods of the preceding section purely by cost, we recognize that in our case the solution by 'Givens' is less cost-intensive than the solution by normal equations.

Better structure-exploitation by 'Givens'

We recognize as the reason for this astonishing fact that Givens is more suitable to exploit the special sparsity. For pre-triagonalization we need $(15g+75)n$ flops only. There we have no g^2n-dependency. The remaining full matrix which we have to treat by a 'normal' Givens-procedure is only of size $n \times (g+6)$. Thus we get a cost-dependency of $3g^2n$ for its triagonalization by Givens rotations.

3x smaller subsystem

On the other hand, the system of normal equations as described in the previous section, has dimension $(2n+g+6) \times (2n+g+6)$. Even though it has a sparsity structure as well, the cost-dependence is $3g^2n$ for its construction and $2g^2n$ for its triagonalization. This is a greater g^2n-dependence than with Givens because we have to treat a much larger system of equations in this case.

5.4.2 Mixed Method

Combining 'Givens' with normal equations

Obviously, to optimize cost it is advantageous to use Givens for pre-triagonalization because this way we arrive – without g^2n-dependence – at quite a small matrix (dimension: $n \cdot (g+6)$). Afterwards we change to a more efficient solution method like normal equations or Householder to complete the triagonalization.

Pre-triagonalized structure

Imagine we have already performed the pre-triagonalization. We arrive at an overdetermined system of equations with the following structure

$$\begin{bmatrix} C & E \\ O_{n\times 2n} & B \end{bmatrix} \cdot \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \approx \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} , \tag{5.36}$$

where C is a quadratic matrix of dimension $2n$ and B is n by $g+6$. Especially in our case, C is a triangular matrix with a specific structure.

Solving an overdetermined subsystem

If we solve now the overdetermined system of equations in the least squares sense

$$B \cdot \mathbf{y} \approx \mathbf{b} \tag{5.37}$$

and compute afterwards the vector $\mathbf{x}$ by backsubstitution

$$\mathbf{x} = C^{-1} \cdot (\mathbf{a} - E \cdot \mathbf{y}) \,, \tag{5.38}$$

Same solution as the complete system

we get the same solution $\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ as if we would by solving the complete overdetermined system of equations (5.36) in the least squares sense.

QR-Decomposition of B

Let Q and R be the matrices of the QR-decomposition of B, i.e.:

$$B = Q \cdot R \,.$$

Constructing an orthogonal transformation for the full system

Then

$$\begin{bmatrix} I_{2n\times 2n} & O_{2n\times n} \\ O_{n\times 2n} & Q^T \end{bmatrix}$$

represents an orthogonal matrix which we can use to multiply (5.36) without influence on its least squares solution. Thus we get

$$\begin{bmatrix} C & E \\ O_{n\times 2n} & R \end{bmatrix} \cdot \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \approx \begin{pmatrix} \mathbf{a} \\ Q^T \cdot \mathbf{b} \end{pmatrix} \,.$$

Solution by backward substitution

Taking into account the special structure of R (nothing but zeros below the diagonal), we recognize that $\mathbf{y}$ is the solution vector of (5.37) as well as the partial solution vector of (5.36), solving both in the least squares sense.

5.4.3 Separated Solution

Identifying the partial matrices

In the following we apply the above method to our algorithm. Identifying the partial matrices and vectors C, E, B, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{x}$ and $\mathbf{y}$, we can form the equations (5.37) and (5.38) for the present case.

Approach for the 'modified surface-footpoints'

The vector $\mathbf{y}$ contains the parameters primarily interesting for bestfit, i.e., position/orientation and geometry $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$ while $\mathbf{x}$ contains the surface coordinate vector $\mathbf{U}$. The equation for $\mathbf{x}$ (5.38) can be interpreted as delivering directly the necessary corrections to the surface coordinates (u_i, v_i) of the footpoints if the surface parameters ($\mathbf{y} = (\mathbf{p}, \mathbf{a}, \mathbf{t})$) change by $\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t}$).

Transforming to structure (5.36)

We have to perform the pre-triagonalization (5.13) to obtain a system of equations with structure (5.36). To identify C, E, B, $\mathbf{a}$ and $\mathbf{b}$ in the system (5.13) we use the abbreviations U, P, A, T and D of section 5.3.1 for the individual submatrices:

Abbreviations U,P,A,T,D

Remember :

$$U := \frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \qquad P := \frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$$

$$A := Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \qquad T := \frac{\partial \mathbf{T}}{\partial \mathbf{t}}$$

$$D := Q^T(\mathbf{a})\hat{\mathbf{X}} - \mathbf{X}'(\mathbf{U}, \mathbf{p}) - \mathbf{T}$$

$$G := \bigoplus_{i=1}^{n} R(\mathbf{a_i}) \qquad \text{defined in (5.4)},$$

and P_e, P_u (defined in (5.11)/(5.12)) are permutation matrices, which permute the equations (P_e), respectively the unknowns (P_u)

Splitting the equation matrix

With these abbreviations the pre-triagonalized system of equations (5.13) with structure (5.36) becomes

$$(P_e \cdot G \cdot \begin{bmatrix} U & P & A & T \end{bmatrix} \cdot P_u) \cdot ({P_u}^T \cdot \mathbf{\Delta W}) \approx P_e \cdot G \cdot D\ . \tag{5.39}$$

Decomposing P_e and P_u

Furthermore, to get the reduced system of equations (5.37) we have to decompose P_e and P_u defined by (5.11) and (5.12) into the corresponding blocks

$$P_e = \begin{bmatrix} P_{(XY)} \\ \\ P_{(\ Z\ )} \end{bmatrix} \qquad \text{and} \qquad P_u = \begin{bmatrix} P_{(U)} & O \\ \\ O & I_{g+6} \end{bmatrix}$$

with the index $_{(X)}$ referring to the equations and unknowns they permute.

Pre-triagonalization with sub-matrices

Then (5.39) becomes

$$\begin{bmatrix} P_{(XY)} \\ \\ P_{(\ Z\ )} \end{bmatrix} \cdot \begin{bmatrix} GU & G\begin{bmatrix} P & A & T \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} P_{(U)} & O \\ \\ O & I_{g+6} \end{bmatrix} \cdot \begin{bmatrix} {P_{(U)}}^T & O \\ \\ O & I_{g+6} \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta U} \\ \\ \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \end{pmatrix}$$

$$\approx \begin{bmatrix} P_{(XY)} \\ \\ P_{(\ Z\ )} \end{bmatrix} \cdot GD$$

Block multiplication

By a block-wise multiplication we get the submatrices C, E and B and the subvectors $\mathbf{a}$ and $\mathbf{b}$.

$$\begin{bmatrix} P_{(XY)}GUP_{(U)} & P_{(XY)}G\begin{bmatrix} P & A & T \end{bmatrix} \\ \\ O_{n\times 2n} & P_{(\ Z\ )}G\begin{bmatrix} P & A & T \end{bmatrix} \end{bmatrix} \cdot \begin{pmatrix} {P_{(U)}}^T\mathbf{\Delta U} \\ \\ \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \end{pmatrix} \approx \begin{pmatrix} P_{(XY)}GD \\ \\ P_{(\ Z\ )}GD \end{pmatrix}$$

$$\iff \quad \begin{bmatrix} C & E \\ O_{n\times 2n} & B \end{bmatrix} \cdot \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \approx \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} \tag{5.40}$$

Getting $\mathbf{\Delta p}$,$\mathbf{\Delta a}$,$\mathbf{\Delta t}$

So the reduced, overdetermined system of equations $B\mathbf{y} = \mathbf{b}$ directly yields the solutions for $\mathbf{\Delta p}$, $\mathbf{\Delta a}$ and $\mathbf{\Delta t}$ decoupled from the solution for $\mathbf{\Delta U}$:

$$G_{(Z)} \cdot \begin{bmatrix} P & A & T \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \approx G_{(Z)} \cdot D \tag{5.41}$$

with

$$G_{(Z)} = P_{(Z)} \cdot G = \begin{bmatrix} \mathbf{r}_3^T(\mathbf{a_1}) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \mathbf{r}_3^T(\mathbf{a_n}) \end{bmatrix}, \tag{5.42}$$

where $\mathbf{r}_3(\mathbf{a_i})$ represents the unit normal vector at the i-th surface-point as defined by the algorithm in section 5.2.1.

Applying standard algorithms

Because this reduced, overdetermined system of equations is not sparse like the complete, overdetermined system, we can apply any *standard* algorithms to solve it without losing any performance. The sparsity can be fully exploited in the pre-triagonalization (5.13).

Equation for $\mathbf{\Delta U}$

We must still solve the equation $C\mathbf{x} = \mathbf{a} - A\mathbf{y}$ (5.38). In the actual case, $\mathbf{x}$ corresponds to $P_{(U)}^T\mathbf{\Delta U}$ and C corresponds to $P_{(XY)}GUP_{(U)}$, thus we get

$$\left(P_{(XY)}GUP_{(U)}\right) \cdot \left(P_{(U)}^T\mathbf{\Delta U}\right) = P_{(XY)}GD - P_{(XY)}G \begin{bmatrix} P & A & T \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix}. \tag{5.43}$$

Considering the special structure of C

By exploiting the special triagonalized block structure of

$$C = P_{(XY)}GUP_{(U)} = (c_{ij}),$$

we can explicitly give the expressions for the individual elements of $\mathbf{\Delta U} = (u_1,v_1 \ldots u_n,v_n)$. For $i = 1, \ldots, n$ we get

$$\Delta v_i = \frac{1}{c_{i+n,i+n}} \cdot \left(D^\star - \begin{bmatrix} P & A & T \end{bmatrix}^\star \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \right)_{i+n} \tag{5.44}$$

and

$$\Delta u_i = \frac{1}{c_{i,i}} \cdot \left(\left(D^\star - \begin{bmatrix} P & A & T \end{bmatrix}^\star \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \right)_i - c_{i,i+n} \cdot v_i \right) \tag{5.45}$$

where we have used the *transformed* matrices $D^\star$ and $\begin{bmatrix} P & A & T \end{bmatrix}^\star$:

$$D^\star = (P_{(XY)}G) \cdot D \quad \text{and} \quad \begin{bmatrix} P & A & T \end{bmatrix}^\star = (P_{(XY)}G) \cdot \begin{bmatrix} P & A & T \end{bmatrix}.$$

Proceeding like this we get the following costs: Costs for combined method

	Multiplic.		Additions	Flops
Pre-Triag.	$9(g+5)n$	$+$	$6(g+5)n$	$= (15g+75)n$
Nrm.Eq.Gener.	$\frac{(g+6)^2 n}{2}$	$+$	# mult.	$= (g^2+12g+36)n$
Nrm.Eq.Elim.	$\frac{(g+6)^3}{6}$	$+$	# mult.	$= \frac{(g+6)^3}{3}$
Backw.Subst.	$\frac{(g+6)^2}{2}$	$+$	# mult.	$= (g+6)^2$
Calculation $\mathbf{\Delta U}$	$(g+7)n + (g+8)n$	$+$	# mult.	$= (4g+30)n$
Total				$(g^2+31g+141)\cdot n$ $+\frac{(g+6)^3}{3} + (g+6)^2$

5.4.4 Further Cost-Optimizations

Alternatively we can choose another bracketing in (5.43) with which it is possible to use the *un*-transformed matrices D and $\begin{bmatrix} P & A & T \end{bmatrix}$. By doing so we can again reduce the total cost: Calculating with the original sub-matrices D,P,A,T

$$G_{(XY)} \cdot U \cdot \mathbf{\Delta U} = G_{(XY)} \cdot \left(D - \begin{bmatrix} P & A & T \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \right) \tag{5.46}$$

with

$$G_{(XY)} = P_{(XY)} \cdot G = \begin{bmatrix} \mathbf{r}_1^T(\mathbf{a_1}) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \mathbf{r}_1^T(\mathbf{a_n}) \\ \\ \mathbf{r}_2^T(\mathbf{a_1}) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \mathbf{r}_2^T(\mathbf{a_n}) \end{bmatrix}.$$

This system splits into n 2×2-sized, already triagonalized systems of equations, Splits into n 2×2-systems

$$\begin{bmatrix} \mathbf{r}_1^T(\mathbf{a_i}) \cdot \mathbf{x}_{u_{(i)}} & \mathbf{r}_1^T(\mathbf{a_i}) \cdot \mathbf{x}_{v_{(i)}} \\ 0 & \mathbf{r}_2^T(\mathbf{a_i}) \cdot \mathbf{x}_{v_{(i)}} \end{bmatrix} \cdot \begin{pmatrix} \Delta u_i \\ \Delta v_i \end{pmatrix} \tag{5.47}$$

$$= \begin{bmatrix} \mathbf{r}_1^T(\mathbf{a_i}) \\ \\ \mathbf{r}_2^T(\mathbf{a_i}) \end{bmatrix} \cdot \left(\mathbf{\Delta x}_{(i)} - \begin{bmatrix} \mathbf{x}_{p_{1_{(i)}}} \ldots \mathbf{x}_{p_{g_{(i)}}} & \mathbf{x}_{a_{(i)}} & \mathbf{x}_{b_{(i)}} & \mathbf{x}_{c_{(i)}} & I_3 \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \right)$$

which can be solved independently from each other. ($\mathbf{x}_{u_{(i)}}$, $\mathbf{x}_{v_{(i)}}$, $\mathbf{x}_{p_{j_{(i)}}}$, $\mathbf{\Delta x}_{(i)}$, $\mathbf{x}_{a_{(i)}}$, $\mathbf{x}_{b_{(i)}}$ and $\mathbf{x}_{c_{(i)}}$ are defined by (5.23), (5.24), (5.25), (5.26), (5.27), (5.28) and (5.29) and $\mathbf{r}_1$ and $\mathbf{r}_2$ by the algorithm in section 5.2.1.) Using the definitions (5.5)

and (5.7) of $\mathbf{r}_1(u_i, v_i) = \mathbf{r}_1(\mathbf{a}_i)$ and $\mathbf{r}_2(u_i, v_i) = \frac{\mathbf{r}_2^\star(u_i,v_i)}{\|\mathbf{r}_2^\star(u_i,v_i)\|} = \mathbf{r}_2(\mathbf{a}_i)$, we can express (5.47) even more explicitly as a function of (u_i, v_i) (see (5.17) and (5.18)):

$$\begin{bmatrix} \| \mathbf{x}_{u_{(i)}} \| & \mathbf{r}_1^T(u_i, v_i) \cdot \mathbf{x}_{v_{(i)}} \\ 0 & \| \mathbf{r}_2^\star(u_i, v_i) \| \end{bmatrix} \cdot \begin{pmatrix} \Delta u_i \\ \Delta v_i \end{pmatrix} \tag{5.48}$$

$$= \begin{bmatrix} \mathbf{r}_1^T(u_i, v_i) \\ \\ \mathbf{r}_2^T(u_i, v_i) \end{bmatrix} \cdot \left(\mathbf{\Delta x}_{(i)} - \begin{bmatrix} \mathbf{x}_{p_{1_{(i)}}} \dots \mathbf{x}_{p_{g_{(i)}}} & \mathbf{x}_{a_{(i)}} & \mathbf{x}_{b_{(i)}} & \mathbf{x}_{c_{(i)}} & I_3 \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} \right) .$$

In each system of equations it takes $3(g + 3)$ multiplications and $3(g + 4)$ additions to calculate the vector in the round brackets '$(\dots)$', and an additional 15 multiplications and 10 additions to calculate all scalar products involving $\mathbf{r}_1$ and $\mathbf{r}_2$, and finally 3 multiplications and 1 addition for the backward substitution.

Reduced pre-triagonalization

Forgoing a complete pre-triagonalization with matrix G, we can use (5.42) to get the solutions for $\mathbf{\Delta p}$,$\mathbf{\Delta a}$ and $\mathbf{\Delta t}$. To visualize the reduction of costs we express (5.41) explicitly:

$$\begin{bmatrix} \mathbf{r}_3^T(\mathbf{a_1}) \cdot \begin{bmatrix} \mathbf{x}_{p_{1_{(1)}}} \dots \mathbf{x}_{p_{g_{(1)}}} & \mathbf{x}_{a_{(1)}} & \mathbf{x}_{b_{(1)}} & \mathbf{x}_{c_{(1)}} & I_3 \end{bmatrix} \\ \vdots \quad \vdots \qquad\qquad\qquad \vdots \\ \mathbf{r}_3^T(\mathbf{a_n}) \cdot \begin{bmatrix} \mathbf{x}_{p_{1_{(n)}}} \dots \mathbf{x}_{p_{g_{(n)}}} & \mathbf{x}_{a_{(n)}} & \mathbf{x}_{b_{(n)}} & \mathbf{x}_{c_{(n)}} & I_3 \end{bmatrix} \end{bmatrix} \cdot \begin{pmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta a} \\ \mathbf{\Delta t} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_3^T(\mathbf{a_1}) \cdot \mathbf{\Delta x}_{(1)} \\ \vdots \\ \mathbf{r}_3^T(\mathbf{a_n}) \cdot \mathbf{\Delta x}_{(n)} \end{pmatrix} . \tag{5.49}$$

Reduced costs

We only have to calculate n times the scalar product between each of the vectors $\mathbf{x}_{p_{j_{(i)}}}$, $\mathbf{x}_{a_{(i)}}$, $\mathbf{x}_{b_{(i)}}$, $\mathbf{x}_{c_{(i)}}$ and $\mathbf{\Delta x}_{(i)}$ and the vector $\mathbf{r}_3(\mathbf{a_i})$. This step is 3 times less cost-intensive than the full pre-triagonalization and needs $3(g+4)$ multiplications and $2(g + 4)$ additions each.

Costs of this variant

We get the following costs:

	Multiplic.		Additions		Flops
Reduced Pre-T.	$3(g + 4)n$	+	$2(g + 4)n$	=	$(5g + 20)n$
Nrm.Eq.Gener.	$\frac{(g+6)^2 n}{2}$	+	# mult.	=	$(g^2 + 12g + 36)n$
Nrm.Eq.Elim.	$\frac{(g+6)^3}{6}$	+	# mult.	=	$\frac{(g+6)^3}{3}$
Backw.Subst.	$\frac{(g+6)^2}{2}$	+	# mult.	=	$(g + 6)^2$
Calculation $\mathbf{\Delta U}$	$3(g + 3)n$+$18n$	+	$3(g + 4)n$+$11n$	=	$(6g + 50)n$
Total					$(g^2 + 23g + 106) \cdot n$ $+\frac{(g+6)^3}{3} + (g + 6)^2$

Most time efficient method

Thus we have found the most time-efficient method to solve the overdetermined system of linear equations (4.22) which is required for the non-linear optimization by Gauss-Newton iteration.

Ratio between computation time and measuring time remains constant

As explained in section 5.1.2, the *memory demands* as well as the *computational costs* increase *linearly* with the number of measuring points by using any of the solution methods proposed above. All of them exploit the sparse structure. Practically this means that the ratio between measuring time and computation time remains *constant* for an *arbitrary number* of measuring points.

Coupling taken into account

The above procedure looks quite similar to (5.34) and (5.35), but in contrast to those equations, here we do not ignore the coupling between the unknowns $\mathbf{\Delta U}$ and $(\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t})$. So we also do not iterate (5.47), as we would do with (5.35) to find the actual footpoints. Equation (5.47) gives the exact dependence of $\mathbf{\Delta u_i}$ and $\mathbf{\Delta v_i}$ from $(\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t})$, carrying out an infinitesimal step with these parameters. So we derived the linearization around the actually used parameters for the correlation (4.12) which was said in section 4.3.1 not to be explicitly known.

5.5 Solution for Rank Deficient Problems

5.5.1 Problem of Rank Deficiency

Especially standard surfaces become rank deficient

When dealing with parametric surfaces it is very important to integrate a mechanism in the algorithm which correctly handles rank deficiency in the overdetermined system of equations. The problem of rank deficiency especially arises when bestfitting standard surfaces. It is caused by their rotational and translational *symmetries*. For example, if we rotate a cylinder around its axis, it still stays in the same position/orientation. The same is true for a displacement along the cylinder axis.

Symmetry causes rank deficiency

Mathematically speaking, the variation of the rotational parameter c and of the translational parameter t_z is equivalent to a variation of all surface coordinates $(u_1,v_1 \ldots u_n,v_n)$. The symmetries cause redundancy in the mathematical description. By applying any classical bestfit based on implicit representation we remove these redundancies by introducing the proper constraints.

Symmetries as trivial constraints

With the parametric representation we have introduced in the last sections, geometry is clearly separated from position/orientation. So all dimension and position/orientation parameters are explicitly accessible. In contrast to an implicit representation, they are not hidden here in a non-trivial way. The necessary constraints become very simple. In the case of the above example (cylinder with its axis along the z-axis in SPR) they are

$$c = const. \quad \text{and} \quad t_z = const. \,,$$

or equivalently

$$\frac{\partial \mathbf{X}}{\partial c} = 0 \quad \text{and} \quad \frac{\partial \mathbf{X}}{\partial t_z} = 0$$

Freezing mechanism

Interpreted in the geometrical sense, these constraints have an effect similar to

that which would be produced if the corresponding parameters were frozen. So we can introduce a 'freezing'-mechanism in our algorithm which keeps the appropriate parameters fixed (see section 6.2.1). It can be simply realized by deleting the appropriate columns $\frac{\partial \mathbf{X}}{\partial W_j}$ in the Jacobian, or, even better, by not introducing them right from the beginning.

5.5.2 How to Prevent Rank Deficiency

Symmetries should be declared

When dealing with regular surfaces we know these redundancies in advance and so they can be taken into account by declaring them – in a standardized way – together with the definition of the surface function. This is described in section 6.2.1 where we will take a closer look at the explicit implementation.

Sculptured surfaces with symmetries

Sculptured surfaces normally do not dispose of any symmetries, but theoretically it is absolutely possible that they do. Consider a standard surface approximated by a polynomial surface. Of course the polynomial surface will inherit all geometrical symmetries from the standard one. If the approximation were exact, this would result in rank deficiency which is *geometrically* but *not algebraically* evident.

Slow convergence with undeclared symmetries

In practice every approximation has some deficiencies which inhibit any rank deficiency; even so this could lead to problems in the iteration process as it would converge very slowly against a 'flat' minimum. (If the system of equations is nearly singular, the built-in damping-mechanism becomes active and/or the step length is reduced in advance by increased Levenberg-Marquardt correction-terms.)

Freezing interactively

The mechanism described in the next section also allows for the freezing of arbitrary degrees of freedom (not declared as symmetries). So, after recognizing the problem, it could be solved manually by additionally freezing the necessary rotational or translational parameters and restarting the optimization process again.

Automatic detection of rotat./ transl.-symmetries

In the case that we are not sure about a symmetry or it is desired that any manual interactions should be completely avoided, an algorithm which eliminates such unexpectedly appearing symmetries automatically proves quite useful. Based on the freezing-mechanism described above, the problem could be reduced to the mere *detection* of an eventually appearing rank deficiency.

All redundancies are moved to the reduced system of equations

However, the reduced system of equations (5.37) is of great benefit in solving this problem. To (5.37) we can apply standard solutions without losing any performance, because it is no longer sparse as the full system of equations (5.36) was. All redundancies will hide now in this system of equations because after the pre-triagonalization (5.13), the equation matrices for the unknowns (u_i, v_i) are fully decoupled from the partial system of equations (5.37). They can be resolved independently using the results of (5.37) on their right hand side only.

SVD for near rank deficiency

The appropriate remedy to the described 'nearly rank deficiency' is the method

of singular value decomposition (SVD). We perform a SVD of the matrix $B = G_{(Z)}\Big[P\,A\,T\Big]$ of (5.40):

$$B = U \cdot \Sigma \cdot V^T \quad , \qquad \Sigma = diag(\sigma_{\ 1}\ , \ldots, \sigma_{g+6})\ , \tag{5.50}$$

where $\sigma_{\ 1}\ \geq \ldots \geq \sigma_{g+6} \geq 0$ and U is a $n \times (g+6)$ orthogonal matrix and V is a $(g+6) \times (g+6)$ orthogonal matrix. The system $B \cdot \mathbf{y} \approx \mathbf{b}$ becomes

$$U \cdot \Sigma \cdot V^T \cdot \mathbf{y} \approx \mathbf{b} \qquad \text{or} \qquad \Sigma \cdot V^T \cdot \mathbf{y} \approx U^T \cdot \mathbf{b}\ . \tag{5.51}$$

Differently weighted linear combinations

By defining the linear combination of the unknown parameters $(\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t}) = \mathbf{y}$, as $z_i := \mathbf{y}^T \cdot \mathbf{cV}_i$ (where $\mathbf{cV}_i$ means i-th column of V) we express (5.51) explicitly as

$$\begin{pmatrix} \sigma_{\ 1}\ \cdot z_{\ 1} \\ \\ \vdots \\ \\ \sigma_{g+6} \cdot z_{g+6} \end{pmatrix} = U^T \cdot \mathbf{b}\ .$$

Eliminating redundancy

So we recognize, that if $\sigma_i = 0$, then z_i may be chosen arbitrarily. Thus this linear combination of dimension and pos./orient. parameters can be set to an arbitrary value (e.g. to zero), without changing anything in the corresponding equation. Since the σ_i are ordered, there is a $\sigma_r > 0$ and $\sigma_j = 0$ for $j = r+1, \ldots, g+6$. We thus have rank deficiency if $r < g+6$. In this case the solution is not unique and we compute the minimum solution corresponding to choosing $z_j = 0$ for $\sigma_j = 0$.

Solution by the pseudoinverse: $\mathbf{y} = B^+\mathbf{b}$

This solution is expressed by

$$\mathbf{y} = \underbrace{V \cdot \Sigma^+ \cdot U^T}_{B^+} \cdot \mathbf{b} \quad , \qquad \Sigma^+ = diag(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0) \tag{5.52}$$

where B^+ is the pseudoinverse of B and Σ^+ the pseudoinverse of Σ.

Small singular values

Changing the value z_i of a linear combination of the parameters ($(\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t}) = \mathbf{y}$), which is multiplied with a rather *small* singular value σ_i, does nearly not affect the right hand side. This causes slow convergence of the Gauss-Newton iteration: Approaching the resulting flat minimum, the built-in damping mechanism combined with Levenberg-Marquardt correction performs step limitation. Therefore it is recommended to choose the rank reasonably and set small singular values to zero.

Algorithm based on SVD-decomposition

So we have the following algorithm to prevent slow convergence:

1. Perform the pre-triagonalization (5.13)
2. Compute the singular value decomposition (5.50) $B = U\Sigma V^T$ of the matrix $B = G_{(Z)} \cdot \Big[\ P \quad A \quad T\ \Big]$ of the reduced overdetermined system of equations (5.41)

3. Determine the rank of the matrix B by considering singular values to be zero if they are smaller than the tolerance $\varepsilon = (g+6) \cdot \sigma_{max} \cdot \epsilon_{machine}$ (according to the recommendation in [PFTV89])

4. Compute the solution $\mathbf{y} = (\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t})$ according to (5.52) and insert the corrections $\mathbf{\Delta p}$, $\mathbf{\Delta a}$ and $\mathbf{\Delta t}$ into the system of equations (5.38) to calculate the corrections $(\Delta u_i, \Delta v_i)$ for the next step in the Gauss-Newton-iteration. Proceeding like this, the parameter corrections $\mathbf{y}$ follow the constraints

$$\mathbf{y}^T \cdot \mathbf{cV}_i = 0 \quad \text{for each } i \text{ where} \quad \sigma_i \leq \varepsilon \ .$$

5.5.3 Discussion

Always SVD?

Why we do not use SVD to avoid rank deficiency in any case? There are different reasons against doing so:

More cost-intensive

First, instead of solving normal equations, we have to perform a singular value decomposition, which is much more cost-intensive. Second, if we eliminate some parameters from the beginning, we can save computation time because we reduce the number of unknowns involved.

Distinguishing different cases

But without talking about costs, there are some other reasons too. We should distinguish the case of real rank deficiency from that of nearly rank deficiency.

Real rank deficiency

The case of *real* rank deficiency would practically appear only in the case when we already have redundancy in the definition of our surface function, for example, if we were to define the coordinates of the circle-center m_x, m_y as unknown parameters in the parametric description of the circle (additionally to the translations t_x, t_y, already defined by the general bestfit-algorithm):

$$x(t, r, m_x, m_y) = m_x + r \cdot \cos t \quad \text{and} \quad y(t, r, m_x, m_y) = m_y + r \cdot \sin t \ .$$

Another reason could be, e.g., if we did not properly declare the rotation in the xy-plane as an appearing symmetry.

Recognizing the reasons for rank deficiency

Applying SVD to these two cases, we would in fact get a correct result, however, we have to interpret the appearing redundancies afterwards. In the above example, we get an individual value for m_x and for t_x, and now we should correctly interpret that the effective center of the circle is

$$m_x^\star = m_x + t_x \quad \text{and} \quad m_y^\star = m_y + t_y \ .$$

So we recognize that we cannot bypass any further considerations in these cases.

Nearly rank deficiency

Another case is the *nearly* rank deficiency. Practically this could appear, e.g., if we try to fit an ellipse through points measured on a circle. (This is only *near* rank deficiency because a *measured* circle would never be an *ideal* circle.)

Steepening the minimum

If we delete here some small singular values, this could help to get the solution with quick convergence with a small loss of accuracy to the final result. On the

other hand, there could be some danger with this procedure, e.g., in the case that we have to determine a very *small* elliptical deviation from a circle or if we want absolutely to guarantee the comparability to any other SW.

Flat minimum: Bad measuring point distribution?

However, if we have to steepen such a flat minimum by zeroing certain singular values, we should always know *why* at all we have to do this. There is always some danger that there is something wrong with our measurement: A bad distribution of the measuring points on the surface can cause a flat minimum. From this point of view, an automatic redundancy elimination is dangerous and should at least be combined with a warning message. A *small* change of *input* data (e.g. measuring errors) could cause then a *large* variation of the *output* data (bestfit parameters).

Overcoming singular locations

As we are dealing with a nonlinear iteration process we can choose a strategy, where we decide for each step anew if we want to eliminate the small singular values or not. This could be an alternative to the damping of the iteration process or to the reduction of step size by the Levenberg-Marquardt correction. It could help in the cases where the Jacobian becomes nearly singular during the iteration process (passing a nearly local minimum or a saddle point), but becomes regular again near the final solution.

Reducing the number of independent fit-parameters

As already mentioned in section 5.5.2, SVD is mainly useful in the case that we are bestfitting a polynomial surface with rotational or translational symmetries and we do not want to treat it interactively in the way described above. Furthermore, the above algorithm is important if we change the geometry of a polynomial surface by varying its coefficients. The problem appearing here is that we have normally an *oversupply* of free parameters (all polynomial coefficients) to vary [BCF92]. Often it would be possible to fit nearly the same geometry with a highly *reduced* number of independent parameters or with lower polynomial degrees. In this case SVD (in combination with an appropriate polynomial base) will be the preferred method of reducing the degrees of freedom of the initially chosen fit model.

Chapter 6

Implementation

6.1 CM Specific Problems

6.1.1 Preliminaries

FUNKE not restricted to CM

The use of FUNKE is not restricted to the field of coordinate metrology (CM) only. It can be used for arbitrary problems, where the 'fixed parameters' of a double or a triple parametric function, dependent on one or two 'moving-parameters', have to be fitted in the least squares sense. Of course, due to the underlying concept of treating the position/orientation parameters separately and in a general manner, it becomes especially useful for geometrical fitting problems. As said in the first chapter, such problems could arise in many fields dealing with geometrical 2D/3D-spaces.

Probe radius correction and deflection compensation are CM-specific problems

However, the main application will remain in the field of CM, as it was developed regarding its special needs. Even so, the evaluation requirements of CM have some similarities to other 3D measuring techniques, for example, to surveying, but there are some special hardware problems appearing only in classical CM using mechanical probes. The two most important are *probe radius correction* and *deflection compensation* of the probe tip. In the next section we will show how they can be taken care off with FUNKE.

6.1.2 Probe Radius Correction

6.1.2.1 Difficulties

Trivial for standard elements

Probe radius correction is trivial for standard elements. It can be easily performed *after* bestfit. Dealing with standard features, the surfaces, fitted by ignoring any radius correction, are still of the same surface class as the original ones. The probe radius correction is nothing but a correction of the dimension parameters in these cases.

No general proceeding

However, each element is corrected in its own, different way. For example, for

planes and cones, probe radius correction results in a correction of position; for spheres and cylinders, probe radius correction results in a correction of the geometry parameters (radius); for toroids, probe radius correction results in a partial correction of the geometry parameters (small radius only). Because of this, the definition of an universal procedure for probe radius correction becomes difficult.

Offset surface of a sculptured surface is not trivial

For complex or sculptured surfaces, the probe radius correction becomes much more difficult than for standard elements. In these cases we cannot perform the probe radius correction *after* bestfit because the offset-surface of a non-standard element is no longer a member of the same surface class.

Same problem as exact bestfit

The problem of correct probe radius correction is related to the same underlying mathematics as the problem of exact bestfit itself. Probe radius correction *after* bestfit is restricted to elements which have an implicit distance function. On the other hand, the approach (1.9) for an approximated distance function of a sculptured surface, as proposed by [GPS80], [Goc82]:

$$d(x,y,z) \cong \frac{F(x,y,z)}{\parallel \mathbf{grad}(F(x,y,z)) \parallel}$$

explained in section 1.2.3.1 would be useless for probe radius correction.

Approach for small distances is useless for probe radius correction

It gives an approximation dealing with small deviations, but it cannot deliver satisfactory results for probe radius correction. Here we are dealing no longer with *small* 'out-of-plane' deviations where a linearization can be considered as a good approach.

Probe radius correction is possible for parametric description

So we can state that based on (1.9) we can only perform an approximate bestfit *without* any probe radius correction. In the following we will show how the exact probe radius correction can be performed based on parametric description. We simply have to modify formula (4.29) by an additional term.

6.1.2.2 Modified System of Equations

Different probe for each measuring point

To describe probe radius correction we define the vector

$$\mathbf{R} := (r_1 \cdot \mathbf{e_1}, \ldots, r_n \cdot \mathbf{e_n}) \,, \tag{6.1}$$

where r_i shall be the probe radius used at the i-th measuring point and $\mathbf{e_i}$ the i-th direction vector with unit length, pointing from the probe center to the contact point.

Replacing measuring points by contact points

Now we can replace the *measuring* point giving the probe center $\hat{\mathbf{x}}_\mathbf{i}$ by the real *contact* point

$$(\hat{\mathbf{x}}_\mathbf{i} + r_i \cdot \mathbf{e_i}) \,.$$

Effective distances

Instead of the distance d_i between the i-th *measuring* point (center of probing sphere) and the i-th actual surface point

$$d_i = \parallel \hat{\mathbf{x}}_\mathbf{i} - R(\mathbf{a}) \left(\mathbf{x_{(i)}} + \mathbf{t} \right) \parallel = \parallel R^T(\mathbf{a}) \hat{\mathbf{x}}_\mathbf{i} - \left(\mathbf{x_{(i)}} + \mathbf{t} \right) \parallel$$

with $\mathbf{x_{(i)}}$ defined by (5.22), we can write for the corrected distance $d_i^\star$ between the i-th *contact* point and the i-th actual surface point

$$d_i^\star = \parallel R^T(\mathbf{a})(\hat{\mathbf{x}}_\mathbf{i} + r_i\mathbf{e_i}) - \Big(\mathbf{x_{(i)}} + \mathbf{t}\Big) \parallel \ . \tag{6.2}$$

Choosing $\mathbf{e_i}$

Geometrically it is evident (and with some calculations it can also be shown algebraically) that we have to choose $\mathbf{e_i}$ as

$$\mathbf{e_i} = -\frac{\mathbf{\Delta x_{(i)}}}{\parallel \mathbf{\Delta x_{(i)}} \parallel} \tag{6.3}$$

with

$$\mathbf{\Delta x_{(i)}} = R^T(\mathbf{a})\hat{\mathbf{x}}_\mathbf{i} - (\mathbf{x_{(i)}} + \mathbf{t})$$

as defined in (5.26).

Modified objective function $Q_2^\star$

It can be shown then, that in doing so, we minimize instead of the original objective function

$$Q_2 = \sum_{i=1}^{n} d_i^2$$

the modified objective function

$$Q_2^\star = \sum_{i=1}^{n} d_i^{\star^2} = Q_2 + \sum_{i=1}^{n} r_i^2 - \sum_{i=1}^{n} 2r_i \parallel \mathbf{\Delta x_{(i)}} \parallel \ .$$

Same formalism

Choosing $\mathbf{e_i}$ like this, our previously developed formalism still remains true.

$\mathbf{e_i}$ approximated by the inverse actual surface normal

With iteration completed, on the one hand, probe center, contact point and actual surface point are aligned; on the other hand, the vectors between the finally calculated surface point and the measuring points (probe centers) are orthogonal to the surface. So we can approximate $\mathbf{e_i}$ also by the actual inverse surface normal. This becomes more correct the more the iteration progresses.

Normal of a parametric surface

Working with parametric representation, we can easily construct the inverse surface normal $-\mathbf{n}$ for any surface point:

$$\mathbf{e_i} = -\mathbf{n}(u,v,\mathbf{p}) = \frac{\frac{\partial \mathbf{x'}}{\partial v}(u,v,\mathbf{p}) \times \frac{\partial \mathbf{x'}}{\partial u}(u,v,\mathbf{p})}{\parallel \frac{\partial \mathbf{x'}}{\partial v}(u,v,\mathbf{p}) \times \frac{\partial \mathbf{x'}}{\partial u}(u,v,\mathbf{p}) \parallel} \ . \tag{6.4}$$

Good approximation for near-plane surfaces

For surfaces not differing much from a plane, the normal vector is nearly constant over the whole surface. So (6.4) seems a better choice than (6.3) because we have a good correction vector right from the beginning. On the other hand, we can always use (6.3) even if the actual normal vector disappears (e.g. polar singularity on the sphere). However, *after* the iteration has converged both approaches are correct, so both deliver the same (correct) result.

Corrected system of equations

To integrate now the probe radius correction into the bestfit procedure we write

instead of (4.29)

$$\left[\frac{\partial \mathbf{X}'}{\partial \mathbf{U}} \, , \, \frac{\partial \mathbf{X}'}{\partial \mathbf{p}} \, , \, Q^T(\mathbf{a}) \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}} \, , \, \frac{\partial \mathbf{T}}{\partial \mathbf{t}} \right] \cdot \mathbf{\Delta W} \tag{6.5}$$

$$\approx Q^T(\mathbf{a})\hat{\mathbf{X}} - \mathbf{X}'(\mathbf{U},\mathbf{p}) - \mathbf{T}(\mathbf{t}) + Q^T(\mathbf{a})\mathbf{R}(\mathbf{U},\mathbf{p},\mathbf{a},\mathbf{t}) \; .$$

Note: We use the original Jacobian dealing with a modified right hand side.

Thus we have a method of probe radius correction which is exact for all kinds of surfaces and is applicable always in the same way.

Universal probe radius correction

6.1.2.3 Modified Jacobian

In (6.5) we do not use the correct Jacobian that would result from including the probe radius correction. This has no effect on the final result, but the convergence is slower this way. However, since the components of the normal vectors comprise 1st derivative terms, the necessary correcting terms would contain 2nd partial derivatives with respect to the surface coordinates; hence they are negligible except at points with extreme curvature.

Uncorrected Jacobian

In our algorithm these 'extreme' points are fitted simultaneously with 'modest' points and also simultaneously with all other parameters. So, in practice, fast convergence is achieved even if there are some 'extreme' points. We have observed a stable behaviour not only in all practical examples but also in all simulated – not completely unrealistic – theoretical examples.

Good convergence behavior

In spite of this, with some additional considerations based on (6.3) we can take into account the influence of the probe radius correction on the Jacobian matrix. We adjust it accordingly, accepting a slightly increased computational effort.

Calculating the Jacobian

To this end we express the corrected distance vectors $\mathbf{\Delta x}^{\star}_{(\mathbf{i})}$ given by

Corrected distance as a function of the uncorrected

$$\mathbf{\Delta x}^{\star}_{(\mathbf{i})} \quad := \quad \underbrace{R^T(\mathbf{a})\hat{\mathbf{x}}_{\mathbf{i}} - (\mathbf{x}_{(\mathbf{i})} + \mathbf{t})}_{\mathbf{\Delta x}_{(\mathbf{i})}} + R^T(\mathbf{a}) r_i \mathbf{e}_{\mathbf{i}} \tag{6.6}$$

(c.f. (6.2)) as a vectorial function of the uncorrected distance vectors $\mathbf{\Delta x}_{(\mathbf{i})}$ defined in (5.26). The resulting Jacobian $\frac{\partial(\mathbf{\Delta x}^{\star})}{\partial(\mathbf{\Delta x})}$ we can use to transform the original Jacobian $\frac{\partial \mathbf{X}}{\partial \mathbf{W}}$ into the modified Jacobian $\frac{\partial \mathbf{X}^{\star}}{\partial \mathbf{W}}$ which takes into account the probe radius correction, too.

With

$$\mathbf{e}_{\mathbf{i}} = -\frac{\mathbf{\Delta x}_{(\mathbf{i})}}{\parallel \mathbf{\Delta x}_{(\mathbf{i})} \parallel}$$

we get

$$\mathbf{\Delta x}^{\star}_{(\mathbf{i})} = \left(I - \frac{R^T(\mathbf{a}) r_i}{\parallel \mathbf{\Delta x}_{(\mathbf{i})} \parallel} \right) \mathbf{\Delta x}_{(\mathbf{i})} \; . \tag{6.7}$$

i-th Jacobian

For the i-th Jacobian $J_{(i)} := \frac{\partial(\mathbf{\Delta x^\star_{(i)}})}{\partial(\mathbf{\Delta x_{(i)}})}$ we then get

$$J_{(i)} = \frac{R^T(\mathbf{a})r_i}{\| \mathbf{\Delta x_{(i)}} \|^3} \mathbf{\Delta x_{(i)}} \mathbf{\Delta x_{(i)}}^T + \left(I - \frac{R^T(\mathbf{a})r_i}{\| \mathbf{\Delta x_{(i)}} \|} \right) . \tag{6.8}$$

This expression can be simplified to

$$J_{(i)} = I + \frac{r_i}{d_i^3} R^T(\mathbf{a}) \begin{bmatrix} -\Delta y_{(i)}{}^2 - \Delta z_{(i)}{}^2 & \Delta x_{(i)}\Delta y_{(i)} & \Delta x_{(i)}\Delta z_{(i)} \\ \Delta x_{(i)}\Delta y_{(i)} & -\Delta x_{(i)}{}^2 - \Delta z_{(i)}{}^2 & \Delta y_{(i)}\Delta z_{(i)} \\ \Delta x_{(i)}\Delta z_{(i)} & \Delta y_{(i)}\Delta z_{(i)} & -\Delta x_{(i)}{}^2 - \Delta y_{(i)}{}^2 \end{bmatrix} . \tag{6.9}$$

Correction matrix

Thus the Jacobian matrix in (6.5) can be corrected easily by multiplying from the left hand side with the additional Jacobian matrix:

$$\frac{\partial(\mathbf{\Delta X^\star})}{\partial(\mathbf{\Delta X})} = \bigoplus_{i=1}^{n} J_{(i)} .$$

By this the modification on the *right* hand side $\mathbf{\Delta X^\star} := (\mathbf{\Delta x^\star_{(1)}}, \ldots, \mathbf{\Delta x^\star_{(n)}})$ is also taken into account on the *left* hand side.

6.1.3 Compensation of Probe Tip Deflection

6.1.3.1 Difficulties

Necessity

When performing any measurements in the μm-range, we must take into account the deflection of the probe tip. The same is true if we use extraordinarily long shafts. This could occur if a workpiece has to be measured at poorly accessible locations, e.g., at the bottom of a long bore.

Normal direction is needed

To compensate for probe tip deflection we should know the actual surface normal (direction of the probing force vector, see figure 6.1) in addition to the coordinates of the measured point. Two basically different types of probing devices have to be distinguished [Loe80]: The *switching* type issuing a single 3D coordinate upon contact with the workpiece surface and the *analogous* type issuing a 3D value, extrapolated (normally to zero-force) from continuously recorded 3D coordinates. The normal direction is only delivered by the second type of sensors.

Restrictions

However, analogous sensors have also some drawbacks compared with switching ones: They are expensive, relatively slow (if they are not in scanning mode), and they are not suited for manually operated CMM's. Furthermore, the direction of the surface normal is only accurate to within 10 - 15 degrees due to the friction effects.

Alternative methods

For these reasons, some effort was taken in the past to search for alternate methods to detect the probing direction by hardware mechanisms. [AKK89] proposes an electrically conductive sphere where the contact point is determined by an electric resistance measurement over the spherical probe surface.

Using nominal normal directions

A purely software-based concept of compensating for probe tip deflection is to insert the nominal instead of the actual surface normal. However, this approach works only when dealing with a CNC-controlled measurement process where the surface coordinates (u, v) of the contact points are known exactly enough. Evaluating the data obtained on a manually operated CMM or a scanning process on a CNC-machine poses serious problems. In both cases, the nominal surface coordinates are not known exactly enough.

Computing actual normal directions by SW

As we have seen in the last section, we are able to compute the surface normal directions in an easy way if we are using a parametric surface description. That is why FUNKE can calculate the actual normal direction and thus compensate for probe deflection more accurately than with the above mentioned conventional methods. Because the normal directions are steadily improved in the course of the iteration process, the final compensation is based on the surface normal directions of the *substitute* feature in its *bestfitted* position/orientation (instead of the *nominal* feature in its *un*fitted position/orientation). Furthermore it does not rely on any additional and problematic probing device information.

6.1.3.2 Model of Compensation

Elliptical probe model

We can compensate for the deflection of the probe tip if we assume the probe to be an ellipsoid instead of a sphere (Figure 6.1). The ellipsoid parameters and its orientation can be determined by calibration.

Virtual ellipsoid

Suppose the orientation of this virtual ellipsoid is described by 3 angles from which we construct the rotation matrix R_E; let its geometry be described by the 3 half axes h_x, h_y and h_z, from which we construct the matrix

$$E = diag(h_x, h_y, h_z) \ .$$

Description of deflection characteristics

Again, we admit different probes to allow for more generality. We describe the deflection characteristics of the probe at the i-th measuring point by R_{E_i} and E_i. Then the direction-dependent radius correction at the i-th point is

$$R_{E_i} \cdot E_i \cdot \mathbf{e_i} \ , \tag{6.10}$$

where $\mathbf{e_i}$ is the unit vector as defined in the last section, pointing from the probe sphere center to the contact point.

Introducing an anisotropic probe radius correction

Instead of an isotropic probe radius correction as discussed in the last section, we can assume now an anisotropic probe radius correction. The only modification of the algorithm for introducing probe tip deflection compensation is to replace the vector $\mathbf{R}$ of (6.1) by

$$\mathbf{R}^\star := (R_{E_1} \cdot E_1 \cdot \mathbf{e_1}, \ldots, R_{E_n} \cdot E_n \cdot \mathbf{e_n}) \ . \tag{6.11}$$

The iteration has to be performed by using this adjusted 'radius vector' $\mathbf{R}^\star$. By such simple modification, we have introduced a software-based probe tip deflection compensation in our algorithm.

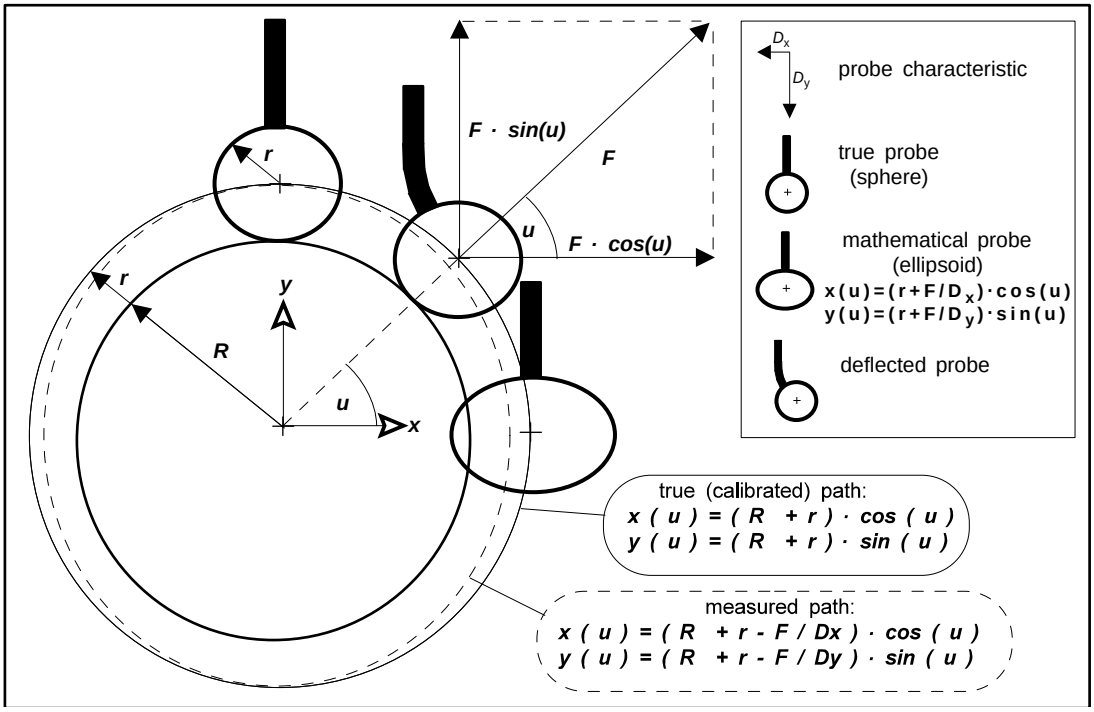


Figure 6.1: Physical model of probe tip deflection

6.2 Reduced Number of Free Parameters

6.2.1 Symmetries

Free fit-parameters have to be reduced by the number of symmetries

A lot of surfaces on industrial workpieces have rotational or translational symmetries. This introduces redundancy in the general position/orientation description as discussed in the previous sections. Symmetries reduce the number of free position/orientation parameters for parameter fitting.

Equation for the minimum number of measuring points

By the number of free parameters we can also calculate the minimum number of measuring points we need to reconstruct the object. This minimum number can be important in CM-practice. We get

$$n_{points} \geq \frac{g + t_s - s}{(d_s - d_o)} \,, \tag{6.12}$$

where g expresses the number of *independent* geometric parameters of a surface, t_s the number of space transformations (3D space: 3 rot.+3 tra.=6, 2D space: 1 rot.+2 tra.=3), s the number of all translational and rotational symmetries, d_s the space dimensions (3D space: 3, 2D space: 2) and d_o the object dimensions (surface: 2, curve: 1).

# of equations ≥ # of unknowns

We obtain the above useful equation by observing the condition:

$$\left[\begin{array}{rcl} \#\ of\ equations & \geq & \#\ of\ unknowns \\ \\ d_s \cdot n & \geq & d_o \cdot n + g + t_s - s \end{array} \right]$$

when forming our system of equations for linear bestfit.

Finding out all symmetries

We already mentioned in the preceding sections that it is important to analyze all symmetries of a given surface. Only by taking them into account can we prevent singularities in the resulting system of equations. For example, we recognize that we can rotate a sphere by x-, y- and z-rotations about its center point and it still remains the same surface in the same position. So we have to take into consideration 3 rotational symmetries whenever bestfitting a sphere by a separated position/orientation description as it is part of our algorithm.

Unmodified algorithm fails for standard surfaces

Up to now, we have described an algorithm that makes use of all degrees of freedom of position/orientation. Even though this is correct in general for a sculptured surface, it is particularly wrong for a standard surface; the algorithm would fail in the form described above. Some complex surfaces have symmetries too. The helical toroid, e.g., appearing as a technical surface in a ball screw drive, has a combined rotation/translation symmetry. It remains the same surface if we translate it and at the same time rotate it about its axis.

Symmetry operation corresponds to an u,v variation

A rotation about or a translation along a symmetry axis is equivalent to the variation of the surface coordinates u and v. Therefore the resulting system of equations becomes singular.

Contradiction to universality

For implementation we should distinguish the surface classes by their symmetries. This is, however, in some contrast to the claim of the universal applicability of the algorithm.

Symmetry vector s

To overcome this problem, we introduce for each class of surface a binary 'symmetry vector' $\mathbf{s}$ describing its symmetries:

$$\mathbf{s} := (\rho_x, \rho_y, \rho_z, \tau_x, \tau_y, \tau_z)$$

where each parameter ρ_i and τ_i assumes a value 0 or 1 (0 =symmetry, 1 =no symmetry).

Symmetry matrix S

By means of this logical vector we can construct the following 'symmetry-matrix' S (which is identical with the identity matrix I_{2n+g+6} in case there are no symmetries):

$$S := \begin{bmatrix} I_{2n+g} & O \\ \\ O & diag(\rho_x, \rho_y, \rho_z, \tau_x, \tau_y, \tau_z) \end{bmatrix} .$$

Special Jacobian

This matrix S can be interpreted as the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{W}^\star}$ of a special functional dependence $\mathbf{W}(\mathbf{W}^\star)$ between two parameter sets: The parameter set $\mathbf{W}^\star$ is identical with the parameter set $\mathbf{W}$ except for certain parameters in $\mathbf{W}^\star$ which are not variable any more.

Correcting the original system

Based on this interpretation we can correct the original, overdetermined system of equations:

$$\underbrace{J(\mathbf{W}) \cdot S}_{J(\mathbf{W}^\star)} \cdot \mathbf{\Delta W}^\star \approx \mathbf{\Delta X}(\mathbf{W}^\star) \; . \tag{6.13}$$

The changed Jacobian $J(\mathbf{W}^\star) = J(\mathbf{W}) \cdot S$ has some zero columns, hence arbitrary values for the corresponding parameters are admitted.

Decomposition $S = S^\star S^{\star T}$

Consider the special decomposition of S:

$$S = S^\star \cdot S^{\star T}$$

where $S^\star$ is a binary matrix with the same number of rows as S, but the number of columns is reduced by the number of zeros in $\mathbf{s}$.

Example of a decomposition

For example, if

$$\rho_z = 0 \quad \text{(rotational symmetry in z)} \quad ,$$

we write for $S^\star$

$$S^\star := \begin{bmatrix} I_{2n+g} & & O & & \\ & 1 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \\ O & 0 & 0 & 1 & 0 & 0 \\ & 0 & 0 & 0 & 1 & 0 \\ & 0 & 0 & 0 & 0 & 1 \end{bmatrix} .$$

Deleting redundant unknowns

This matrix $S^\star$ delivers the correction of the original system of equations in an easy way which allows us to reduce memory requirement and computation time: By multiplying the Jacobian from the right hand side with $S^\star$ we delete the corresponding columns; similarly, by multiplying the vector of unknowns from the left hand side by the transposed matrix $S^{\star T}$, we delete the corresponding parameter elements:

$$\underbrace{J(\mathbf{W}) \cdot S^\star}_{reduced\ matrix} \cdot \underbrace{S^{\star T} \cdot \mathbf{\Delta W}^\star}_{reduced\ vector} \approx \mathbf{\Delta X}(\mathbf{W}^\star) \, . \tag{6.14}$$

Symmetry vector as an additional surface-specific component

This logical vector $\mathbf{s}$ we have to define in addition to the parametric description for the complete specification of any newly introduced surface. By this, we regain the full universality of the proposed surface-independent bestfit.

Symmetries of some elements

For example, for the following common elements, we get the symmetry vectors:

$\mathbf{s}_{line}$	$=$	$(1,1,0,1,1,0)$	n_{min}	$=$	$\frac{0+6-2}{3-1}$	$=$ 2
$\mathbf{s}_{circle}$	$=$	$(1,1,0,1,1,1)$	n_{min}	$=$	$\frac{1+6-1}{3-1}$	$=$ 3
$\mathbf{s}_{plane}$	$=$	$(1,1,0,0,0,1)$	n_{min}	$=$	$\frac{0+6-3}{3-2}$	$=$ 3
$\mathbf{s}_{sphere}$	$=$	$(0,0,0,1,1,1)$	n_{min}	$=$	$\frac{1+6-3}{3-2}$	$=$ 4
$\mathbf{s}_{cylinder}$	$=$	$(1,1,0,1,1,0)$	n_{min}	$=$	$\frac{1+6-2}{3-2}$	$=$ 5
$\mathbf{s}_{cone}$	$=$	$(1,1,0,1,1,1)$	n_{min}	$=$	$\frac{1+6-1}{3-2}$	$=$ 6
$\mathbf{s}_{torus}$	$=$	$(1,1,0,1,1,1)$	n_{min}	$=$	$\frac{2+6-1}{3-2}$	$=$ 7
$\mathbf{s}_{ellipsoid}$	$=$	$(1,1,1,1,1,1)$	n_{min}	$=$	$\frac{3+6-0}{3-2}$	$=$ 9
$\mathbf{s}_{screw}$	$=$	$(1,1,1,1,1,0)$ *	n_{min}	$=$	$\frac{1+6-1}{3-2}$	$=$ 6
$\mathbf{s}_{poly.surf.}$	$=$	$(0,0,0,0,0,0)$ **	n_{min}	$=$	$\frac{g+6-6}{3-2}$	$=$ $g^{\star\star\star}$

* : One rotational symmetry is combined with a translational symmetry
** : Rotations and translation are implicitly included in polyn.coeff.(see (5.14))
***: # of independent geometry parameters of a polyn.surface: $g \leq 3st - 4$

These examples are in accordance to the following standard parametric representations (SPR) of these elements: Examples of SPR

$$\mathbf{x}'_{line}(u,v,\mathbf{p}) = (0\,,\,0\,,\,u)$$

$$\mathbf{x}'_{circle}(u,v,\mathbf{p}) = (p_1\cos(u)\,,\,p_1\sin(u)\,,\,0)$$

$$\mathbf{x}'_{plane}(u,v,\mathbf{p}) = (u\,,\,v\,,\,0)$$

$$\mathbf{x}'_{sphere}(u,v,\mathbf{p}) = (p_1\cos(u)\cos(v)\,,\,p_1\sin(u)\cos(v)\,,\,p_1\sin(v))$$

$$\mathbf{x}'_{cylinder}(u,v,\mathbf{p}) = (p_1\cos(u)\,,\,p_1\sin(u)\,,\,v)$$

$$\mathbf{x}'_{cone}(u,v,\mathbf{p}) = (v\cos(u)\,,\,v\sin(u)\,,\,v\cot(p_1))$$

$$\mathbf{x}'_{torus}(u,v,\mathbf{p}) = ((p_1+p_2\cos(v))\cos(u)\,,\,(p_1+p_2\cos(v))\sin(u)\,,\,p_2\sin(v))$$

$$\mathbf{x}'_{ellipsoid}(u,v,\mathbf{p}) = (p_1\cos(u)\cos(v)\,,\,p_2\sin(u)\cos(v)\,,\,p_3\sin(v))$$

$$\mathbf{x}'_{screw}(u,v,\mathbf{p}) = (v\cos(u)\,,\,v\sin(u)\,,\,p_1u)$$

$$\mathbf{x}'_{poly.}(u,v,\mathbf{p}) = (\textstyle\sum_{k=0}^{s}\sum_{l=0}^{t}p_{x_{kl}}u^kv^l,\ \sum_{k=0}^{s}\sum_{l=0}^{t}p_{y_{kl}}u^kv^l,\ \sum_{k=0}^{s}\sum_{l=0}^{t}p_{z_{kl}}u^kv^l)$$

6.2.2 Constraints

6.2.2.1 'Frozen' Parameters

Fixed nominal values

In coordinate metrology it is often necessary or useful to pre-assign fixed values to some parameters, normally the nominal values, varying the remaining parameters only. Sometimes such an assignment accounts for the specified functionality of the measured workpiece in a better way and sometimes it simply improves the numerical conditions for a given bestfit problem.

Example: Small cylinder segment

For example, if we have to determine the position/direction of a cylinder, which can be probed only in a small segment, it would be easier to do if its radius were preset to the nominal radius. This way, large variations of the output data (position/orientation) caused by a small spread of the input data (measuring errors!) can be avoided.

Simple constraints

In all these cases we have fitting problems subject to simple constraints. Based on the knowledge of the preceding section, we can handle these kind of constraints in the same way as the already treated symmetries.

Constraint vector $\mathbf{b}$

All unknowns appearing in $\mathbf{W}$ can be held fixed individually by simply introducing a corresponding (logical) vector $\mathbf{c}$:

$$\mathbf{c} := (\upsilon_1, \nu_1, \ldots, \upsilon_n, \nu_n, \pi_1, \ldots, \pi_g, \rho_x, \rho_y, \rho_z, \tau_x, \tau_y, \tau_z)$$

with values 0 or 1 for each element.

Constraint matrix C

This vector $\mathbf{c}$ defines the 'constraint matrix' C :

$$C := diag(\upsilon_1, \nu_1, \ldots, \upsilon_n, \nu_n, \pi_1, \ldots, \pi_g, \rho_x, \rho_y, \rho_z, \tau_x, \tau_y, \tau_z) \ .$$

Freezing matrix $\mathbf{F}$

Combining the symmetry matrix S with the constraint matrix C we get the 'freezing matrix' F :

$$F := S \cdot C \ . \tag{6.15}$$

Freezing degrees of freedom

We call this matrix a 'freezing matrix' because it allows us to 'freeze' arbitrary degrees of freedom of the bestfitted surface class. In analogy to (6.13) we can write

$$J(\mathbf{W}) \cdot F \cdot \mathbf{\Delta W}^\star \approx \mathbf{\Delta X}(\mathbf{W}^\star) \tag{6.16}$$

Decomposition $F = F^\star F^{\star T}$

and also perform the decomposition

$$F = F^\star \cdot F^{\star T} \ .$$

Reduced system

This way we get the reduced system of equations

$$\underbrace{J(\mathbf{W}) \cdot F^\star}_{reduced\ matrix} \ \cdot \ \underbrace{F^{\star T} \cdot \mathbf{\Delta W}^\star}_{reduced\ vector} \ \approx \ \mathbf{\Delta X}(\mathbf{W}^\star) \ . \tag{6.17}$$

Implementation by Boolean operations

Of course it would be very inefficient to implement (6.17) explicitly. Because S,

C and F are binary matrices the matrix multiplication (6.15) can be replaced by a logical AND between $\mathbf{s}$ and $\mathbf{c}$:

$$\mathbf{f} := ((s_1 \ AND \ c_1), \ \ldots \ , (s_{2n+g+6} \ AND \ c_{2n+g+6})) \ .$$

Then the matrix F is given by

$$F = diag(\ \mathbf{s} \ AND \ \mathbf{c}\ ) \ .$$

Deleting mechanism

This leads to the following general procedure: first construct the original matrix of the system of equations. Scan through all columns, keeping or not keeping each column depending on the corresponding value of the logical vector $\mathbf{f}$: 1 means *keep* and 0 means do *not* keep. With the solutions of the reduced linear system of equations, change the free unknowns, while leaving the frozen ones (given by $\mathbf{f}$) unchanged.

Directly setting up the reduced system

Instead of deleting a column, of course, we can also leave it away from the beginning. On the one hand this is more efficient from the point of view of computational efficiency, especially if we often make use of the possibility of freezing degrees of freedom; on the other hand, it is more complicated to implement. As we use freezing quite often (see also the next section), the latter option was chosen for the actual implementation.

6.2.2.2 General Constraints

Treating general constraints

Here we try to generalize the procedure described in the last section (i.e., freezing individual parameters) to cope with general constraints. It can be applied to cases where the constraints can be brought to a form in which a subset of parameters can be expressed as a function of the remaining ones.

Eliminating unknowns

The method is based on the concept of eliminating as many unknowns as there are constraints. When dealing with implicit functions we normally use the Lagrangian principal function to treat the constraints. This is the standard method for this type of function. Parameter elimination is usually not recommended in these cases [HH69]. On the other hand, there are some special reasons to use the elimination method for our algorithm. First, the freezing method as described above is a special case of the elimination method. So the procedure for dealing with constraints can be implemented easily for general use. Second, in a lot of practical cases, elimination comes in a natural way (see first example), and the necessary corrections are quite easy to carry out.

Advantages and drawbacks

Additionally we save CPU-time and memory, similarly to the freezing method explained above. Of course we should select the parameters to be eliminated carefully. In linear systems we should do this in such a way that the matrix K to be inverted (as described below) becomes as well-conditioned as possible.

Expressing constraints

A general constraint may be interpreted as a relation between some individual parameters extracted from the whole of all parameters to be optimized. Thus,

each constraint can be expressed by a corresponding implicit function of the parameter set $\mathbf{W}$:

$$\begin{array}{rcl} F_1(\mathbf{W}) = 0 & : & \text{1-th constraint} \\ \vdots & & \vdots \\ F_k(\mathbf{W}) = 0 & : & k\text{-th constraint} \end{array} \tag{6.18}$$

where the number of constraints k is sensibly lower than the number of optimization parameters l ($k < l$).

Explicit form

We try to express a subset of k parameters $W_1, \dots W_k$ as a function of the remaining $l - k$ parameters:

$$\begin{array}{rcl} W_1 & = & W_1(W_{k+1}, \dots, W_l) \\ \vdots & & \vdots \\ W_k & = & W_k(W_{k+1}, \dots, W_l) \end{array} . \tag{6.19}$$

If we succeed in doing this we can perform an optimization subject to the given constraints simply by eliminating these k unknowns. In practice, this is often the case; e.g. with only one constraint this is equivalent to solving the corresponding relation $F(\mathbf{W}) = 0$ with respect to a single unknown.

Completing the functional dependency

With the $k - l$-fold identity

$$\begin{array}{rcl} W_{k+1} & = & W_{k+1}(W_{k+1}, \dots, W_l) \\ \vdots & = & \vdots \\ W_l & = & W_l(W_{k+1}, \dots, W_l) \end{array}$$

we complete (6.19) to a functional dependence $\mathbf{W}(\mathbf{W}^\star)$ on the original parameter set $\mathbf{W}$ from a sub-parameter-set

$$\mathbf{W}^\star := (W_{k+1}, \dots, W_l) .$$

Jacobian of constraints

Thus, by computing the Jacobian J_{constr} of the functional dependencies (6.19) we can write the whole Jacobian as

$$\frac{\partial \mathbf{W}}{\partial \mathbf{W}^\star} = \begin{bmatrix} J_{constr} \\ I_{l-k} \end{bmatrix} . \tag{6.20}$$

Modifying the unconstrained problem

This matrix can be used to modify (4.22)

$$J(\mathbf{W}(\mathbf{W}^\star)) \cdot \underbrace{\frac{\partial \mathbf{W}}{\partial \mathbf{W}^\star}(\mathbf{W}^\star)}_{correction\ term} \cdot \boldsymbol{\Delta}\mathbf{W}^\star \approx \hat{\mathbf{X}} - \mathbf{X}(\mathbf{W}(\mathbf{W}^\star)) . \tag{6.21}$$

Recalculation by each iteration step

In the solution procedure an iteration step delivers the corrections $(\Delta W_{k+1}, \dots, \Delta W_l) = \boldsymbol{\Delta}\mathbf{W}^\star$ directly while the corrections $(\Delta W_1, \dots, \Delta W_k)$ have to be calculated from (6.19). To form then the system of equations for the following iteration step, the *complete* parameter set $(W_1, \dots, W_l) = \mathbf{W}$ has to be used again.

With some small modifications we can solve a large number of constrained bestfit problems by using the same algorithm but implementing additionally

Generalizing also constrained problems

$$\mathbf{W}(\mathbf{W}^\star) \quad \text{and} \quad \frac{\partial \mathbf{W}}{\partial \mathbf{W}^\star} \, .$$

We especially discuss the case (6.19) because in practice mostly only a few unknowns appear in the constraints. These are the parameters representing dimension, position/orientation $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$. The (much larger) vector $\mathbf{U}$, representing all surface coordinates, is not of direct interest for the solution of a bestfit problem, so it is normally not involved in the constraints.

Normally a few elements involved into constraints

6.2.2.3 Linear constraints

Consider the important special case of *linear* constraints

Expressing linear constraints

$$\sum_{i=1}^{l} b_{1i} \cdot W_i = c_1 \, , \quad \ldots \, , \quad \sum_{i=1}^{l} b_{ki} \cdot W_i = c_k \tag{6.22}$$

or

$$B \cdot \mathbf{W} = \mathbf{c} \, .$$

If we assume that the rank of the matrix B is full (otherwise there are some constraints, which are equivalent), we can always bring the constraints into form (6.19).

Linear constraints can always be brought to form (6.19)

We write

Decomposition of the matrix equation

$$\left[\; B \; \right] \cdot \mathbf{W} = \left[\; K \quad L \; \right] \cdot \mathbf{W} = \mathbf{c} \, , \tag{6.23}$$

where K and L are defined as

$$K := \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \vdots & \vdots \\ b_{k1} & \cdots & b_{kk} \end{bmatrix} \quad \text{and} \quad L := \begin{bmatrix} b_{1(k+1)} & \cdots & b_{1l} \\ \vdots & \vdots & \vdots \\ b_{k(k+1)} & \cdots & b_{kl} \end{bmatrix} \, .$$

Because B has full rank, there exists a permutation matrix P, which permutes the unknowns $\mathbf{W}$ (together with the columns of B)

Permuting the unknowns to make K nonsingular

$$\underbrace{BP}_{B'} \cdot \underbrace{P^T \mathbf{W}}_{\mathbf{W}'} = \mathbf{c}$$

such that K (as a sub-matrix of B') is nonsingular. In the following we assume that we have already permuted (6.22) in the sense above, i.e., $B := B'$ and $\mathbf{W} := \mathbf{W}'$.

Resolving to $\mathbf{W}^\star$

So we can multiply (6.23) from the left hand side by K^{-1}

$$\left[\begin{array}{cc} I_k & K^{-1}\cdot L \end{array}\right]\cdot\mathbf{W} = K^{-1}\cdot\mathbf{c} \tag{6.24}$$

and we get

$$(W_1,\ldots,W_k)^T = K^{-1}\cdot\mathbf{c} - K^{-1}\cdot L\cdot\mathbf{W}^\star\ . \tag{6.25}$$

Functional dependency $\mathbf{W}(\mathbf{W}^\star)$

Completing the system of equations as shown in the last section yields for the desired functional dependence $\mathbf{W}(\mathbf{W}^\star)$

$$\mathbf{W}(\mathbf{W}^\star) = \underbrace{\left[\begin{array}{c} -K^{-1}\cdot L \\ I_{(l-k)} \end{array}\right]}_{Jacobian\ \frac{\partial\mathbf{W}}{\partial\mathbf{W}^\star}}\cdot\mathbf{W}^\star + \left[\begin{array}{c} K^{-1}\cdot\mathbf{c} \\ 0 \\ \vdots \\ 0 \end{array}\right]\ . \tag{6.26}$$

6.2.2.4 Examples

Ellipsoid with constant ellipticity

For example take an ellipsoid which shall be fitted in such a way that the length ratio of its principal axes remains constant. We express this with the help of the two constants k_1 and k_2:

$$a:b = k_1 \qquad \text{and} \qquad b:c = k_2\ .$$

So, we get the linear system

$$\left[\begin{array}{ccc} 1 & 0 & -k_1 \\ 0 & -k_2 & 1 \end{array}\right]\cdot\left(\begin{array}{c} a \\ c \\ b \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ 0 \end{array}\right)\ .$$

With

$$K^{-1} = \left[\begin{array}{cc} 1 & 0 \\ 0 & -\frac{1}{k_2} \end{array}\right]$$

we get

$$\left[\begin{array}{c} -K^{-1}\cdot L \\ 1 \end{array}\right] = \left[\begin{array}{c} k_1 \\ \frac{1}{k_2} \\ 1 \end{array}\right]\ .$$

Thus, with the relation

$$a(b) = k_1\cdot b \qquad \text{and} \qquad c(b) = \frac{1}{k_2}\cdot b\ ,$$

together with the terms for correcting the Jacobian

$$\frac{\partial a}{\partial b} = k_1 \qquad \text{and} \qquad \frac{\partial c}{\partial b} = \frac{1}{k_2}\ ,$$

we can solve this problem. In the special case of a sphere we have $k_1 = k_2 = 1$.

Cylinder touching 2 planes

As another practical example, consider a cylinder which has to be fitted touching two given planes, as it might appear in a fillet. The direction of the cylinder axis is given by the cross product of the two plane normal vectors. To keep the example as simple as possible, we use a coordinate system with the preset cylinder-axis as the z-axis. So we have the planes given by

$$n_x{}^{(1)} \cdot x + n_y{}^{(1)} \cdot y + d^{(1)} = 0$$

$$n_x{}^{(2)} \cdot x + n_y{}^{(2)} \cdot y + d^{(2)} = 0 \ ,$$

where $(n_x^{(1)}, n_y^{(1)}, 0)$ and $(n_x^{(2)}, n_y^{(2)}, 0)$ are the normal vectors of the two planes.

Constraints

The translation vector $\mathbf{t} = (t_x, t_y)$ and the cylinder radius r are related by the two constraints:

$$n_x{}^{(1)} \cdot t_x + n_y{}^{(1)} \cdot t_y + d^{(1)} = r$$
$$n_x{}^{(2)} \cdot t_x + n_y{}^{(2)} \cdot t_y + d^{(2)} = r \ .$$

Noting in the standardized manner

According to (6.23) we write

$$\begin{bmatrix} -1 & n_x{}^{(1)} & n_y{}^{(1)} \\ -1 & n_x{}^{(2)} & n_y{}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} r \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} -d^{(1)} \\ -d^{(2)} \end{bmatrix} \ . \tag{6.27}$$

Thus, we get for K^{-1}

$$K^{-1} = \begin{bmatrix} \frac{n_x^{(2)}}{n_x^{(1)} - n_x^{(2)}} & -\frac{n_x^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ \frac{1}{n_x^{(1)} - n_x^{(2)}} & -\frac{1}{n_x^{(1)} - n_x^{(2)}} \end{bmatrix}$$

and further

$$\begin{bmatrix} -K^{-1} \cdot L \\ \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x^{(1)} n_y^{(2)} - n_x^{(2)} n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ \frac{n_y^{(2)} - n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} K^{-1} \cdot \mathbf{c} \\ \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{d_1 n_x^{(2)} - d_2 n_x^{(1)}}{n_x^{(2)} - n_x^{(1)}} \\ \frac{d_1 - d_2}{n_x^{(2)} - n_x^{(1)}} \\ 0 \end{bmatrix} \ .$$

With (the reduced)

$$\mathbf{W} = \begin{pmatrix} r \\ t_x \\ t_y \end{pmatrix} \qquad \text{and} \qquad \mathbf{W}^\star = \begin{pmatrix} t_y \end{pmatrix}$$

Functional dependencies

we get for the functional dependency

$$\mathbf{W}(\mathbf{W}^\star) = \begin{pmatrix} r(t_y) \\ t_x(t_y) \\ t_y(t_y) \end{pmatrix} = \begin{bmatrix} \frac{n_x^{(1)} n_y^{(2)} - n_x^{(2)} n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ \frac{n_y^{(2)} - n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ 1 \end{bmatrix} \cdot t_y + \begin{bmatrix} \frac{d_1 n_x^{(2)} - d_2 n_x^{(1)}}{n_x^{(2)} - n_x^{(1)}} \\ \frac{d_1 - d_2}{n_x^{(2)} - n_x^{(1)}} \\ 0 \end{bmatrix}, \qquad (6.28)$$

and for the Jacobian

$$\frac{\partial \mathbf{W}}{\partial \mathbf{W}^\star} = \begin{pmatrix} \frac{\partial r}{\partial t_y} \\ \frac{\partial t_x}{\partial t_y} \\ \frac{\partial t_y}{\partial t_y} \end{pmatrix} = \begin{bmatrix} \frac{n_x^{(1)} n_y^{(2)} - n_x^{(2)} n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ \frac{n_y^{(2)} - n_y^{(1)}}{n_x^{(1)} - n_x^{(2)}} \\ 1 \end{bmatrix}. \qquad (6.29)$$

So we supplement the system of equations with (6.29) to obtain the correction of t_y and then we use (6.28) to calculate the corrections of r and t_x.

6.3 Fitting Compound Features

6.3.1 General Procedure

Relevancy

Functional requirements of a workpiece or the conditions under which it was manufactured often lead to a special type of constrained bestfit problems. This is the problem of fitting a group of surfaces as a whole. In commercially available CM-software there exist only some few special solutions for this type of problems (see e.g. [For92],[RMM94]). By solving it, however, we also get the key to a lot of practical problems which need 'workarounds' or which cannot be solved satisfactorily today.

'Dynamic constraints'

Instead of 'static constraints' (where we have to observe a position/orientation relationship to other surfaces, which stay fixed during the bestfit), 'dynamic' constraints have to be observed here. With 'dynamic constraints', we express that the *relative* position/orientation between two or more surfaces has to be observed. These surfaces do not stay fixed during the bestfit, but are all subject to be fitted.

Sculptured surfaces are normally composed surfaces

This problems arises especially when dealing with sculptured surfaces. The mathematical description of a sculptured surface is composed of a number of separate mathematical functions (patches) each of which describes only a small part of the whole surface [KS95]. For example, the VDA-FS-standard specifies such a description [VDA82]. The surface is subdivided into individual parts for mathematical reasons only, not for geometrical ones. This means that a coherent surface normally should be constructed without any gaps and edges; in CAD-terms it should have C^0- and C^1-continuity. Fitting this multitude of 'patches' as a whole would keep these characteristics intact while an individual fit of each one would visibly destroy them.

If implicit functions are used, this problem turns out to be very hard to solve. It is an important advantage to use parametric description and to have geometry separated completely from position/orientation as realized in FUNKE.

Easy to solve with separated geometry description

It is also possible to assign to each nominal point a different surface-function. An appropriate implementation has to manage this properly: The x, y, z-coordinates and the derivatives

Assigning each nominal point a surface function

$$\mathbf{x_{(i)}} \quad , \quad \mathbf{x_{u_{(i)}}} \quad , \quad \mathbf{x_{v_{(i)}}} \quad , \quad \mathbf{x_{p1_{(i)}}} \ldots \mathbf{x_{pg_{(i)}}} \tag{6.30}$$

of the i-th surface point with surface coordinates (u_i, v_i) have to be calculated by accessing the surface function assigned to this nominal point.

When introducing the formalism for the automated generation of the Jacobian in section 4.5.2, we assumed that the values (6.30) are the function-values and the partial derivatives, respectively, of the same function

Uniform function up to now

$$\mathbf{x}'(u, v, \mathbf{p}) \ .$$

Now we can drop this assumption. When dealing with a group of N surfaces we can define a corresponding set of N surface describing functions

Split function

$$\mathbf{x}'^{(A)}(u, v, \mathbf{p}), \ldots, \mathbf{x}'^{(N)}(u, v, \mathbf{p}) \ .$$

Furthermore we need a relation

Point surface relation

$$i \mapsto K(i) \quad ; \quad (1 \leq i \leq n) \qquad \text{and} \qquad (A \leq K \leq N)$$

assigning each point i of the compound surface to exactly one surface K such we can evaluate

$$\mathbf{x_{(i)}} = \mathbf{x}'^{(K(i))}(u_i, v_i, \mathbf{p}) \qquad , \qquad \mathbf{x_{u_{(i)}}} = \frac{\partial \mathbf{x}'^{(K(i))}}{\partial u}(u_i, v_i, \mathbf{p}) \qquad \text{etc.}$$

Normally it is evident to which surface an individual point belongs. However, if the point is located very close to the boundary between two surfaces which are not visibly separated by an edge (C_1-continuity), the point should be monitored during the iteration process. If the point runs out of the boundaries of the actually assigned surface, it has to be re-assigned to another surface.

Monitoring border points

Furthermore we have to take into consideration that not every individual function $\mathbf{x}'^{(K)}$ depends on the whole vector $\mathbf{p}$, i.e., on all geometry parameters appearing in the bestfit. Let $\mathbf{p}^{(K)}$ be the geometry parameters actually appearing in the individual function $\mathbf{x}'^{(K)}$. Then we can define the set of all geometry parameters $\mathbf{p}^{(A)} \ldots \mathbf{p}^{(N)}$ as the geometry parameter vector $\mathbf{p} = (p_1, \ldots, p_g)$ of the combined feature. By an appropriate data structure we can handle these correlations accordingly, as shown in the next section.

Geometry parameters of the combined feature

6.3.2 Data Structure

6.3.2.1 Object Oriented Approach

Function address-pointer

With the help of a function-address pointer we can access the data of a certain surface point in order to calculate its coordinates and derivatives.

Virtual function

In an object oriented language this could be realized in a more elegant way by a 'virtual function', defined on a general surface class which can be 'overwritten' individually by the surface function of a sub-class like 'sphere', 'plane', etc.

Most methods can be inherited from a general surface class

Because only the geometry describing function would be class specific, we have a lot of 'methods' in the superior, general surface class which can be 'inherited' by all sub-classes derived from it. In addition to the general bestfit procedure described here, this could be, e.g., graphical representation, input/output, space transformations, etc.

N instances of the same surface class

If we are dealing with N surfaces of the same type (e.g., planes), but in different positions/orientations, we can start from the same original surface function, e.g., $\mathbf{x}'(u,v) = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix}$, and construct the N surface functions by attributing appropriate values to their position/orientation offsets:

$$\mathbf{x}'^{(A)}(u,v,\mathbf{p}) = R(\mathbf{a}^{(A)}) \cdot (\mathbf{x}'(u,v,\mathbf{p}) + \mathbf{t}^{(A)})$$

$$\vdots$$

$$\mathbf{x}'^{(N)}(u,v,\mathbf{p}) = R(\mathbf{a}^{(N)}) \cdot (\mathbf{x}'(u,v,\mathbf{p}) + \mathbf{t}^{(N)}) \ .$$

In an object oriented language, we would implement N 'instances' of the same class in this case.

Difference between $\mathbf{a}^{(K)}, \mathbf{t}^{(K)}$ and $\mathbf{a}, \mathbf{t}$

It should be noted that the position/orientation offsets

$$\mathbf{a}^{(A)}, \mathbf{t}^{(A)}, \ldots, \mathbf{a}^{(N)}, \mathbf{t}^{(N)}$$

which here define the relative (*known*) position/orientations of the N surfaces between each other are not to be confused with the position/orientation parameters

$$\mathbf{a} \quad \text{and} \quad \mathbf{t}$$

used previously as *variables* in the bestfit procedure, there defining the bestfitted (*unknown*) position/orientation of the surface

$$\mathbf{x}^{(K)}(u,v,\mathbf{p},\mathbf{a},\mathbf{t}) = R(\mathbf{a}) \cdot (\mathbf{x}'^{(K)}(u,v,\mathbf{p}) + \mathbf{t}) \quad .$$

Mechanism for compound fit

With these definitions, we get a suitable tool to fit two or more features si-

multaneously while keeping their relative position/orientation unchanged: The parameters for the unknown position/orientation

$$\underbrace{\mathbf{a} \quad / \quad \mathbf{t}}_{globally\ \ used}$$

are not stored for each surface individually, but rather an address pointer is stored for each individual surface which can point to an arbitrary set of position/ orientation parameters. This way it is possible to let different surfaces have the same physical position/orientation by simply assigning the address of a common parameter array to their individual position/orientation pointers.

Great number of possibilities

By this mechanism we can select those surfaces which we want to bestfit together (with common pos./orient. parameters $\mathbf{a}, \mathbf{t}$) and those which we want to bestfit separately (with different pos./orient. parameters $\mathbf{a}, \mathbf{t}$). Furthermore, we can determine surfaces which inherit automatically the position/orientation of other surfaces, etc. However, the only data which are always stored individually for each surface are the position/orientation offsets:

$$\underbrace{\mathbf{a}^{(K)} \quad / \quad \mathbf{t}^{(K)}}_{individually\ \ used} .$$

6.3.2.2 Recursive Structure

Hierarchical, recursive data-structure

In the actual implementation this is realized in combination with a hierarchical, recursive data-structure. It allows us to combine an arbitrary group of surfaces to a new upper element which includes all of the specified surfaces. Because this element has the same physical position/orientation as its sub-elements, by default we let their position/orientation pointers point to the same parameter set $\mathbf{a}, \mathbf{t}$. This allows us to fit each surface individually or to fit an arbitrary group of surfaces as a whole (compound feature fit).

Physically and logically duplicated data

Creating new elements by combining existing elements would cause redundancies if all data were duplicated whenever a new element were generated. The measuring points, for example, logically belong to the upper element as well as to every sub-element, but physically they exist only once. Consequently we also have to store them only once. By reserving an address-pointer instead of a complete data array for each individual surface, we can logically connect them to different surfaces in a surface hierarchy without duplicating them physically, and keeping them up-to-date with every change.

C/C++ treat pointers and arrays equivalently

This is realizable in every modern computer language, however, in C or C++, which treat pointers and arrays in a nearly equivalent way, this can be implemented quite elegantly.

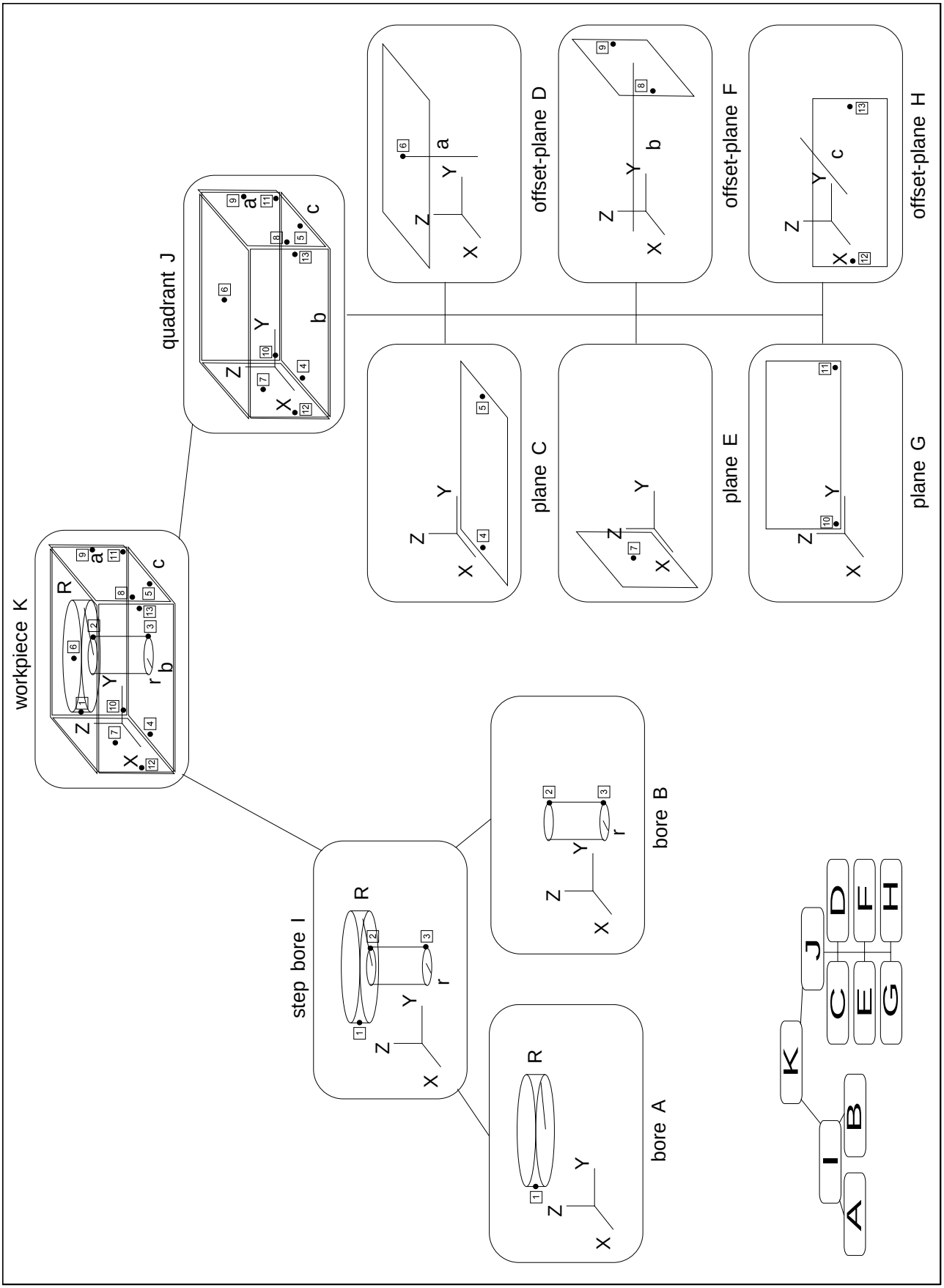


Figure 6.2: Hierarchical datastructure

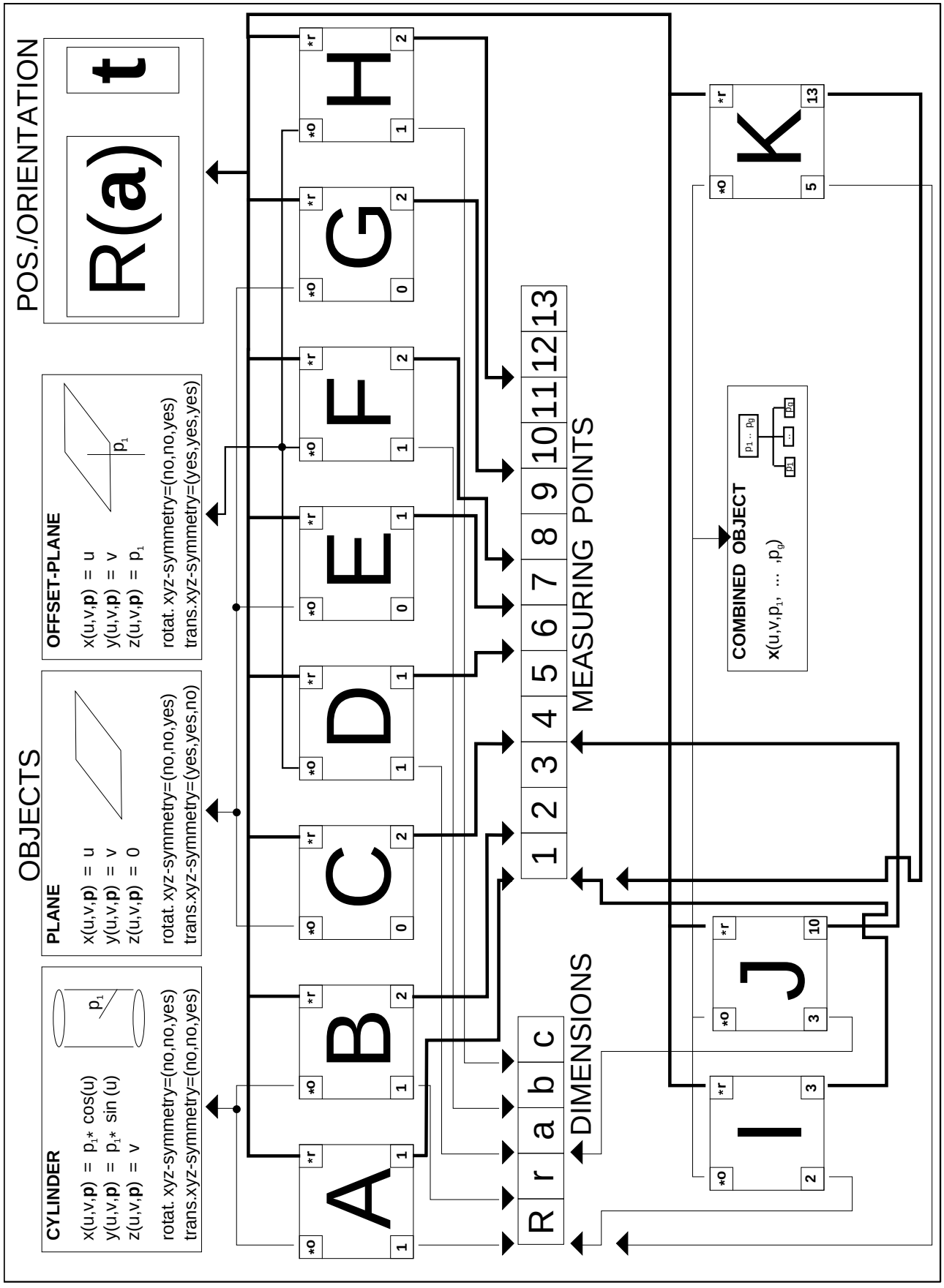


Figure 6.3: Implementation of the datastructure

6.3.2.3 Example Data Structure

Sample workpiece

As an example for the above concepts consider a workpiece consisting of a square block with a step bore inside (see figure 6.2). We assume that either the relative positions/orientations between the different surfaces are sufficiently accurate, due to the underlying manufacturing process, (for example we know, that the workpiece was manufactured in a *single* clamping and thus the machine coordinate system does not have to be re-determined), or that they can be *assumed* to be sufficiently accurate, due to the desired functionality of the workpiece. On the other hand, we are interested in 5 geometry ('dimension') parameters on the workpiece, either because we assume them to differ from their nominal values, due to the underlying manufacturing process, or because they are especially important for the desired functionality of the workpiece. These generalized 'dimensions' are the height a, width b and depth c of the square block, together with the two radii R, r of the step bore. To determine these 'dimensions' correctly, we have to bestfit position/orientation of the workpiece *simultaneously* with bestfitting its 'dimension' parameters. (In the same way we would also fit a standard feature.)

Not possible to evaluate with standard SW

With standard CM-software we can only evaluate each surface individually by taking into account its respective minimum number of measuring points (3 or 5) but we cannot fit the compound feature as a whole.

Minimum number of measuring points of the compound feature is 11

The whole workpiece has no symmetries (even though the individual surfaces have ones); Therefore we have to determine all the 6 position/orientation parameters $\mathbf{a}, \mathbf{t}$ in addition to the 5 dimension parameters. Thus the minimum number of measuring points for the compound feature is 11.

$3\times -5\times$ minimum number is recommended

Normally we would take 3 to 5 times the minimum number of measuring points. If we assume the form deviations to be random, this will improve the result by about a factor of 2 compared with the minimum number of measuring points ($\sim \sqrt{number}$). Additionally we get some information about the form deviation of the whole workpiece by taking this highly recommended number of measuring points (here, e.g., 50).

Small form deviation

A small number of measuring points (near the minimum number) we should take only then, if we are sure that the form deviation is small (in our numerical example $< 0.96 >$) and has absolutely no systematic behaviour. Any number of measuring points substantially larger than the minimum conveys information about the magnitude of the form deviation.

Well-chosen distribution

Much more important, however, is a well-chosen distribution of the measuring points, even if a large number of them is taken. This particularly concerns compound features like in the present example. Otherwise we would obtain uncertain values for the dimension parameters.

Limitation to 13 measuring points

In our example we know the behaviour of the form deviation (see above). Due to clearness while explaining graphically the underlying data structure, and also due to the easy reconstruction of the given numerical example in the following section, this is because we limit the number of measuring points in this example to 13 for determining the selected 11 parameters of the considered compound feature. As

shown in figure 6.2 it will be important to choose the measuring points on the workpiece surfaces in an appropriate way.

Suggestive decomposition

We propose the following decomposition of the workpiece: The step bore is one subpart, the square block another. For their part, they split up into two cylinders (step bore) and six planes (square block). Now we have to map this decomposition to the implemented data-structure. For any given measuring task the optimum decomposition scheme will be chosen, adapted to the actual problem.

Same structure

In the FUNKE data representation all of the individual elements have the same structure (named 'element-type'), irrespective of whether they represent cylinders, planes, offset-planes or even combined objects. The actual object, represented by one of the elements $A \dots K$, is referenced by an object pointer in the 'element'-structure (upper left hand corner of the elements $A \dots K$ in figure 6.3) which points to the object specific data (essentially surface function and surface symmetries).

Meets object oriented philosophy

This has the advantage of allowing us to treat all elements the same way. It agrees with the object oriented philosophy to integrate as many similarities as possible in a common superior class (actually this class would be represented by the class of all parametric surfaces in 3D-space $\mathbf{x}(u, v, \mathbf{p})$) while the individual subclasses (actually the class of cylinders, of planes, etc.) are intended to differ as slightly as possible from the other classes. This allows it to 'inherit' as much as possible and to 'overwrite' as little as possible.

Combined objects are equivalent to basic objects

The 'elements' themselves can be linked arbitrarily in a hierarchical structure of arbitrary depth as illustrated by the example. So we can apply in FUNKE the same methods to combined objects (step bore, square block) as we normally apply to basic objects (cylinder, plane, offset-plane). This makes it quite flexible. The flexibility is increased by the fact that we can also freeze or not freeze every bestfit parameter individually, irrespective of whether it belongs to a basic or to a combined object.

Each element should be complete

Since we can treat all elements the same way, each element should appear to the system to be complete, having its own set of measuring points (with all related data like surface coordinates, normal vectors and probe characteristics, etc.), its own set of dimension parameters and its own position/orientation.

Relative/common position/orientation

The position/orientation of an element can be split up into a common position/orientation $\mathbf{a}, \mathbf{t}$ (which can be viewed as the position/orientation of the whole workpiece), and the individual, relative position/orientation $\mathbf{a}^{(K)}$,$\mathbf{t}^{(K)}$ of the actual element X.

Referenced position/orientation

The relative position/orientation $\mathbf{a}^{(K)}$,$\mathbf{t}^{(K)}$ is thus stored individually for each element (not drawn in figure 6.3) while the common position/orientation is referenced by an address pointer to the common parameters $\mathbf{a}, \mathbf{t}$ (Upper right hand corner of the elements $A \dots K$ in figure 6.3). This subdivision into an individual (relative) and a common position/orientation is implied by the fact that the position/orientation of each individual surface changes by the same amount as that of the workpiece as a whole (or an arbitrary subpart of it) changes by a translation or a rotation.

Flexibility to determine the workpiece coordinate system

This gives us the flexibility to fit an individual surface, a group of surfaces or the workpiece as a whole. If we want to take the position/orientation of an individual surface (element $A \dots H$) or a combined element (element I, J, K) as a reference, we fit the corresponding element first and afterwards we freeze the common position/orientation $\mathbf{a}, \mathbf{t}$ while bestfitting the following elements. Thus we can determine the workpiece coordinate system by an arbitrary combination of surfaces and not only in the classical way.

Updating $\mathbf{a}^{(K)}, \mathbf{t}^{(K)}$

The offset-rotations and translations $\mathbf{a}^{(A)}, \mathbf{t}^{(A)} \dots \mathbf{a}^{(K)}, \mathbf{t}^{(K)}$ are not changed by the bestfit. Normally these parameters represent the *nominal*, pos./orient. of the elements relative to each other independent of the actually chosen workpiece coordinate system. In our example, the different instances C, E and G of the object 'plane' differ only by their offset rotations. The plane C has no offset rotation while the plane E has the offset rotation $(-\frac{\pi}{2}, 0, 0)$ and the plane G $(0, \frac{\pi}{2}, 0)$. Analogously we can prescribe the values of the offset-rotations of the elements D, F and H. Depending on the intended goal of the measurement we can keep these nominal values or overwrite them by the actually determined ones. This adds again to flexibility.

Complete, independent elements

Besides position/orientation we need a set of dimension parameters and a set of measuring points (and correlated data) to make an element appear complete to the system such that it can be treated independently. There is a problem which appears while completing logical elements (as represented by I, J and K) to independent elements, exactly treatable like the physical elements A, B, C, D, E, F, G and H: Data would be duplicated which exist physically only once.

Contiguous storage of measuring points and dimension parameters

To solve this problem, we choose the data structure shown in figure 6.3: The measuring points and the dimension parameters of all sub-elements are stored contiguously without any memory gaps between. This allows us to define an address pointer for the dimension parameter set (lower left hand corner of the elements $A \dots K$ in the figure 6.3) as well as one for the set of measuring points (lower right hand corner of these elements). This can be done for each element irrespective of whether it represents a basic object or a combined object that exists only as a logical entity.

Physical copy can be inhibited

According to this concept, any data that physically exist only once can be multi-referenced in a logical way without physical duplication. From database theory we know that the reason for not copying physically is not only the waste of memory or CPU-time, but mainly the problem of keeping coherence in the database. Imagine that each time we are updating a couple of measuring points, e.g., by performing a measurement, we would have to copy their values to all logical elements referring to some of these measuring points. With the data-structure chosen above this is absolutely unnecessary.

Referencing to the example

Of course the *number* of dimension parameters and measuring points also has to be defined accordingly. In the example shown we recognize that the element K (workpiece) has 5 geometry parameters (R, r, a, b and c) and 13 measuring points; the element I (step-bore) has 2 geometry parameters (R and r) and 3 measuring points and the element J (square block) has 3 geometry parameters (a, b and c)

and 10 measuring points. Furthermore, we recognize that the elements A, B, D, F and H (see following paragraph) have 1 geometry (dimension) parameter each and the elements A, D and E have 1 measuring point while the elements B, C, F, G and H have 2 measuring points each.

Fitting a square block

To bestfit a combined object of type 'square block' (like that represented by element J), not only observing its position/orientation $\mathbf{a}, \mathbf{t}$, but also its geometry a, b, c, we have to reference its geometry parameters a, b, c additionally on the individual planes C, D, E, F, G, H which represent the square block. But planes normally have no geometry parameters. Therefore we have to use the following procedure:

Introducing 'offset-planes'

We introduce an additional object 'offset-plane' (elements D, F and H) which has a geometry parameter p_1 representing the offset in z-direction instead of the translation parameter t_z. Because the t_z-parameter is omitted we have to introduce an additional symmetry (in z-direction) to avoid redundancy. By means of this construction we can fit the square block while observing its height, width and depth a, b, c.

Reason for introducing offset planes

This procedure is necessary because some (i.e., $t_z^{(D)}, t_z^{(F)}, t_z^{(H)}$) of the relative pos./ orient. parameters $\mathbf{a}^{(K)}, \mathbf{t}^{(K)}$ which are stored individually for each surface (see above) and thus are not subject to the general bestfit procedure, transform now to (variable) dimension parameters (a, b, c) of the compound feature.

Compound fit compared to individual fit

Note: If we determined $(t_z^{(D)}, t_z^{(F)}, t_z^{(H)})$ by fitting these three planes individually, updating each time $t_z^{(D)}$, $t_z^{(F)}$ and $t_z^{(H)}$ respectively by the actually fitted (general) t_z, this would correspond to an other solution belonging to an other measurement problem.

Generalized concept

Evidently, now we can generalize the above procedure to a commonly applicable implementation concept:

1. Check the dimension parameters of the compound feature: Which ones are real *dimension* parameters of the individual surfaces? Which ones are hidden (relative) *pos./orient.* parameters? If there are 'dimension' parameters of the second type, we eventually need new objects.

2. Create as few new objects as necessary. Is it possible to reuse the *same* object several times by creating different instances (differing by their $\mathbf{a}^{(K)}, \mathbf{t}^{(K)}$) ?

3. In the newly introduced objects define these pos./ orient. parameters (of step 1) as additional 'dimension' parameters.

4. Introduce an additional rotation/translation symmetry to each pos./orient. parameter transformed to such a 'dimension' parameter (this inhibits redundancies).

6.3.2.4 Numerical Example

Same compound feature

The same compound feature we used to explain the concept of the underlying data structure is used now as a numerical example. It shows the additional possibilities we have with the proposed method compared with classical methods.

Measuring points

In order for the computation to be reproduced, we give the data of the measuring points with same number and distribution as in the example above (see figure 6.2).

First table

The first table shows a pointwise bestfit followed by a search for the footpoints (surface points with nearest distances to the measuring points). We obtain the averaged form deviation $\sqrt{\frac{\sum_{i=1}^{n} d_i^2}{n-1}} = 16161 \mu m$. The pointwise fit as well as the footpoint search we can simulate with FUNKE by freezing the appropriate parameters: The surface coordinates $(u_1, v_1, \ldots, u_n, v_n)$ for the pointwise fit, and the position/orientation parameters $\mathbf{a}$ and $\mathbf{t}$ for the footpoint search. Because this method can only fit the nominal feature, the dimension parameters $\mathbf{p}$ are frozen, too.

Second table

We can now alternate the pointwise fit and the footpoint search as explained in section 4.3.1. We have to repeat these already iterative procedures 3219 times until their results converge to the value $44.082939 \mu m$. This is shown in the second table.

Third table

We can check the result by using FUNKE. Because we have fitted the nominal feature we must freeze the dimension parameters $\mathbf{p}$ at their nominal values in order to correctly compare the results. With 7 iteration steps we get the same value $44.082939 \mu m$ as before. This is shown in the third table.

Forth table

The main advantage of FUNKE is that we can also fit the dimension parameters together with position/orientation. By doing so we reduce the average form deviation from $44.1 \mu m$ to $0.96 \mu m$. It takes 8 iterations for the algorithm to converge to the exact value of $0.964670 \mu m$. This is shown in the forth table. The dimension parameters are slightly corrected there. An interesting effect is that one fitted radius changes its sign. This is because the parametric representation of the cylinder mathematically allows both results (neg. and pos.) while, of course, only the positive one makes sense physically. We have to take this into account for a further use of the results. Mathematically there exist two minima (see also section 7.2), however, this is no problem if we start closer to the desired one. This is absolutely realistic in CM-practice: Normally we also have approximated values for the surface coordinates (u, v) which were chosen here arbitrarily distributed (see section 3.3).

Fifth table

At last we demonstrate that we have good convergence with FUNKE, using a poor approximation also for the dimension parameters $\mathbf{p}$. FUNKE needs 9 steps to converge to the correct value $0.964670 \mu m$, starting from the given (absolutely wrong) values of all parameters. This is shown in the fifth table.

Table 1 (Example 1) : Pointwise fit followed by a search for the footpoints (nearest distances)

1.) dimensions R,r,a,b,c,translation X, translat.Y,translat.Z in [mm]
2.) rotation X,rotation Y,rotation Z in [rad]
3.) <deviation> = Sqrt((d(1)^2 + ... + d(n)^2) / (n-1)) in [micro m]

dimens. R	dimens. r	dimens. a	dimens. b	dimens. c	translat.X	translat.Y	translat.Z	rotation X	rotation Y	rotation Z	<deviation>
100.000000	50.000000	200.000000	800.000000	400.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	527189.736830
100.000000	50.000000	200.000000	800.000000	400.000000	-46.665590	-441.705600	-301.769829	-0.368466	0.248338	-0.334343	182759.889807
100.000000	50.000000	200.000000	800.000000	400.000000	-31.768324	-365.394829	-291.402729	-0.423617	0.289004	-0.334021	166737.027545
100.000000	50.000000	200.000000	800.000000	400.000000	-28.569535	-365.508938	-290.201484	-0.421029	0.308083	-0.336665	166699.491099
100.000000	50.000000	200.000000	800.000000	400.000000	-27.716104	-364.381083	-291.812000	-0.425159	0.312332	-0.337582	166696.625652
100.000000	50.000000	200.000000	800.000000	400.000000	-27.481673	-364.353534	-291.865591	-0.425222	0.314050	-0.337710	166696.383115
100.000000	50.000000	200.000000	800.000000	400.000000	-27.394644	-364.268286	-291.989703	-0.425535	0.314476	-0.337805	166696.361726
100.000000	50.000000	200.000000	800.000000	400.000000	-27.372810	-364.260714	-292.002135	-0.425560	0.314631	-0.337818	166696.359797
100.000000	50.000000	200.000000	800.000000	400.000000	-27.364744	-364.253885	-292.012188	-0.425585	0.314673	-0.337826	166696.359620
100.000000	50.000000	200.000000	800.000000	400.000000	-27.362623	-364.252882	-292.013760	-0.425588	0.314687	-0.337827	166696.359603
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361882	-364.252308	-292.014612	-0.425590	0.314691	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361677	-364.252196	-292.014784	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361608	-364.252146	-292.014858	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361589	-364.252135	-292.014876	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361582	-364.252130	-292.014883	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361582	-364.252130	-292.014883	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361582	-364.252130	-292.014883	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361581	-364.252130	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361581	-364.252130	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361581	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361581	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014884	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	166696.359602

search for the footpoints:

100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	19311.761629
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16175.019432
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.906775
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.774526
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.772779
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.772754
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.772754

Table 2 (Example 2) : Repeated pointwise fit followed by a search for the footpoints

1.) dimensions R,r,a,b,c,translation X, translat.Y,translat.Z in [mm]
2.) rotation X,rotation Y,rotation Z in [rad]
3.) <deviation> = Sqrt((d(1)^2 + ... + d(n)^2) / (n-1)) in [micro m]

dimens. R	dimens. r	dimens. a	dimens. b	dimens. c	translat.X	translat.Y	translat.Z	rotation X	rotation Y	rotation Z	<deviation>
100.000000	50.000000	200.000000	800.000000	400.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	527189.736830
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	166696.359602
100.000000	50.000000	200.000000	800.000000	400.000000	-27.361580	-364.252129	-292.014885	-0.425591	0.314693	-0.337828	16161.772758
100.000000	50.000000	200.000000	800.000000	400.000000	-14.266475	-362.881240	-289.511989	-0.436123	0.312296	-0.358471	13269.184834
100.000000	50.000000	200.000000	800.000000	400.000000	-14.266475	-362.881240	-289.511989	-0.436123	0.312296	-0.358471	11014.715472
100.000000	50.000000	200.000000	800.000000	400.000000	-6.299594	-361.561204	-288.249721	-0.444443	0.310247	-0.371211	9246.543242
100.000000	50.000000	200.000000	800.000000	400.000000	-6.299594	-361.561204	-288.249721	-0.444443	0.310247	-0.371211	7845.243614
100.000000	50.000000	200.000000	800.000000	400.000000	-1.403409	-360.409541	-287.695261	-0.451058	0.308499	-0.379167	6725.082226
100.000000	50.000000	200.000000	800.000000	400.000000	-1.403409	-360.409541	-287.695261	-0.451058	0.308499	-0.379167	5819.704263
100.000000	50.000000	200.000000	800.000000	400.000000	1.626303	-359.443699	-287.530247	-0.456332	0.307000	-0.384203	5080.610320
100.000000	50.000000	200.000000	800.000000	400.000000	1.626303	-359.443699	-287.530247	-0.456332	0.307000	-0.384203	4470.754415
100.000000	50.000000	200.000000	800.000000	400.000000	3.510435	-358.650121	-287.567606	-0.460542	0.305706	-0.387442	3962.852061
100.000000	50.000000	200.000000	800.000000	400.000000	3.510435	-358.650121	-287.567606	-0.460542	0.305706	-0.387442	3536.153921
100.000000	50.000000	200.000000	800.000000	400.000000	4.686048	-358.006504	-287.699615	-0.463901	0.304580	-0.389563	3175.209004
100.000000	50.000000	200.000000	800.000000	400.000000	4.686048	-358.006504	-287.699615	-0.463901	0.304580	-0.389563	2868.163312
100.000000	50.000000	200.000000	800.000000	400.000000	5.420268	-357.489518	-287.865914	-0.466577	0.303591	-0.390978	2605.995221
100.000000	50.000000	200.000000	800.000000	400.000000	5.420268	-357.489518	-287.865914	-0.466577	0.303591	-0.390978	2381.617778
100.000000	50.000000	200.000000	800.000000	400.000000	5.877380	-357.077583	-288.034054	-0.468704	0.302715	-0.391939	2189.447101
100.000000	50.000000	200.000000	800.000000	400.000000	5.877380	-357.077583	-288.034054	-0.468704	0.302715	-0.391939	2024.934127
100.000000	50.000000	200.000000	800.000000	400.000000	6.159009	-356.751791	-288.187789	-0.470391	0.301932	-0.392604	1884.334331
100.000000	50.000000	200.000000	800.000000	400.000000	6.159009	-356.751791	-288.187789	-0.470391	0.301932	-0.392604	1764.468490
100.000000	50.000000	200.000000	800.000000	400.000000	6.328413	-356.496071	-288.320095	-0.471723	0.301224	-0.393071	1662.607503
100.000000	50.000000	200.000000	800.000000	400.000000	6.328413	-356.496071	-288.320095	-0.471723	0.301224	-0.393071	1576.356234
100.000000	50.000000	200.000000	800.000000	400.000000	6.425238	-356.297024	-288.429036	-0.472769	0.300577	-0.393402	1503.602225
100.000000	50.000000	200.000000	800.000000	400.000000	6.425238	-356.297024	-288.429036	-0.472769	0.300577	-0.393402	1442.463883
100.000000	50.000000	200.000000	800.000000	400.000000	6.474539	-356.143626	-288.515349	-0.473586	0.299981	-0.393637	1391.270490
100.000000	50.000000	200.000000	800.000000	400.000000	6.474539	-356.143626	-288.515349	-0.473586	0.299981	-0.393637	1348.539195
100.000000	50.000000	200.000000	800.000000	400.000000	6.492293	-356.026895	-288.581050	-0.474217	0.299427	-0.393803	1312.965244
100.000000	50.000000	200.000000	800.000000	400.000000	6.492293	-356.026895	-288.581050	-0.474217	0.299427	-0.393803	1283.407516
100.000000	50.000000	200.000000	800.000000	400.000000	6.488802	-355.939558	-288.628649	-0.474700	0.298907	-0.393918	1258.878459
100.000000	50.000000	200.000000	800.000000	400.000000	6.488802	-355.939558	-288.628649	-0.474700	0.298907	-0.393918	1238.529974
100.000000	50.000000	200.000000	800.000000	400.000000	6.470785	-355.875755	-288.660722	-0.475063	0.298414	-0.393996	1221.640876
100.000000	50.000000	200.000000	800.000000	400.000000	6.470785	-355.875755	-288.660722	-0.475063	0.298414	-0.393996	1207.602223
... After 3219 iterations :											
100.000000	50.000000	200.000000	800.000000	400.000000	-9.693915	-364.914341	-275.494999	-0.438539	0.185613	-0.377517	44.082939
100.000000	50.000000	200.000000	800.000000	400.000000	-9.693914	-364.914341	-275.494999	-0.438539	0.185613	-0.377517	44.082939

Table 3 (Example 3) : FUNKE bestfit with 'freezed' dimension parameters

1.) dimensions R,r,a,b,c,translation X, translat.Y,translat.Z in [mm]
2.) rotation X,rotation Y,rotation Z in [rad]
3.) <deviation> = Sqrt((d(1)^2 + ... + d(n)^2) / (n-1)) in [micro m]

dimens. R	dimens. r	dimens. a	dimens. b	dimens. c	translat.X	translat.Y	translat.Z	rotation X	rotation Y	rotation Z	<deviation>
100.000000	50.000000	200.000000	800.000000	400.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	527189.736830
100.000000	50.000000	200.000000	800.000000	400.000000	-26.962169	-452.706995	-316.023314	-0.624417	-0.122582	-0.498455	171685.214823
100.000000	50.000000	200.000000	800.000000	400.000000	-45.630362	-388.413334	-255.988059	-0.400808	0.041846	-0.366483	30283.388745
100.000000	50.000000	200.000000	800.000000	400.000000	-20.392948	-374.580907	-270.184962	-0.415833	0.128853	-0.371635	4722.388937
100.000000	50.000000	200.000000	800.000000	400.000000	-11.059554	-365.826426	-274.687535	-0.435105	0.176279	-0.375801	533.752739
100.000000	50.000000	200.000000	800.000000	400.000000	-9.740600	-364.940877	-275.460415	-0.438423	0.185299	-0.377460	46.151206
100.000000	50.000000	200.000000	800.000000	400.000000	-9.694078	-364.914438	-275.494855	-0.438539	0.185612	-0.377516	44.082946
100.000000	50.000000	200.000000	800.000000	400.000000	-9.693915	-364.914341	-275.494999	-0.438539	0.185613	-0.377517	44.082939
100.000000	50.000000	200.000000	800.000000	400.000000	-9.693914	-364.914341	-275.494999	-0.438539	0.185613	-0.377517	44.082939

Table 4 (Example 4) : FUNKE bestfit with good starting values for the dimension parameters

1.) dimensions R,r,a,b,c,translation X, translat.Y,translat.Z in [mm]
2.) rotation X,rotation Y,rotation Z in [rad]
3.) <deviation> = Sqrt((d(1)^2 + ... + d(n)^2) / (n-1)) in [micro m]

dimens. R	dimens. r	dimens. a	dimens. b	dimens. c	translat.X	translat.Y	translat.Z	rotation X	rotation Y	rotation Z	<deviation>
100.000000	50.000000	200.000000	800.000000	400.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	527189.736830
-20.065216	-14.562427	187.115190	731.479803	360.791347	-1.038748	-407.026995	-266.761233	-0.624418	-0.347418	-0.498455	184688.743146
95.208182	-45.973801	155.282514	823.750745	268.329153	20.138940	-446.973310	-165.445432	-0.104923	-0.478284	-0.412981	95729.572108
103.474889	-51.500345	206.847157	810.140663	372.144427	-33.068164	-426.564142	-213.573856	-0.310157	-0.154191	-0.342600	56994.316213
80.156201	-40.348628	226.690949	796.749513	450.664729	-34.558360	-363.022860	-306.592444	-0.472547	0.296814	-0.360566	31906.281426
99.726790	-49.221744	202.973734	799.161943	401.500185	-17.896774	-368.172489	-272.286239	-0.432052	0.156420	-0.370856	5239.178770
99.813552	-50.058359	200.290259	800.067822	400.111014	-10.269841	-365.303144	-275.159683	-0.437049	0.181577	-0.376859	205.220945
99.856704	-50.154012	200.098050	800.104551	399.919636	-9.841929	-365.132103	-275.326918	-0.437967	0.183980	-0.377291	1.503619
99.856870	-50.154131	200.097516	800.104574	399.918752	-9.840719	-365.131728	-275.326592	-0.437969	0.183985	-0.377292	0.964670
99.856870	-50.154131	200.097516	800.104574	399.918752	-9.840720	-365.131729	-275.326591	-0.437969	0.183985	-0.377292	0.964670

Table 5 (Example 5) : FUNKE bestfit with poor starting values for the dimension parameters

1.) dimensions R,r,a,b,c,translation X, translat.Y,translat.Z in [mm]
2.) rotation X,rotation Y,rotation Z in [rad]
3.) <deviation> = Sqrt((d(1)^2 + ... + d(n)^2) / (n-1)) in [micro m]

dimens. R	dimens. r	dimens. a	dimens. b	dimens. c	translat.X	translat.Y	translat.Z	rotation X	rotation Y	rotation Z	<deviation>
29.000000	180.000000	-410.000000	620.000000	850.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	594316.089763
-20.049520	-14.577652	187.113973	731.480363	360.791810	-1.008625	-407.027417	-266.822259	-0.624416	-0.347108	-0.498453	303072.087744
-63.015165	119.305386	40.643918	748.687737	721.603062	-90.691699	-340.333855	-391.900708	-0.864494	0.403745	-0.365649	296115.194310
-48.917810	26.661958	297.978063	782.825685	359.481912	-143.953175	-503.485998	-277.707882	-0.281067	-0.311984	-0.270679	205323.215223
-113.729066	57.582421	176.752762	794.703638	394.123905	-44.588745	-388.964187	-225.339851	-0.315758	-0.155836	-0.349026	23508.765484
-83.865977	48.000531	200.408805	799.605880	399.750621	-28.811233	-375.202443	-261.710803	-0.399977	0.060549	-0.356239	8275.015153
-97.063565	49.712234	202.156064	800.052896	401.595331	-12.883469	-366.212359	-275.337671	-0.432931	0.172987	-0.374212	2468.015527
-99.708094	50.127929	200.113774	800.101908	399.926072	-9.882995	-365.155458	-275.312879	-0.437842	0.183714	-0.377236	74.651874
-99.856749	50.154130	200.097519	800.104573	399.918757	-9.840741	-365.131738	-275.326587	-0.437969	0.183985	-0.377292	0.967399
-99.856870	50.154131	200.097516	800.104574	399.918752	-9.840720	-365.131729	-275.326591	-0.437969	0.183985	-0.377292	0.964670
-99.856870	50.154131	200.097516	800.104574	399.918752	-9.840720	-365.131729	-275.326591	-0.437969	0.183985	-0.377292	0.964670

Table 6 : (X,Y,Z)-coordinates of the measuring points

coordinate	X	Y	Z	element
measuring point 1	31.6266	-118.2398	25.7440	A
measuring point 2	-6.2994	40.9176	18.6509	B
measuring point 3	11.0747	77.4910	-58.1322	B
measuring point 4	129.4105	-289.6457	-211.3621	C
measuring point 5	-87.6900	338.9653	38.4063	C
measuring point 6	-9.9867	-42.1593	100.6373	D
measuring point 7	138.3013	-369.7778	-102.1767	E
measuring point 8	43.1076	404.7678	172.0516	F
measuring point 9	-272.9729	318.6363	53.5095	F
measuring point 10	-41.6399	-361.2241	-184.9495	G
measuring point 11	-289.7178	287.9307	-1.2912	G
measuring point 12	316.9116	-271.4253	-27.8910	H
measuring point 13	73.3131	354.0138	162.5525	H

Chapter 7

Verification

7.1 General Remarks

Significance

The verification of an algorithm is always an important step. In metrology it is crucial [Lot83], [Dri89], [Kna93]. Results of different algorithms should be comparable, i.e., they have not only to be somehow reasonable, but they have to be exact in a precisely defined way [DIN86], [Bri89]. Metrology often appears in connection with quality control. This crucial responsibility for product quality additionally increases its significance.

State-of-the-art sculptured surface fit

Contrary to this, the algorithms for sculptured and complex surfaces available today are usually not capable of delivering the exact least squares solutions. So we can test them only empirically and check if their results are reasonable and useful in practice.

Comparison is always possible

When dealing with standard surfaces as well as with sculptured or complex surfaces, however, we can *compare* two algorithms in a very objective way (even if they only approximate the bestfit).

'Quality rate' of an algorithm

As a prerequisite it has to be decided by which norm (e.g., Gauss or Chebychev) the quality of the results of the compared algorithms has to be measured. As the measure for the quality of the results, we take the value of the objective function derived from the selected bestfit criterion. Of course we have to calculate this value based on the exact distance function of the bestfitted surface. To be sure, we should never rely on the calculated distances and even less on the distance function used by the algorithm itself. The algorithm under consideration might use an approximated distance function for simplicity or for the lack of an explicit one.

Vectorial description of standard elements

When dealing with standard elements where correct distance functions exist, it is useful to define them as a function of the coordinates $(\hat{x}, \hat{y}, \hat{z})$ of an arbitrary measuring point $\hat{\mathbf{x}}$ together with the output obtained by the algorithm to be tested. These output parameters are the dimensions $\mathbf{p}$ and position/orientation $\mathbf{a}$ and $\mathbf{t}$. For standard elements, position/orientation are mostly given by a position

and a direction vector [Wir93]. Because all standard elements have at least one rotational symmetry it is sufficient to consider *one* direction vector only.

In the following we use a *normalized* direction vector $\mathbf{n}$. The position vector $\mathbf{r}$ defines directly the three possible degrees of freedom of translation t_x, t_y, t_z, while the direction vector $\mathbf{n}$ includes indirectly the remaining two rotational degrees of freedom a and b:

Correlation between $\mathbf{r}, \mathbf{n}$ and $\mathbf{t}, \mathbf{a}$

$$\mathbf{n} = \begin{pmatrix} -\sin(b) \\ \sin(a)\cos(b) \\ \cos(a)\cos(b) \end{pmatrix} . \tag{7.1}$$

Thus the preferred form of a distance function is:

Distance as a function of $\hat{\mathbf{x}}, \mathbf{p}, \mathbf{r}, \mathbf{n}$

$$d(\hat{\mathbf{x}}, \mathbf{r}, \mathbf{n}, \mathbf{p}) . \tag{7.2}$$

In this function we insert the parameters $\mathbf{r}, \mathbf{n}$ and $\mathbf{p}$ delivered as a solution of the algorithm to be tested together with a measuring point $\hat{\mathbf{x}}$. This way we obtain directly the distance between the measuring point and the surface as calculated by this algorithm. Doing this n times, for all given measuring points, we can calculate the actual value of the selected objective function (e.g. Gauss) as a function of the output parameters of the algorithm to be tested:

Evaluating value of objective function

$$Q_2(\mathbf{r}, \mathbf{n}, \mathbf{p}) = \sum_{i=1}^{n} d^2(\hat{\mathbf{x}}_\mathbf{i}, \mathbf{r}, \mathbf{n}, \mathbf{p}) . \tag{7.3}$$

7.2 Distance Calculation for Standard Surfaces

In the following we derive a method for computing the distance function of any (rotation symmetric) standard element in form (7.2):

Method to compute the distance function of a standard element

Such a distance function can be used for testing and comparing arbitrary bestfit implementations (like FUNKE or also conventional solutions).

Serves for both: bestfit implementation and testing

The basic idea is to express the distance function in a well-chosen coordinate system where it is easy to define. Afterwards a coordinate transformation is performed, expressed only by means of the position-vector $\mathbf{r}$ and the direction-vector $\mathbf{n}$. Applying this procedure, we get, e.g., the well known 'Hessian normal form' for a plane.

Algorithm: Parameter and coordinate transformation

1. Describe the distance to the desired surface in a principal axes system of cylinder coordinates (ρ, z). By this we obtain rather simple distance functions:

   Examples:

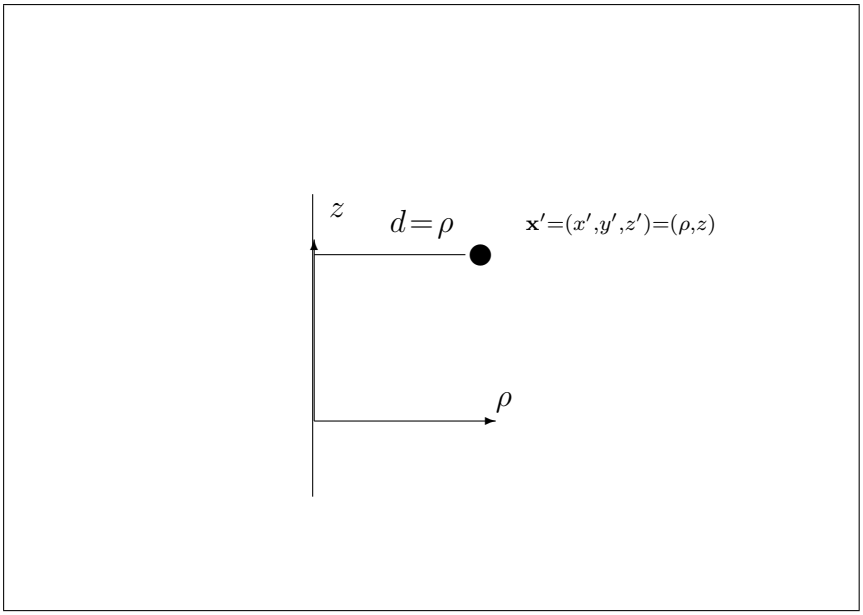


Figure 7.1. Straight line (coincident with the z-axis):

implicit surface function: $\rho = 0$
distance function: $d(\rho, z) = \rho$

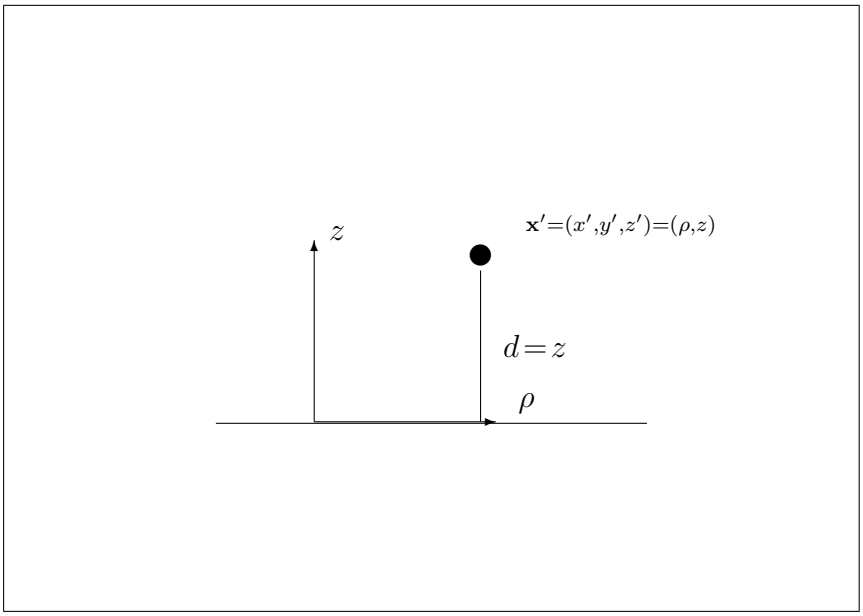


Figure 7.2. Plane (normal to the z-axis):

implicit surface function: $z = 0$
distance function: $d(\rho, z) = z$

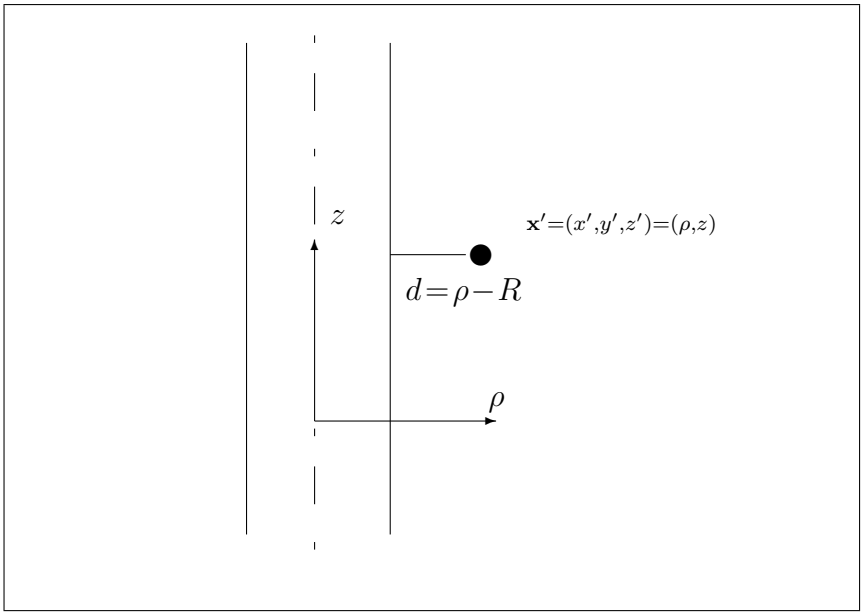


Figure 7.3. Cylinder (axis coincident with the z-axis) with radius R:

implicit surface function: $\rho = R$
distance function: $d(\rho, z) = \rho - R$

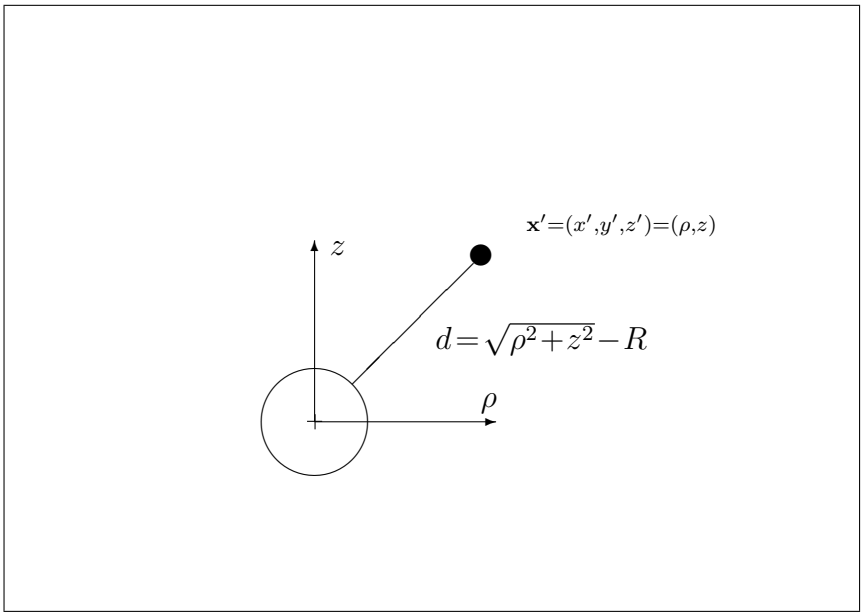


Figure 7.4. Sphere (center at the origin) with radius R:

implicit surface function: $\rho^2 + z^2 = R^2$
distance function: $d(\rho, z) = \sqrt{\rho^2 + z^2} - R$

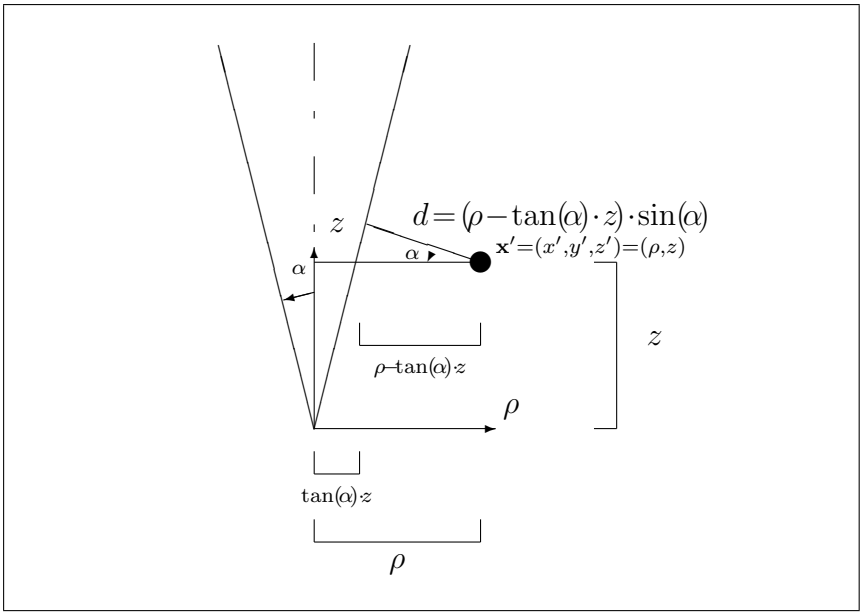


Figure 7.5. Cone (axis coincident with the z-axis) with angle α:

implicit surface function: $\frac{\rho}{z} = \tan(\alpha)$
distance function: $d(\rho, z) = (\rho - \tan(\alpha)z) \cdot \sin(\alpha)$

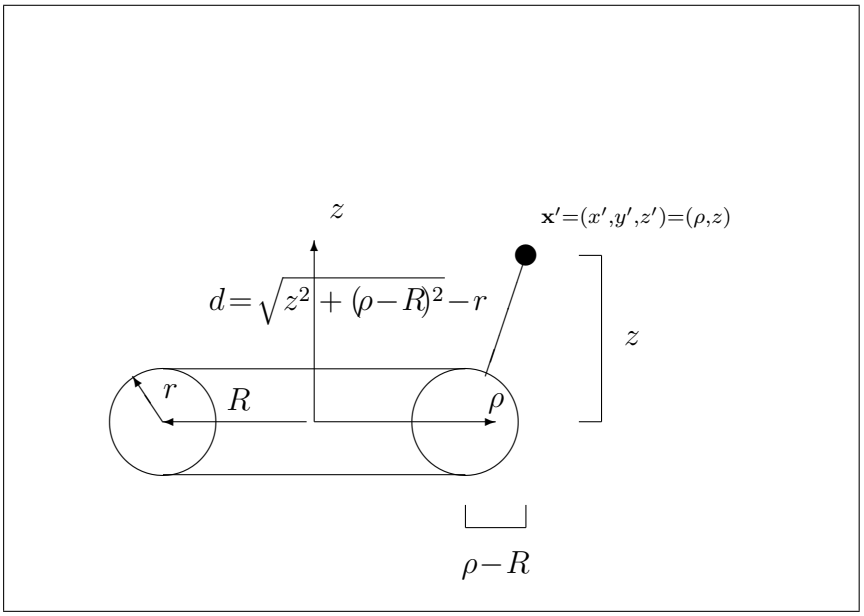


Figure 7.6. Toroid (normal to the z-axis) with large/small radius R/r:

implicit surface function: $(\rho - R)^2 + z^2 = r^2$
distance function: $d(\rho, z) = \sqrt{z^2 + (\rho - R)^2} - r$

2. Additionally define Cartesian coordinates in the selected principal axes coordinate system

$$\mathbf{x}' := (x', y', z') \qquad \text{with} \qquad \rho^2 = x'^2 + y'^2 \qquad \text{and} \qquad z = z'$$

3. Describe the translation from a general cartesian coordinate system to this principal axes system by the position vector of the actual element

$$\mathbf{x}' = \mathbf{x} - \mathbf{r}$$

4. Express the z coordinate by the direction vector $\mathbf{n}$

$$z = (\mathbf{n}, \mathbf{x}')$$

5. Express the coordinate ρ by the two already calculated parameters $\mathbf{x}'$ and z

$$\rho = \sqrt{\|\mathbf{x}'\|^2 - z^2}$$

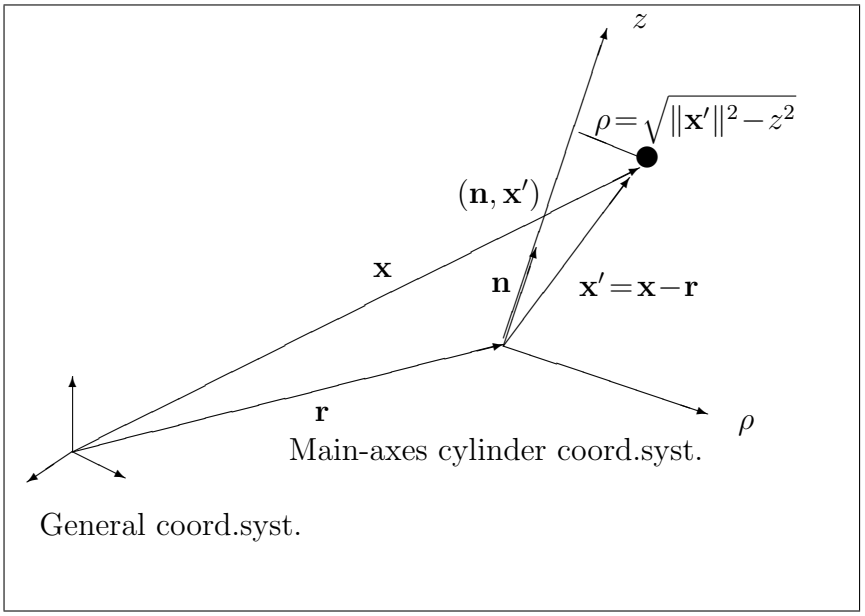


Figure 7.7. Coordinate system transformation

In this way we obtain the general distance function in the form (7.2) by substituting z and ρ of the simple distance function (of step 1.) by $\mathbf{x} = (x, y, z)$. Substituting the coordinates

This function can serve as a test for the output of a given algorithm (dimension parameters $\mathbf{p}$, position and direction vector $\mathbf{r}/\mathbf{n}$). By inserting any measuring points $\hat{\mathbf{x}}$ together with the above output parameters we obtain the remaining distances based on the bestfit solution being tested. Bestfit-test

Examples:

Plane: Distance function

Plane:

$$d(\rho, z) = z = (\mathbf{n}, \mathbf{x}') = (\mathbf{n}, \mathbf{x} - \mathbf{r}) = (\mathbf{n}, \mathbf{x}) - (\mathbf{n}, \mathbf{r})$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}) = n_x x + n_y y + n_z z + \underbrace{D}_{-(\mathbf{n},\mathbf{r})} \tag{7.4}$$

(7.4) is the well-known 'Hessian normal form' of the plane, derived here in an unconventional way.

Cylinder: Distance function

Cylinder:

$$d(\rho, z, R) = \rho - R$$

$$= \sqrt{\|\mathbf{x}'\|^2 - z^2} - R = \sqrt{\|\mathbf{x} - \mathbf{r}\|^2 - (\mathbf{n}, \mathbf{x} - \mathbf{r})^2} - R$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}, R) =$$

$$\sqrt{(x - r_x)^2 + (y - r_y)^2 + (z - r_z)^2 - (n_x x + n_y y + n_z z)^2} - R \qquad \text{if} \qquad \mathbf{r} \perp \mathbf{n} \tag{7.5}$$

Straight line: Distance function

Straight line:

$$d(\rho, z) = \rho = \sqrt{\|\mathbf{x}'\|^2 - z^2} = \sqrt{\|\mathbf{x} - \mathbf{r}\|^2 - (\mathbf{n}, \mathbf{x} - \mathbf{r})^2}$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}) =$$

$$\sqrt{(x - r_x)^2 + (y - r_y)^2 + (z - r_z)^2 - (n_x x + n_y y + n_z z)^2} \qquad \text{if} \qquad \mathbf{r} \perp \mathbf{n} \tag{7.6}$$

Simplification by $\mathbf{r} \perp \mathbf{n}$

In (7.5) and (7.6) we have taken advantage of the fact that – due to the translational symmetry of the straight line and of the cylinder – we are free to choose the position vector $\mathbf{r}$. The only condition is that it points to the rotation axis. The choice $\mathbf{r} \perp \mathbf{n}$ simplifies the above formulas. If dealing with software that does not calculate this special solution, we have to resort to the general formula. However, using the above distance functions for a (conventional) bestfit implementation, we can use the simplified formulas (7.5) and (7.6) because we have anyway to perform the minimization, subject to $\mathbf{r} \perp \mathbf{n}$ (or a similar constraint), as otherwise we would get a singular system of equations.

Sphere: Distance function

Sphere:

$$d(\rho, z, R) = \sqrt{\rho^2 + z^2} - R$$

$$= \sqrt{\|\mathbf{x}'\|^2 - z^2 + z^2} - R = \|\mathbf{x}'\| - R = \|\mathbf{x} - \mathbf{r}\| - R \tag{7.7}$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}, R) = \sqrt{(x - r_x)^2 + (y - r_y)^2 + (z - r_z)^2} - R \tag{7.8}$$

(7.8) represents the well-known equation for a sphere but normalized by the $\sqrt{\dots}$ operation as discussed in section 1.2.3. So it can not only describe the surface itself but also express the geometrical distance between an arbitrary 3D-point and the sphere surface. Normalized sphere function

Cone: Cone: Distance function

$$d(\rho, z, \alpha) = (\rho - \tan(\alpha)z)\sin(\alpha)$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}, \alpha)$$

$$= (\sqrt{\|\mathbf{x}-\mathbf{r}\|^2 - (\mathbf{n}, \mathbf{x}-\mathbf{r})^2} - \tan(\alpha)(\mathbf{n}, \mathbf{x}-\mathbf{r}))\sin(\alpha) \tag{7.9}$$

Toroid: Toroid: Distance function

$$d(\rho, z, R, r) = \sqrt{z^2 + (\rho - R)^2} - r$$

$$\Longrightarrow d(\mathbf{x}, \mathbf{r}, \mathbf{n}, R, r)$$

$$= \sqrt{(\mathbf{n}, \mathbf{x}-\mathbf{r})^2 + (\sqrt{\|\mathbf{x}-\mathbf{r}\|^2 - (\mathbf{n}, \mathbf{x}-\mathbf{r})^2} - R)^2} - r \tag{7.10}$$

This allows us to verify an algorithm not in an absolute way, but relative to the output of another algorithm or to a known result. This way we can measure and compare the quality of the algorithm under test. Relative verification

This method does not allow us to recognize the theoretical correctness of an algorithm, but rather its incorrectness, namely in case the algorithm yields a result that is inferior to an already known result. Thus, calculating the distances allows us to check the results of an algorithm based on practical or on theoretical examples. Incorrectness can be stated

First, we can check to see that the result looks reasonable, second we can compare it with that of another algorithm and specify a rate of quality, and third we can test the algorithm on theoretically constructed or practical bestfit problems, the results of which are already known (but perhaps not the optimal ones!). All these tests together yield a good estimate of the correctness of the tested algorithm. Possible tests

Of course, such statements can be valid only for a restricted range of the input parameters. Because usually we are dealing with nonlinear, iterative algorithms, convergence strongly depends on the chosen initial values. Furthermore, as we search for the minimum, there is always the danger of arriving at a local minimum instead of the global one. Given range of input range

However, in the practice of coordinate metrology it is obvious in most cases if a local instead of the global minimum has been found. Local minimum and global minimum

As an example for a CM-bestfit where normally more than one minimum exists, we consider the cylinder bestfit. In case of a fit through the minimum number of points (5), there are normally three possible solutions. By adding additional measuring points we get an overdetermined problem which has correspondingly three different (local and global) minima. Based on the nominal data, of course, Cylinder bestfit with more than one minimum

it will be obvious which one of these is the actually required minimum.

Not always global minimum desired

Moreover, a lot of cases can theoretically be constructed (e.g. by the reconstruction of a sculptured surface from measured data) where in fact not a global but a *local* minimum is sought. Figure **??** (representing, e.g., a slice of a sculptured surface) shows two possible ways (global and local minimum) a parametric polynomial curve $(x(u), y(u))$ could pass through the measured data. Of course it is obvious that we prefer the *local* minimum curve (to be a curve on a *real* surface), even though the global minimum curve fits better (in our example exactly).

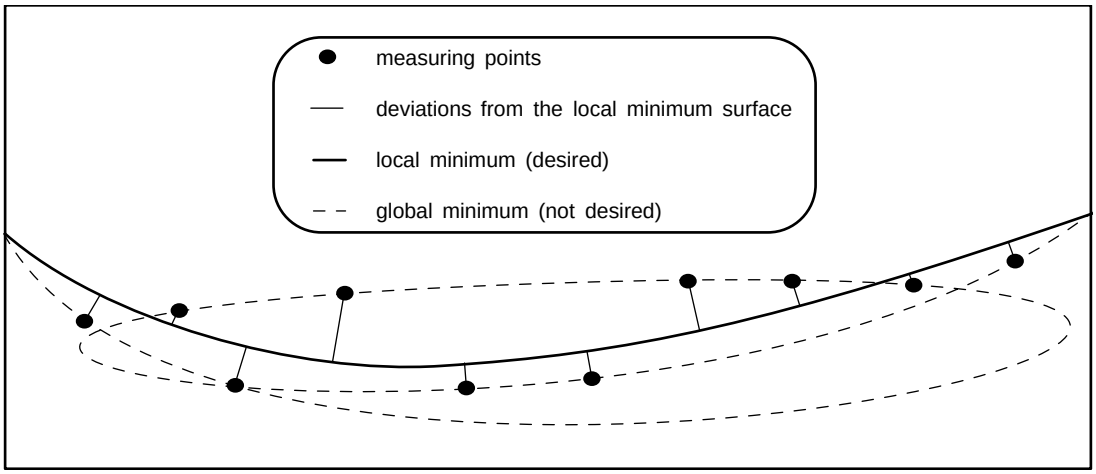


Figure 7.8: Global and local minimum of a sculptured surface bestfit

Local minimum in the neighborhood of the nominal values

With the appropriate initial values (i.e. the nominal values) $\mathbf{p_0}, \mathbf{a_0}$ and $\mathbf{t_0}$ (located in the neighborhood of the desired minimum), we implicitly *select* the minimum which the algorithm has to search for.

Well distributed points for sharp minimum

In good CM-practice one should always work with enough measuring points. This assures that the *desired* minimum is very likely identical to the *global* minimum of the actual bestfit problem. Furthermore, one should *steepen* this global minimum as far as possible by a *reasonable* distribution of the measuring points (See section 5.5.3). This will shield the *output* data (i.e., the bestfit parameters) as much as possible from the influence of *disturbances* in the *input* data that may be caused by measuring errors or random form deviations.

7.3 Distance Calculation for Sculptured Surfaces

Depending on the outputted distances

When testing bestfit algorithms for non-standard surfaces, a special problem is the lack of an exact distance function. Thus we depend on the distance-information delivered by the algorithm itself. This is, however, not an objective testing method.

Re-calculate distances, based on the delivered u, v values

A better concept is to use only the *surface coordinates* (u, v) instead of distances as the output of the algorithm under consideration. Using this information only makes sure that we get points lying exactly on the surface. We are not sure if

we have the real footpoints; perhaps the algorithm is not able to calculate the exact ones or it delivers the wrong ones by mistake. However, the distances evaluated by means of the surface coordinates (u, v) can not be shorter than the real ones. Thus intentional or unintentional cheating becomes impossible. We can now calculate an upper limit for the value of the objective function:

$$Q_2(\mathbf{a}, \mathbf{t}, \mathbf{p}) = \sum_{i=1}^{n} d^2(\hat{\mathbf{x}}_\mathbf{i}, u_i, v_i, \mathbf{a}, \mathbf{t}, \mathbf{p}) \ , \tag{7.11}$$

where the function d is the 'distance function' as defined in (4.8).

Residual calculation as a comparing tool

For example, if we use this procedure to test the algorithm described in section (4.2) which fits a point cloud to another point cloud, we obtain a larger Q_2 than the exact one. Thus *residual calculation* is also a useful tool for assessing the quality of (approximately calculating) bestfit algorithms for *sculptured* surfaces.

7.4 Generating Test Data

Different ways to get test-data

To check if an algorithm is correct, we need test data together with known results. We can obtain these results in three different ways.

Test-data from a real measurement

First, we could obtain them physically, e.g., by rotating and translating a workpiece by pre-defined amounts. Even though this is possible, it would be a time-consuming and unprecise method.

Test-data from a reference algorithm

The second way is to obtain the desired results by using a reference algorithm. The disadvantage is that normally we have no reference algorithm for a *sculptured* surface. But also dealing with surfaces where reference algorithms exist, we have no guarantee that they are perfect in all cases.

Test-data from simulation

These drawbacks can be avoided by a third method: Start with a known solution and then construct theoretically a bestfit problem belonging to it. This can be done in two different ways. A more physical approach would be to develop a model of the probing process on a coordinate measuring machine and then to simulate it.

Test-data from null-space method

A more mathematical approach is the so-called *null-space method* proposed by Cox and Forbes [CF92]. They modify a 'trivial' bestfit problem (trivial because the residual vector is zero) in a well-defined way to a 'normal' bestfit problem (with non-zero residual vector) in such a way that the initial solution is left unchanged.

Advantages and drawbacks

The advantage of this method is that we can construct a problem with an exactly known solution (not only with known limits). A drawback is that we can only generate problems in a subspace of the space of all possible bestfit problems. We could also say that we have less 'reality' than when simulating the physical process of probing. Besides these two rather academic objections, the main problem – especially in our case – is that the null space method needs the values of

the Jacobian matrix. A key step in our algorithm is the function-independent generation of the Jacobian matrix; thus, we cannot check this important step by using the null-space method. Doing this so would look like an examination, where candidate and expert are the same person.

Realistic simulation

To show the correctness of our algorithm in a realistic way, we should use a physical simulation. It generates realistic measuring points based on known position/orientation and dimension.

7.5 Simple Simulation Model

Starting with nominal values

We simulate the physical probing process starting from nominal position/orientation $\mathbf{t_0}, \mathbf{a_0}$ and nominal dimension parameters $\mathbf{p_0}$.

Probing at the nominal locations in normal direction

For an automatic measurement the probing points on the surface are calculated with these nominal values and a set of surface coordinates $(u_{1_0}, v_{1_0}, \ldots, u_{n_0}, u_{n_0})$. These points have to be approached in the normal direction, to minimize the influence of friction. The normal directions are also calculated based on the nominal values because the real values are not known at the beginning.

Simulated deviations

Now well-defined deviations have to be simulated. The *actual* surface we simulate with dimension deviations $\mathbf{\Delta p}$. Its *actual* position/orientation is simulated with the pretended deviations $\mathbf{\Delta a}$ and $\mathbf{\Delta t}$. Form deviations will be treated in the next section.

Simple model

The center of the probing sphere approaches the *actual* surface along the straight line given by the *nominal* normal direction in the surface point, where a measurement is required. In a simple model we assume the measuring point to be the center of the probing sphere at the moment it touches the simulated (substitute) surface.

Nominal path of the probe center

First we have to calculate the nominal normal direction in the nominal point. The straight line going through the nominal point with this direction is given by:

$$\mathbf{x}(l) = R(\mathbf{a_0}) \cdot (\mathbf{x}(u_0, v_0, \mathbf{p_0}) + l \cdot \mathbf{n}(u_0, v_0, \mathbf{p_0}) + \mathbf{t_0}) \tag{7.12}$$

for $-\infty < l < \infty$, where (u_0, v_0) are the nominal surface coordinates and the surface normal $\mathbf{n}(u, v, \mathbf{p})$ is computed by

$$\mathbf{n}(u, v, \mathbf{p}) = \frac{\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v}}{\| \frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v} \|}(u, v, \mathbf{p}) \ .$$

Line between probe center and contact point

The measuring point is situated on this straight line. At the moment of contact the center of the probe sphere has distance r (=sphere radius) from the simulated (substitute) surface:

$$\mathbf{x}(u, v, \mathbf{a}^*, \mathbf{t}^*, \mathbf{p}^*) \ ,$$

where

$$\mathbf{p}^* = \mathbf{p_0} + \mathbf{\Delta p} \qquad \mathbf{t}^* = \mathbf{t_0} + \mathbf{\Delta t} \qquad \mathbf{a}^* = \mathbf{a_0} + \mathbf{\Delta a} \ .$$

This can be described by the expression

$$\mathbf{x}(u,v) = R(\mathbf{a}^*) \cdot (\mathbf{x}(u,v,\mathbf{p}^*) + r \cdot \mathbf{n}(u,v,\mathbf{p}^*) + \mathbf{t}^*) \ , \tag{7.13}$$

representing an offset surface.

Location of probe center in the moment of touching the surface

Thus, to find the center of the probe sphere at the moment of contact (Figure 7.9), we equate the expressions (7.12) and (7.13). This reduces the problem to finding the solution u^*, v^* and l^* of the nonlinear set of equations:

$$R(\mathbf{a}^*) \cdot (\mathbf{x}(u^*,v^*,\mathbf{p}^*) + r \cdot \mathbf{n}(u^*,v^*,\mathbf{p}^*) + \mathbf{t}^*)$$

$$-R(\mathbf{a_0}) \cdot (\mathbf{x}(u_0,v_0,\mathbf{p_0}) - l^* \cdot \mathbf{n}(u_0,v_0,\mathbf{p_0}) + \mathbf{t_0}) = 0 \ . \tag{7.14}$$

Calculating the unknowns u^*, v^* and l^*

Equation (7.14) can be solved by a Newton iteration. Inserting the so determined value l^* in (7.12), we can calculate the synthetical measuring point delivered by this simple model.

7.6 Refined Simulation Model

7.6.1 Deficiencies of the Simplified Model

Spread around the given driving path

We now refine this simple model, taking into consideration that the movement and thus the contact of the probe on the surface cannot be performed while keeping the sphere center always on an ideal straight line.

Non-ideal regulation

One reason is the restriction given by a non-ideal numerical control of the axes.

SW-compensation

Another reason is the SW-compensation of the geometric errors of a CMM: The compensation is applied to the measured coordinates but not to the movements.

No orthogonal approach

Furthermore, the sphere could slip in the moment of contact, particularly if we take into account that it approaches along the *nominal* normal direction, not identical to the actual one. When the probe sphere drives to a microscopical edge or corner it would evade randomly.

Driving is less accurate than measuring

For all these reasons the driving accuracy of a CMM is a magnitude worse than the accuracy of the delivered X, Y, Z-coordinates.

Modelling by random deviations

All of these effects can be modelled by a rotationally symmetric random deviation around the theoretical, straight driving path of the probe sphere center.

Probability cylinder

Instead of a straight line, we introduce a probability cylinder centered on it. For the probability of a deviation from the ideal line we assume a Gaussian distribution (random influences), whose width σ depends on the amount of disturbances mentioned above.

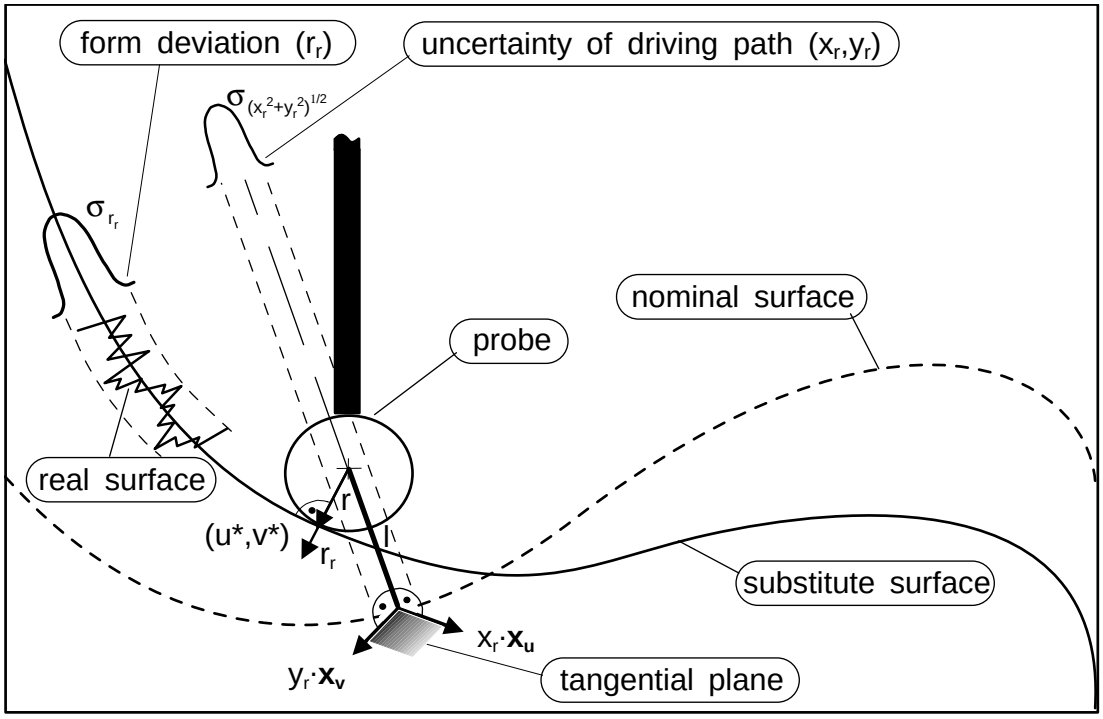


Figure 7.9: Mathematical model of the probing process

7.6.2 Simulating Uncertainty of Driving Path

Probability contributions in two dimensions

For a two-dimensional deviation where we assume in two orthogonal directions two independent probability deviations with the center in origin and with the same width σ, we get a rotation-symmetric and thus isotropic probability distribution [PFTV89]:

$$\begin{aligned} p(x,y)dxdy &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} dxdy \\ &= \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} dxdy \\ &= \frac{1}{2\pi\sigma^2} e^{-\frac{r^2}{2\sigma^2}} \underbrace{\left| \begin{array}{cc} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \phi} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \phi} \end{array} \right|}_{r} dr d\phi \,. \end{aligned} \tag{7.15}$$

We recognize that the probability p is independent of ϕ, thus the probability is isotropic.

$$p(r,\phi) = r \cdot \frac{1}{2\pi\sigma^2} e^{-\frac{r^2}{2\sigma^2}} \tag{7.16}$$

$$\Longrightarrow p(r)dr = dr \int_0^{2\pi} p(r,\phi)d\phi = \frac{r}{\sigma^2} e^{-\frac{r^2}{2\sigma^2}} dr \tag{7.17}$$

Probability to deviate by r from the ideal path

$p(r)$ is the probability that the probe meets the surface in the distance r from the intersection point given by the simple model above.

With this probability deviation independent of direction ϕ we define a *probability cylinder* around the direct driving path (Figure 7.9). If we chose random components x_r and y_r distributed according to (7.15), we get randomly shifted straight lines instead of (7.12):

Randomly shifted straight lines

$$\mathbf{x}(l)_{random} = \tag{7.18}$$

$$R(\mathbf{a_0})\cdot(\mathbf{x}(u_0, v_0, \mathbf{p_0}) + x_r\cdot\mathbf{x_u}(u_0, v_0, \mathbf{p_0}) + y_r\cdot\mathbf{x_v}(u_0, v_0, \mathbf{p_0}) + l\cdot\mathbf{n}(u_0, v_0, \mathbf{p_0}) + \mathbf{t_0}) \ ,$$

where

$$\mathbf{x_u}(u, v, \mathbf{p}) = \frac{\frac{\partial \mathbf{x}}{\partial u}}{\| \frac{\partial \mathbf{x}}{\partial u} \|} \qquad \mathbf{x_v}(u, v, \mathbf{p}) = \frac{\frac{\partial \mathbf{x}}{\partial v}}{\| \frac{\partial \mathbf{x}}{\partial v} \|} \ .$$

This expression has to be inserted into (7.14).

7.6.3 Simulating Form Deviations

So far we have described a substitute feature with no form deviations. This type of deviation cannot be described in a simple way by a pre-defined value like the dimension and position/orientation deviation. To specify these deviations by *one* parameter only we use a Gaussian probability distribution with standard deviation σ. This would meet also the statement made in section 2.1: Ideally a more general class of substitute features with additional dimension parameters (e.g. conicity, ellipticity) should be defined; so the remaining *form* deviations would include (as far as possible) no *systematic* deviations.

Simulating form deviation as a random distribution

Of course we could imagine a lot of other appropriate ways to describe a form deviation. Even if the form deviation contains *systematic* influences, it is not a bad approach to treat its frequency distribution as a random distribution. If we perform a bestfit observing a Gaussian criterion, it is reasonable also to use a Gaussian distribution. In the case of a Chebychev bestfit it would be reasonable to simulate a uniform block distribution.

Appropriate provability deviation

From the implementation point of view, there is the problem that a standard random number generator delivers uniformly distributed values. If we want to have random values based on Gaussian distribution, we have to consider how to transform a *uniform* random variable into a *Gaussian* random variable.

Transforming a Uniform Distribution into a Gaussian Distribution

This can be done by the use of the so-called Box-Muller method [PFTV89].

Box-Muller method

The linear transformation

Transforming the range

$$w = \frac{W - W_{Max}}{W_{Max} - W_{Min}} \tag{7.19}$$

transforms a uniform random variable W in the range $[W_{min}, W_{max}]$ to a uniform random variable w in the range $[-1, 0]$.

Next, we need a parameter transformation $w(r)$ of the probability integral

Searching for deviation in form (7.17)

$$1 = \int_{W_{Min}}^{W_{Max}} p(W)dW \equiv \int_{-1}^{0} p(w)dw$$

in such a way as to obtain a deviation in form of (7.17):

$$\int_{-1}^{0} \underbrace{p(w)}_{uniform\ \ deviate} dw = \int_{0}^{\infty} \underbrace{p\,(w(r))}_{constant\ \ 1} \frac{dw}{dr} dr = \int_{0}^{\infty} \underbrace{p(r)}_{deviate\ \ (7.17)} dr \ .$$

Parameter transformation

By specifying

$$w(r) = -e^{-\frac{r^2}{2\sigma^2}} \tag{7.20}$$

Inverse function

we get

$$r(w) = \sqrt{-2 \cdot \ln(-w)} \cdot \sigma \tag{7.21}$$

Derivative

for the inverse function and therefore

$$\frac{dw}{dr} = \frac{r}{\sigma^2} \cdot e^{-\frac{r^2}{2\sigma^2}} \ . \tag{7.22}$$

Gaussian random generator

Thus, by transforming the values delivered by the uniform random generator with the function (7.20) we get random values distributed according to the probability distribution (7.17):

$$\int_{-1}^{0} p(w)dw = \int_{-1}^{0} \underbrace{p(w)}_{=1} \frac{dw}{dr} dw = \int_{0}^{\infty} \frac{r}{\sigma^2} \cdot e^{-\frac{r^2}{2\sigma^2}} dr = \int_{0}^{\infty} p(r)dr \ .$$

Two independent distributions

As shown in (7.15) this distribution can be transformed into two independent Gaussian distributions with the simple coordinate transformation

$$x = r \cdot \cos(\phi)$$

$$y = r \cdot \sin(\phi)$$

while we input for ϕ another uniformly distributed random value.

7.6.4 Improved Model

Introducing form deviations

The form deviations are measured normal to the substitute feature, i.e., normal to the ideal assumed actual surface in (7.13) (Figure 7.9). Therefore the form deviations can be simulated by a random value r_r which is inserted accordingly in (7.13):

$$\mathbf{x}(u,v)_{random} = R(\mathbf{a}^*) \cdot (\mathbf{x}(u^*,v^*,\mathbf{p}^*) + (r + r_r) \cdot \mathbf{n}(u^*,v^*,\mathbf{p}^*) + \mathbf{t}^*) \ . \tag{7.23}$$

Introducing probe tip deflection

As another refinement, probe tip deflection can be simulated. This can be done on the basis of (6.4) and (6.10) by replacing the probe sphere with an ellipsoid.

The deflection is described by a given probe characteristic matrix C which is calculated according to (6.10): Probe characteristic matrix

$$C := R_E \cdot E \ , \tag{7.24}$$

where E defines the half axes of an ellipsoid $E = diag(h_x, h_y, h_z)$ and the rotation matrix R_E defines the orientation of this ellipsoid.

Instead of (7.23) we can write then Modified formula

$$\mathbf{x}(u,v)_{random} = R(\mathbf{a}^*) \cdot (\mathbf{x}(u^*, v^*, \mathbf{p}^*) + C \cdot \mathbf{n}(u^*, v^*, \mathbf{p}^*) + r_r \cdot \mathbf{n}(u^*, v^*, \mathbf{p}^*) + \mathbf{t}^*) \ . \tag{7.25}$$

By this final refinement, we obtain a tool which delivers artificial data by simulating the physical reality of the measuring process and the workpiece to a high degree. It allows for testing any algorithm like, in particular, FUNKE. Physical reality

A bestfit algorithm should always allow for finding the theoretically correct result. For a few simulated measuring points, it would find a solution which is rather better than the simulated one. Increasing the number of points, the solution approaches the simulated one. For a large number of points the correct bestfit solution is only slightly better than the simulated solution ($\cong$ 0.1% using 10000 points). So we get the following rules: Rules

1. Providing a simulated form deviation of 0.0, a remaining form deviation of 0.0 should be found as well.

2. If a form deviation > 0.0 is simulated, an equal or a better one should be found.

3. The more measuring points (statistics!) are used, the nearer the best-fitted form deviation will approach the simulated one; however, even then the following statement is valid:

   Fitted average form deviation $\leq$ Simulated average form deviation

Otherwise, the bestfit algorithm has failed.

Chapter 8

Summary

8.1 Background

Evaluation software in CM

The core of Coordinate Metrology (CM) is the evaluation software. On one hand it turns a Coordinate Measuring Machine (CMM) into a flexible tool because it becomes principally capable of measuring all kinds of surfaces. On the other hand, it also allows – in contrast to quantifying summarized deviations for pure quality control – a proper analysis of the deviations according to *position/orientation, dimension and form* for an improved production process.

8.2 Goals

Generalized evaluation concept

By consistently pursuing the basic goals of Coordinate Metrology, we contribute to a generalized and standardized evaluation concept which allows us to extend these goals also to complex and sculptured surfaces. With the proposed algorithm and the software FUNKE we can additionally compute these surfaces in a very efficient way, absolutely comparable with the bestfit of standard surfaces. Especially in this field, however, there is a gap in the commercially available software tools today.

Parametric description

The algorithm uses the parametric description of a surface $\mathbf{x}(u, v)$, which is needed (sometimes apparently, sometimes in a hidden form) for any NC-driven production process. (Movements of axes as functions of time represent the individual parametric functions $x(t), y(t)$, etc.) Thus, we do not depend on the availability of implicit distance functions $d_{\mathbf{x}} = f(\mathbf{x})$ which are given analytically only for standard elements. The interface to CAD/CAM-systems can be established in quite a natural way.

Sophisticated evaluation

Furthermore, even standard surfaces can be treated in a more sophisticated manner because it becomes possible to define additional parameters which allow us to analyze deviations caused by the actual production process. We can bestfit a more complex substitute feature, including the nominal feature as a subclass

(e.g. upper class 'ellipsoid' containing the subclass 'sphere'). We are then able to determine additional dimension parameters for describing all possible systematic errors of the manufacturing process until the remaining form deviations are reduced to the random errors only. Thus, measuring results can be used for a quantitative correction of the workpiece instead of a simple 'pass/fail'-decision.

Extended flexibility

In the evaluation process we get more flexibility. We can not only evaluate *individual* standard features but also *compounds* of features (e.g., a square block) as a whole, according to position/orientation, dimension and form. These compound features can be combinations of standard surfaces as well as combinations of sculptured/complex ones. This allows us to adapt the evaluation to the desired functionality of the workpiece.

Constraints

Because we can freeze each bestfit parameter individually we can treat special or poorly conditioned measurement problems in quite an easy way. There is also a concept to treat more general constraints in a standardized way.

Radius correction and deflection compensation

Probe radius correction and deflection compensation can be taken into account using different types of probes on the same workpiece. The deflection compensation of the probe tip can be performed by using the normal direction of the substitute feature instead of the normal direction of the nominal feature or the (inaccurate) normal direction delivered by an analogous probe.

Sculptured surface measurement on manually operated CMM's

This allows us to use not only analogous probing devices but also the much cheaper switching ones for sculptured surface measurement. Thus, and also because we use a surface-oriented instead of a pointwise calculating bestfit, we can also perform sculptured surface measuring with manually operated Coordinate Measuring Machines.

8.3 Realization

Function-independent bestfit

We want to make the bestfit procedure independent of the actual surface function. We try then to separate the geometrical description of a surface from its position/orientation description. For this reason we define a generally applicable interface with g parameters $(p_1, \ldots p_g) = \mathbf{p}$ describing geometry, based on the parametric surface (or curve) description $\mathbf{x}(u, v, \mathbf{p})$. The description is in a coordinate system where it can be held as simple as possible (e.g. $\mathbf{x}_{plane} = (u, v, 0)$) and where no position/orientation-information is hidden in the remaining parameters. For standard surfaces this is a principal axes system.

Keeping parametric structure

Instead of minimizing an objective function depending on the individual distances which would turn out to be very complicated in parametric representation, we keep the useful triple structure by directly forming a nonlinear, overdetermined system of equations whose least squares solution is identical with the minimized sum of squares of the individual distances.

Gauss-Newton method

This system of equations is solved by the Gauss-Newton method where we have to solve a sequence of overdetermined, linear systems in the least squares sense.

The individual matrices of equations are given by the Jacobians, not only as functions of all bestfit parameters $\mathbf{p}, \mathbf{a}$ and $\mathbf{t}$, but also as functions of all surface coordinates $(u_1, v_1, \ldots, u_n, v_n)$. For this reason, the matrices are very large if we use a large number of measuring points.

Separating the Jacobian

By splitting up the Jacobian into surface-function-*dependent* and into generally applicable, surface-function *independent* parts, we can automate its generation, treating the actual surface function as a generic function which can be implemented by referencing it by a function address pointer. This allows us to make the bestfit-algorithm surface independent.

Evaluation time proportional to measuring time

For solving the linear system, we have to search for efficient solution methods. Otherwise, because of the reasons mentioned above, the computational effort would increase with the *third* power of the number of measuring points while the memory demands would increase with the *second* power. Using Gauss-Newton (instead of Newton), the resulting system of equation becomes very sparse. Exploiting this sparsity, we can achieve that computational costs and memory demands increase only *linearly* with the number of measuring points. From the practical point of view, this means that the ratio between measuring time and calculation time remains *constant*.

Exploiting sparsity

Different solution methods for solving the sparse system are discussed. With the intention of achieving the best possible savings in memory space, we can use the system of normal equations directly, bypassing the overdetermined system. From the point of view of cost efficiency it turns out that an even better way can be found in our case. By a sequence of orthogonal transformations (Givens rotations) which can also be interpreted geometrically, we transform the system of equations to a special structure. Using this structure we solve a *reduced*, overdetermined system of equations instead of the full one.

SVD to detect hidden redundancies

This reduced, overdetermined system of equations is not sparse anymore. So we do not loose any computational performance by applying a standard solution method to this system. Besides solving it by normal equations or orthogonal transformations (more recommendable from the point of view of numerical stability), we can also solve it by the Singular Value Decomposition (SVD) which allows us to recognize hidden redundancies. This can be especially important for fitting or modifying the form of a sculptured surface.

Detailed problems

As a next step some detailed problems of implementation are discussed. We show that probe radius correction and probe tip deflection compensation can be performed in quite an easy way based on the parametric description of a surface. Then, the concept of dealing with rotation and translation symmetries is shown. By a similar concept we can freeze all parameters appearing in bestfit and also treat more general constraints. On a simple example, the flexibility we have with the implemented hierarchical data structure in combination with the generally applicable bestfit algorithm is demonstrated.

Residual calculation

Furthermore, the verification of a bestfit algorithm is discussed. This plays a more and more important role in metrology. For this reason we show a simple method to compute the implicit distance function of a rotationally symmetric

surface as a function of its position r and direction n. Based on these distance functions we discuss concepts to compare and test bestfit algorithms.

Simulation

Finally a simulation algorithm is presented which simulates the probing process in a realistic manner. With this simulation we can efficiently produce artificial measuring points with defined spread (according to form deviation) on an imaginary surface with known, predefined position/orientation and dimension deviations. Together with residual calculation and the 'null space'-method as presented by Cox and Forbes [CF92], we propose this simulation algorithm as a general testing tool for bestfit algorithms, not only restricted to our purposes.

8.4 Future

L_2-norm can be minimized exactly

The proposed method can be used for Gaussian (L_2-norm) bestfit. Using parametric description for minimizing any norm other than L_2 would become quite complicated (c.f., section 4.3.1) and could not be performed in general manner at reasonable computational costs.

Approximation of the Chebychev L_∞-norm

The Chebychev norm (L_∞-norm), however, would especially be of some interest for practical use [Hei89]. Introducing weights in the overdetermined system of equations, we can perform a sequence of Gaussian bestfits where the weights for the individual equations in each step are changed to approximate the L_∞ solution [Law61]. The first step is performed with unweighted distances. The squares of the resulting distances are used as weights for the next step and so on. By this procedure we could *approximate* an L_2, L_4, L_8, L_{16}-norm, etc.; finally, we would arrive at an approximation of the Chebychev-norm because big distances are weighted more and more. This is planned as an enhancement of FUNKE.

Compensation

Furthermore FUNKE can be used for the compensation [DFK88], [SB93] and approximation of sculptured surfaces. 'Compensation' means that after bestfitting the nominal shape of a sculptured surface according to position/orientation, a 'compensating surface' with slightly corrected shape is computed. The corrections are made not in the *same* direction but in that *opposite* to the form deviations (normal to the nominal surface). This way, we can correct *reproducible* influences of the process by an additional manufacturing step.

Industrial applications based on FUNKE

Due to the wide scope and flexibility of FUNKE we refrained from transforming it into a general ready-to-use form. Instead of this, a large subroutine library is included by means of which the appropriate software for a specific measuring task can be implemented relatively easily but still on the source code level. In co-operation with industry, it is another future goal now to make this possible more and more also on the interactive level for special, but often used, measuring tasks, where FUNKE has some advantages over conventional evaluation methods.

User friendliness on the source code level

Furthermore, as explained in section 6.3.2.1, the underlying data structure is influenced by an object oriented concept (although most parts are written in the language C). So a planned upgrade would be to develop an object oriented interface (in language C++) for the basic algorithms (in language C). Such an improved

user friendliness on the source code level would make it easy to add/implement software modules aimed at specific applications with a minimum of time expenditure.

Optical devices integrated in FUNKE

Last but not least, FUNKE is now being extended for use with state-of-the-art optical 3D measuring devices [Bre93] which yield large number of measuring points ($\geq 250'000$) in a short time, though currently with relatively poor accuracy ($\geq 50\mu m$). This system can be applied to special measuring tasks as well as to digitizing, combined with surface reconstruction. This becomes more and more important in connection with the quickly growing field of *rapid prototyping*.

Bibliography

[AKK89] H. Aoyama, M. Kawai, and T. Kishinami. A New Method for Detecting the Contact Point between a Touch Probe and a Surface. *Annals of the CIRP*, 38(1):517–520, 1989.

[BC76] P. Bourdet and A. Clément. Controlling a Complex Surface with a 3 Axis Measuring Machine. *Annals of the CIRP*, 25(1):359–361, 1976.

[BC88] P. Bourdet and A. Clément. A Study of Optimal-Criteria Identification Based on the Small-Displacement Screw. *Annals of the CIRP*, 37(1):503–506, 1988.

[BCF92] B.P. Butler, M.G. Cox, and A.B. Forbes. The reconstruction of workpiece surfaces from probe coordinate data. In *Mathematics of Surfaces*, Edinburgh (Scotland), September 1992. IMA.

[BD89] Paul T. Boggs and Janet R. Donaldson. Orthogonal Distance Regression. Technical Report NISTIR 89-4197, U.S. Departement of Commerce, November 1989. Applied and Computational Mathematics Division.

[BFH94] B.P. Butler, A.B. Forbes, and P.M. Harris. Algorithms for Geometric Tolerance Assessment. Report DITC 228/94, NPL, November 1994.

[BR90] Paul T. Boggs and Janet E. Rogers. Orthogonal Distance Regression. *Contemporary Mathematics*, 112:183–194, 1990.

[Bre93] B. Breuckmann. Topometrische 3D-Messtechnik. *Bildverarbeitung und optische Messtechnik*, page 124 ff., 1993.

[Bri89] British Standards Institution. *Assessment of position, size and departure from nominal form of geometric features*, 1989. BS 7172.

[Buc89] A. Bucher. N-Koordinaten-Messtechnik für prozessnahes Messen. Fertigungstechnisches Kolloquium ETH Zürich, June 1989.

[CF92] M.G. Cox and A.B. Forbes. Strategies for Testing Form Assessment Software. Report DITC 211/92, NPL, December 1992.

[DBPK84] N. Duffie, J. Bollinger, R. Piper, and M. Kroneberg. CAD-Directed Inspection and Error Analysis Using Surface Patch Databases. *Annals of the CIRP*, 33(1):347–350, 1984.

[DFK88] N. Duffie, S. Feng, and J. Kann. CAD-Directed Inspection, Error Analysis and Manufacturing Process Compensation Using Tricubic Solid Databases. *Annals of the CIRP*, 37(1):149–152, 1988.

[DH93] C. Diaz and T.H. Hopp. Testing of coordinate measuring system software. In *Proceedings of the American Society for Quality*, 1993.

[DIN86] DIN-32880. Koordinatenmesstechnik: Geometrische Grundlagen und Begriffe. Technical report, DIN, 1986.

[Dri89] R. Drieschner. Test von Software für Koordinatenmessgeräte mit Hilfe simulierter Daten. *VDI-Berichte*, 751:351–365, 1989.

[ES89] M. Engeli and U. Schneider. Kurven- und flächenorientierte Modellierung mit NURBS. *CAD-CAM Report*, 3:100–105, 1989.

[For87] A.B. Forbes. Fitting an Ellipse to Data. Report DITC 95/87, NPL, December 1987.

[For91] A.B. Forbes. Least-Squares Best-Fit Geometric Elements. Report DITC 140/89, NPL, February 1991.

[For92] A.B. Forbes. Geometric Tolerance Assessment. Report DITC 210/92, NPL, October 1992.

[Gan90] W. Gander. Algorithms for the Polar Decomposition. *SIAM Journal on Scientific and Statistical Computation*, 11(6), 1990.

[Gar91] Th. Garbrecht. *Ein Beitrag zur Messdatenverarbeitung in der Koordinatenmesstechnik*. PhD thesis, Univ. Stuttgart, 1991.

[Gaw89] B. Gawande. *Auswerteverfahren für die Prüfung von Werkstücken mit gekrümmten Oberflächen mit Koordinatenmessgeräten*. PhD thesis, Univ. der Bundeswehr Hamburg, 1989.

[GGS94] Walter Gander, Gene H. Golub, and Rolf Strebel. Least-Squares Fitting of Circles and Ellipses. *BIT*, 34:558–578, July 1994.

[GMPW90] C. Gächter, B. Monstein, C. Putzi, and A. Wirtz. Die Vektorielle Tolerierung, ein Werkzeug zur Funktionsanalyse. *Technische Rundschau*, 41:26–37, 1990.

[Goc82] G. Goch. *Theorie der Prüfung gekrümmter Werkstück-Oberflächen in der Koordinatenmesstechnik*. PhD thesis, Hochschule der Bundeswehr Hamburg, 1982.

[Goc90] G. Goch. Efficient Multi-Purpose Algorithm for Approximation and Alignement Problems in Coordinate Measurement Techniques. *Annals of the CIRP*, 39(1):553–556,727, 1990.

[GPS80] G. Goch, R.D. Peters, and F. Schubert. Beschreibung gekrümmter Flächen in der Mehrkoordinaten-Messtechnik. *VDI-Berichte*, 378:85–91, 1980.

[GR89] Th. Garbrecht and S. Roth. Strategien für das Messen von Freiformflächen. *VDI-Berichte*, 751:303–320, 1989.

[GS74] G. Geise and S. Schipke. Ausgleichsgerade, -kreis, -ebene und -kugel im Raum. *Mathematische Nachrichten*, 62(5):65–75, 1974.

[GSW90] M. Gulliksson, I. Söderkvist, and P.A. Wedin. Algorithms for Orthogonal Regression. Technical report, Univ. of Umea, June 1990. Draft.

[GT92] G. Goch and U. Tschudi. A Universal Algorithm for the Alignment of Sculptured Surfaces. *Annals of the CIRP*, 41(1):597–600, 1992.

[GvL89] G.H. Golub and C.F. van Loan. *Matrix Computations*. John Hopkins University Press, 1989.

[Hei89] M. Heinrichowski. *Normgerechte und funktionsorientierte Auswerteverfahren für punktweise erfasste Standardformelemente*. PhD thesis, Univ. der Bundeswehr Hamburg, 1989.

[Hen74] H. Henning. Darstellung gekrümmter Flächen durch Approximation von Messpunkten. *wt-Z. ind. Fertigung*, 64:676–681, 1974.

[HH69] P. Henrici and P. Huber. *Analysis*. AMIV, 1969.

[Hir88] Rolf Hirschi. Vom Rundheitsmessgerät zum Rundheitsmesssystem. *Technische Rundschau*, 18:28–41, 1988.

[Hop93] T.H. Hopp. Computational Metrology. *Manufacturing review*, 6(4):295–304, 1993.

[Hos92] J. Hoschek. Interpolation und Approximation von Punktmengen. TH Darmstadt, 1992.

[ISO81a] ISO-4156-1981. Straight cylindrical involute splines – Metric module, side fit – Generalities, dimensions and inspection. Technical report, International Organization for Standardization, 1981.

[ISO81b] ISO-4559-1981. Technical drawings – Geometrical tolerancing – Datums and datum-systems for geometrical tolerances. Technical report, International Organization for Standardization, 1981.

[ISO85] ISO-1101. Technical drawings, geometrical tolerancing, tolerancing of form, orientation, location and run-out, generalities, definitions, symbols, indications on drawings. Technical report, International Organization for Standardization, 1985.

[ISO94] TC 184/SC 4 ISO. Integrated generic resources: Geometric and topological representation. Industrial Automation Systems and Integration - Product data representation and exchange ISO 10303-42, International Organization for Standardization, Geneva, 1994.

[Ker87] J. Kern. *Theorie zur Berechnung von Formelementen der Koordinatenmesstechnik nach der Methode der kleinsten Fehlerquadratsumme*. PhD thesis, TU Dresden, 1987.

[Kle93] H. Klein. Prüfen von Freiformgeometrien. *TECHNICA*, 15/16:10–15, 1993.

[Kna93] W. Knapp. Feasibility Study for a Metrology Software Repository. Final report, Community Bureau of Reference (BCR), Brüssel (Luxembourg), December 1993.

[Kru86] H.J. Krumholz. *Optimierte Istgeometrie-Berechnung in der Koordinatenmesstechnik*. PhD thesis, RWTH Aachen, 1986.

[KS95] W. Knapp and D. Sourlier. Typical Applications, Case Study 3: Exact Measurement of a Sculptured Surface (Aircraft Wing). In John A. Bosch, editor, *Coordinate Measuring Machines and Systems*, chapter 12, pages 327–332. Marcel Dekker, March 1995.

[Law61] C.L. Lawson. *Contributions to the theory of linear least maximum approximation*. PhD thesis, UCLA, Los Angeles, 1961.

[Loe80] D. Loebnitz. *Untersuchungen zur Form- und Lageprüfung mit Hilfe von Mehrkoordinatenmessgeräten*. PhD thesis, RWTH Aachen, 1980.

[Lot83] W. Lotze. Zur Vereinheitlichung von Ausgleichsalgorithmen und deren Prüfung in der Koordinatenmesstechnik. *Wissenschaftliche Zeitschrift der TU Dresden*, 32(2):217–220, 1983.

[Lot90] W. Lotze. Mathematische Modellierung der geometrischen Messtechnik. *Technische Rundschau*, 31:36–45, 1990.

[MYL92] C.H. Menq, H.T. Yau, and G.Y. Lai. Automated Precision Measurement of Surface Profile in CAD-Directed Inspection. *IEEE Transactions on Robotics and Automation*, 8(2):268–278, April 1992.

[Nag94] L. Nagineviciene. Algorithmen zur Auswertung von Koordinatenmessungen an spiralverzahnten Kegelrädern. *Microtechnic*, 3:37–38, 1994.

[Neu91] H.J. Neumann. Neue Messverfahren und Auswertemethoden ermöglichen einen praxisgerechten Einsatz der Koordinatenmesstechnik in der Schmiedeindustrie. 3.IDS-Maschinenkolloquium, November 1991.

[Org90] IGES/PDES Organization. Initial Graphics Exchange Specification V.5.0. Technical Report NISTIR 4412, U.S. Dep. of Commerce, 1990.

[PFTV89] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1989.

[PH89] T. Pfeifer and A. Hemdt. Lageprüfung an Freiformkurven und -flächen in der Koordinatenmesstechnik. *Technisches Messen*, 56:17–22, 1989.

[Pra87] Vaughan Pratt. Direct Least-Squares Fitting of Algebraic Surfaces. *Computer Graphics*, 21(4):145–151, July 1987.

[PvH90] T. Pfeifer and A. vom Hemdt. Berechnung der Basiselemente und die Tasterkompensation in der Koordinatenmesstechnik. *Technisches Messen*, 57:114–123, 1990.

[Reg90] F. Regensburger. *Messung und Auswertung doppelt gekrümmter Flächen*. PhD thesis, TU Dresden, 1990.

[RMM94] A.H. Rentoul, G. Mullineux, and A.J. Medland. Interpretation of errors from inspection results. *Computer Integrated Manufacturing Systems*, 7(3):173–178, 1994.

[SB92] D. Sourlier and A. Bucher. Normgerechter Bestift-Algorithmus für Freiformflächen oder andere nicht-reguläre Ausgleichsflächen in Parameterform. *Technisches Messen*, 59(7/8):293–302, 1992.

[SB93] D. Sourlier and A. Bucher. Steigerung der Bearbeitungsgenauigkeit komplexer Geometrien durch exakte Mess-Auswertung und Kompensation. In *1.Internationales IWF-Kolloquium: Feinstbearbeitung technischer Oberflächen*, Zürich (Switzerland), April 1993. GWF/IWF.

[SB94] D. Sourlier and A. Bucher. Universelle Auswertmethoden. *Automation Precision*, 15(9):77–80, September 1994.

[SB95] D. Sourlier and A. Bucher. Surface-independent theoretically exact bestfit for arbitrary sculptured, complex, or standard geometries. *Precision Engineering*, 17(2):101–113, April 1995.

[Sch91] H. Schwetlick. Nichtlineare Parameterschätzung: Modelle, Schätzkriterien und numerische Algorithmen. *GAMM-Mitteilungen*, 2:13–51, 1991.

[Sch94] Th. Schreiber. *Analyse und Approximation von unstrukturierten Daten und Freiformflächen*. PhD thesis, Univ. Kaiserslautern, 1994.

[SH76] R. Strauss and H. Henning. Ein leistungsfähiges Verfahren zur Approximation von Raumpunkten durch ein Flächenstück. *Angewandte Informatik*, 9:401–406, 1976.

[Sim91] F. Simon. Fünfachsiges Fräsen mit fertigungsoptimierendem Messen an Turbinenschaufeln. *VDI-Z*, 133(6):90–101, 1991.

[Spa86] H. Spaeth. Orthogonal Least Squares Fitting with Linear Manifolds. *Numerische Mathematik*, 48:441–445, 1986.

[Sta90] H.J. Stadtfeld. Flexible Fertigung bei Kegelrad-Entwicklung und Produktion. *Schweizer Präzisions-Fertigungstechnik*, pages 51–58, 1990.

[SW80] M. Simmler and A. Wirtz. Ein neuer Weg in der Mehrstellenmesstechnik. *VDI-Z*, 122(15/16):655–661, 1980.

[VDA82] Working Group CAD/CAM VDA. VDA Surface Interface (VDAFS). Technical report, VDA-CAD/CAM-Committee, 1982.

[vH89] A.P. vom Hemdt. *Standardauswertung in der Koordinatenmesstechnik - ein Beitrag zur Geometrieberechnung*. PhD thesis, RWTH Aachen, 1989.

[Wae80] F. Waeldele. Gewindemessung mit rechnergesteuerten Drei-Koordinaten-Messgeräten. Technical Report 378, VDI, 1980.

[WBH75] M. Weck, P. Bagh, and R. Holler. Measuring of Gears with a Numerical Controlled 3-Axis Measuring Machine. *Annals of the CIRP*, 24(1):375–378, 1975.

[WG87] A. Weckenmann and B. Gawande. Prüfen von Werkstücken mit gekrümmten Flächen auf Koordinatenmessgeräten. *Technisches Messen*, 54(7/8):277–284, 1987.

[Wil93] D.E. Williams. *Optical Methods in Engineering Metrology*. Chapman and Hall, London, 1993.

[Wir88] A. Wirtz. Vektorielle Tolerierung zur Qualitätssteuerung in der mechanischen Fertigung. *Annals of the CIRP*, 37(1):493–498, 1988.

[Wir93] A. Wirtz. Vectorial Tolerancing, a Basic Element for Quality Control. Technical report, Computer Aided Tolerancing, April 1993.

[Wol84] H.R. Wollersheim. *Theorie und Lösung ausgewählter Probleme der Form- und Lageprüfung auf Koordinaten-Messgeräten*. PhD thesis, RWTH Aachen, 1984.

[WP83] A. Weckenmann and R.-D. Peters. Mathematische Beschreibung von gekrümmten Oberflächen für die Anwendung in der Koordinatenmesstechnik. *Annals of the CIRP*, 32(1):453–457, 1983.

[Xio91] Y. Xiong. Coordinate measurement techniques of complex surfaces and curves. In *Conference on PE and MS*, pages 147–161, Tiajin, 1991. CIRP.

Symbols

$\mathbf{x}$	$=$	(x, y, z)	3D-coordinates
u, v			surface coordinates
$\mathbf{P}$			fixed parameters
$\mathbf{r}$			position vector
$\mathbf{n}$			direction vector
n			number of measuring points
d_i			deviations
$d_{\mathbf{x}}$			dist. of point $\mathbf{x}$ to the surface
$\mathbf{n_u}, \mathbf{n_v}$			u, v-directions
g			number of dimension-parameters
$\mathbf{p}$	$=$	$(p_1, \ldots, p_g)$	dimension/geometry parameters
$\mathbf{x}'(u, v, \mathbf{p})$			special parametric representation
m_x, m_y			center coordinates of the circle
r			radius of the circle
Q_m			objective function based on L_m-norm
Q			orthogonal matrix
$\mathbf{a}$	$=$	(a, b, c)	xyz-rotations
$\mathbf{t}$	$=$	(t_x, t_y, t_z)	xyz-translations
$\bar{\mathbf{t}}$	$=$	$R(\mathbf{a}) \cdot \mathbf{t}$	alternatively defined translation
$\mathbf{x}(u, v, \mathbf{p}, \mathbf{a}, \mathbf{t})$			general parametric representation
$R(\mathbf{a})$	$=$	$R_x(a) \cdot R_y(b) \cdot R_z(c)$	rotation matrix
$\mathbf{p_0}$	$=$	$(p_{1_0}, \ldots, p_{g_0})$	nominal dimensions
$\mathbf{x'}_{nom}(u, v)$	$=$	$\mathbf{x}'(u, v, \mathbf{p_0})$	nominal surface in SPR
$\mathbf{t_0}$	$=$	$(t_{x0}, t_{y_0}, t_{z0})$	nominal position
$\mathbf{a_0}$	$=$	(a_0, b_0, c_0)	nominal orientation
$\mathbf{x}_{nom}(u, v)$	$=$	$\mathbf{x}(u, v, \mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0})$	nominal surface in nom. pos./orient.
u_{i0}, v_{i0}			surface coord. of the nominal points
$\tilde{\mathbf{x}}_\mathbf{i}$	$=$	$\mathbf{x}(u_{i0}, v_{i0}, \mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0})$	nominal points
$\hat{\mathbf{x}}_\mathbf{i}$	$=$	$(\hat{x}_i, \hat{y}_i, \hat{z}_i)$	measuring points
$\mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*$			bestfitted parameters
$\mathbf{\Delta p}, \mathbf{\Delta a}, \mathbf{\Delta t}$			parameter corrections
$\mathbf{x}_{sub}(u, v)$	$=$	$\mathbf{x}(u, v, \mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*)$	substitute surface (bestfitted)
$u_i{}^*, v_i{}^*$			surface coord. of footp.(after bestfit)
$\bar{\mathbf{x}}_\mathbf{i}$	$=$	$\mathbf{x}(u_i{}^*, v_i{}^*, \mathbf{p}^*, \mathbf{a}^*, \mathbf{t}^*)$	footpoints (after bestfit)
$\mathbf{\Delta u}_i, \mathbf{\Delta v}_i$			corrections of the surface coord.
$d(\mathbf{x})$			distance function
u_i^M, v_i^M	$=$	$u_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t}), v_i^M(\mathbf{p}, \mathbf{a}, \mathbf{t})$	surface coord. of the footpoints
u_i, v_i			actual surface coordinates
$\mathbf{x_i}(u_i, v_i, \mathbf{p}, \mathbf{a}, \mathbf{t})$			actual surface points
L_m	$=$	$\sqrt[m]{\sum_{i=1}^{n} d_i^m}$	L_m-norm

$\mathbf{D}$	$=$	$(d_1, \ldots, d_n)$	vector of all deviations
$\mathbf{\Delta x}_i$	$=$	$(\Delta x_i, \Delta y_i, \Delta z_i)$	vector of 3D-coord. diff.
$\mathbf{\Delta X}$	$=$	$(\mathbf{\Delta x}_1, \ldots, \mathbf{\Delta x}_n)$	vector of all 3D-coord. diff.
$\hat{\mathbf{X}}$	$=$	$(\hat{\mathbf{x_1}}, \ldots, \hat{\mathbf{x_n}})$	vector of all measuring points
$\tilde{\mathbf{X}}$	$=$	$(\tilde{\mathbf{x_1}}, \ldots, \tilde{\mathbf{x_n}})$	vector of all nominal points
$\bar{\mathbf{X}}$	$=$	$(\bar{\mathbf{x}}_\mathbf{1}, \ldots, \bar{\mathbf{x}}_\mathbf{n})$	vector of all footpoints
$\mathbf{U}$	$=$	$(u_1, v_1, \ldots, u_n, v_n)$	vector of all surface coordinates
$\mathbf{T}$	$=$	$(\mathbf{t}, \ldots, \mathbf{t})$	vector defined by the translation $\mathbf{t}$
$Q(\mathbf{a})$	$=$	$\bigoplus_{i=1}^{n} R(\mathbf{a})$	matrix defined by the rotation $R(\mathbf{a})$
$\mathbf{W}$	$=$	$(\mathbf{U}, \mathbf{p}, \mathbf{a}, \mathbf{t})$	vector of all unknowns
$\mathbf{X}(\mathbf{W})$	$=$	$(\mathbf{x_1}(\mathbf{W}), \ldots, \mathbf{x_n}(\mathbf{W}))$	vector of all surface functions
$\mathbf{W_0}$	$=$	$(\mathbf{U_0}, \mathbf{p_0}, \mathbf{a_0}, \mathbf{t_0})$	initial values of the unknowns
$\mathbf{\Delta W}$			vector of all corrections (1 step)
$\mathbf{\Delta W}^\star$			vector of all final corrections
$J(\mathbf{W})$			actual Jacobian
$\mathbf{X}'$	$=$	$(\mathbf{x}'_\mathbf{1}, \ldots, \mathbf{x}'_\mathbf{n})$	vector of all function values (SPR)
$\frac{\partial \mathbf{f}}{\partial \mathbf{e}}$	$=$	$(\frac{\partial f_i}{\partial e_j})$	sub-part of the Jacobian
Q_a	$=$	$\frac{\partial Q}{\partial a}$	$Q(\mathbf{a})$ derived with respect to a
Q_b	$=$	$\frac{\partial Q}{\partial b}$	$Q(\mathbf{a})$ derived with respect to b
Q_c	$=$	$\frac{\partial Q}{\partial c}$	$Q(\mathbf{a})$ derived with respect to c
R_a	$=$	$\frac{\partial R}{\partial a}$	$R(\mathbf{a})$ derived with respect to a
R_b	$=$	$\frac{\partial R}{\partial b}$	$R(\mathbf{a})$ derived with respect to b
R_c	$=$	$\frac{\partial R}{\partial c}$	$R(\mathbf{a})$ derived with respect to c
$\mathbf{a_i}$	$=$	(a_i, b_i, c_i)	angles of 3 consecutive Givens rotations
$\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$			3 orthog. vectors defined by 3 Givens rot.
$R(\mathbf{a_i})^T$	$=$	$\left[\begin{array}{ccc} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{array} \right]$	matrix of 3 consecutive Givens rotations
G			matrix for pre-triagonalization
P_e			permutation matrix for the equations
P_u			permutation matrix for the unknowns
$\mathbf{a_{kl}}$	$=$	$(a_{x_{kl}}, a_{y_{kl}}, a_{z_{kl}})$	polynomial coefficients
$p_{kl}(u, v, \mathbf{p})$			basic polynomials
$p_{kl_{(i)}}$	$=$	$p_{kl}(u_i, v_i, \mathbf{p})$	polynomial values
U	$=$	$\frac{\partial \mathbf{X}'}{\partial \mathbf{U}}$	'**U**-part' of the Jacobian
P	$=$	$\frac{\partial \mathbf{X}'}{\partial \mathbf{p}}$	'**P**-part' of the Jacobian
A	$=$	$Q^T \cdot \frac{\partial \mathbf{X}}{\partial \mathbf{a}}$	'**a**-part' of the Jacobian
T	$=$	$\frac{\partial \mathbf{T}}{\partial \mathbf{t}}$	'**T**-part' of the Jacobian
D	$=$	$Q^T \hat{\mathbf{X}} - \mathbf{X}'(\mathbf{U}, \mathbf{p}) - \mathbf{T}$	right hand vector
$\mathbf{x}_{(i)}$	$=$	$\mathbf{x}'(u_i, v_i, \mathbf{p})$	function values
$\mathbf{x}_{u_{(i)}}$	$=$	$\frac{\partial \mathbf{x}'}{\partial u}(u_i, v_i, \mathbf{p})$	values of u-derivatives
$\mathbf{x}_{v_{(i)}}$	$=$	$\frac{\partial \mathbf{x}'}{\partial v}(u_i, v_i, \mathbf{p})$	values of v-derivatives
$\mathbf{x}_{p_{j_{(i)}}}$	$=$	$\frac{\partial \mathbf{x}'}{\partial p_j}(u_i, v_i, \mathbf{p})$	values of $\mathbf{p}$ derivatives
$\mathbf{\Delta x}_{(i)}$	$=$	$R^T \hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}'(u_i, v_i, \mathbf{p}) - \mathbf{t}$	i-th difference value
$\mathbf{x}_{a_{(i)}}$	$=$	$R^T R_a \cdot \mathbf{x}'(u_i, v_i, \mathbf{p})$	i-th special value (a)
$\mathbf{x}_{b_{(i)}}$	$=$	$R^T R_b \cdot \mathbf{x}'(u_i, v_i, \mathbf{p})$	i-th special value (b)
$\mathbf{x}_{c_{(i)}}$	$=$	$R^T R_c \cdot \mathbf{x}'(u_i, v_i, \mathbf{p})$	i-th special value (c)

$P_{(XY)}$			'XY-part' of the matrix P_e
$P_{(Z)}$			'Z-part' of the matrix P_e
$P_{(U)}$			'**U**-part' of the matrix P_u
$G_{(XY)}$			'XY-part' of the matrix G
$G_{(Z)}$			'Z-part' of the matrix G
$\sigma_{\ 1\ \ \ldots}\sigma_{g+6}$			singular values
Σ	$=$	$diag(\sigma_{\ 1\ }, \ldots, \sigma_{g+6})$	diagonal matrix of the singular values
Σ^{+}			pseudo-inverse of the matrix Σ
σ_{max}			maximum singular value
$\epsilon_{machine}$			numerical accuracy of the machine
$\mathbf{e_i}$			vector to the i-th contact point
r_i			i-th probe radius
$\mathbf{R}$	$=$	$(r_1 \cdot \mathbf{e_1}, \ldots, r_n \cdot \mathbf{e_n})$	vector of all probe radius corrections
$\mathbf{n}(u, v, \mathbf{p})$			surface normal
$\mathbf{\Delta x^{\star}_{(i)}}$	$=$	$\mathbf{\Delta x_{(i)}} + R^T(\mathbf{a}) r_i \mathbf{e_i}$	reduced coordinate differences
$\mathbf{\Delta X^{\star}}$	$=$	$(\mathbf{\Delta x^{\star}_{(1)}}, \ldots, \mathbf{\Delta x^{\star}_{(n)}})$	vector of all reduced coord.diff.
$J_{(i)}$	$=$	$\frac{\partial(\mathbf{\Delta x^{\star}_{(i)}})}{\partial(\mathbf{\Delta x_{(i)}})}$	i-th Jacobian for the reduced diff.
E_i			characteristic of probe deflection
R_{E_i}			orientation of probe deflection
$\mathbf{R^{\star}}$	$=$	$(R_{E_1} E_1 \mathbf{e_1}, \ldots, R_{E_n} E_n \mathbf{e_n})$	modified vector of probe radius corr.
$\mathbf{s}$	$=$	$(\rho_x, \rho_y, \rho_z, \tau_x, \tau_y, \tau_z)$	'symmetry' vector
S	$=$	$S^{\star} \cdot S^{\star T}$	'symmetry' matrix
$S^{\star}$			matrix from decomposition
$\mathbf{c}$			'constraint' vector
C	$=$	$diag(\mathbf{c})$	'constraint' matrix
$\mathbf{f}$	$=$	$\mathbf{s}\ AND\ \mathbf{c}$	'freezing' vector
F	$=$	$F^{\star} \cdot F^{\star T}$	'freezing' matrix
$F^{\star}$			matrix from decomposition
$\mathbf{x'}^{(K)}(u, v, \mathbf{p})$			surf. function of the K-th surface
$\mathbf{p}^{(K)}$			dimensions of the K-th surface
$\mathbf{a}^{(K)}, \mathbf{t}^{(K)}$			pos./orient.of the K-th surface
$\mathbf{x'}$	$=$	$\mathbf{x} - \mathbf{r}$	coordinate transformation
ρ	$=$	$\sqrt{\|\mathbf{x'}\|^2 - z^2}$	cylinder coordinate (ρ-direct.)
z	$=$	$\mathbf{n}^T \cdot \mathbf{x'}$	cylinder coordinate (z-direct.)
$d(\mathbf{x}, \mathbf{r}, \mathbf{n}, \mathbf{p})$			distance function (cart. coord.)
$d(\rho, z, \mathbf{p})$			distance function (cyl. coord.)
x_r			random value in x
y_r			random value in y
r_r			random value in r

Curriculum Vitae

Personal Data:

Name: Sourlier
First name: David Michael
Date of birth: January 29th, 1959
Nationality: Swiss
Marital status: married, three children

Professional Training:

1965-71 Primary school in Birmensdorf (ZH)
1971-77 Secondary school 'Real- und Literar-Gymnasium Zürich-Freudenberg'
1977 'Federal Baccalaureate Type A', study in the Department of Mathematics and Physics of the Swiss Federal Institute of Technology (ETH) in Zürich
1978 First pre-diploma
1979 Second pre-diploma
1982 Diploma in physics; diploma thesis in the field of photo-acoustics at the Institute of Technical Physics of the ETH

Professional Career:

1980-82 Teaching physics in the junior college 'Gymnasium Zürich-Freudenberg'.
1982 Diploma thesis followed by a four month industrial project
dealing with the development and numerical simulation of an infrared optical device, patented for photoacoustical gas detection.
1983-89 Development engineer in the R&D-company 'Triatex International AG' in Zürich; Development of algorithms for the mathematical modelling, simulation, recipe calculation, quality control and calibration of added and subtracted colourings (colour metrics), since 1985 leader of the group 'Colour Metrics'.
1989-90 As contractor to the engineering office 'REMUTAS AG' in Wetzikon (ZH) collaboration in the project 'Networked Energy Management' of the company 'Stäfa Control Systems' (SCS) in Stäfa, Switzerland.
1990-95 Scientific assistant in the sector 'Dimensional Metrology' at the Institute of Machine Tools and Manufacturing of ETH.

Lebenslauf

Personalien:

Name: Sourlier
Vorname: David Michael
Geburtsdatum: 29.1.1959
Nationalität: Schweiz
Zivilstand: Verheiratet, 3 Kinder

Schulen:

1965-71 Primarschule in Birmensdorf (ZH)
1971-77 Real- und Literar-Gymnasium 'Zürich-Freudenberg'
1977 Eidgenössische Matura Typus A, Studienbeginn an der Abteilung für Mathematik und Physik der ETH-Zürich
1978 1.Vordiplom
1979 2.Vordiplom
1982 Schlussdiplom Physik und Diplomarbeit auf dem Gebiet der Photoakustik am Institut für 'Technische Physik' der ETH Zürich

Berufstätigkeit:

1980-82 Erteilen von Physik-Unterricht am Real- und Literar-Gymnasium 'Zürich-Freudenberg'.
1982 Diplomarbeit und viermonatiges Industrieprojekt; Entwicklung und numerische Simulation einer Infrarot-Optik, patentiert für photoakustische Gas-Detektion.
1983-89 Entwicklungsingenieur bei der Forschungs- und Entwicklungs-Firma 'Triatex International AG' in Zürich; Entwicklung von Algorithmen zur Modellierung, numerischen Simulation, Rezept-Vorausberechnung, Qualitätskontrolle und Einkalibrierung additiver und subtraktiver Farbmischungen (Farbmetrik), ab 1985 als Leiter des Bereichs 'Farbmetrik'.
1989-90 Im Auftrag des Ingenieur-Büros 'REMUTAS AG' in Wetzikon (ZH), Mitarbeit beim Projekt 'Vernetztes Energie-Management' der Firma 'Stäfa Control Systems' (SCS) in Stäfa.
1990-95 Wissenschaftlicher Mitarbeiter im Sektor 'Messtechnik' am 'Institut für Werkzeugmaschinen und Fertigung' an der ETH-Zürich.