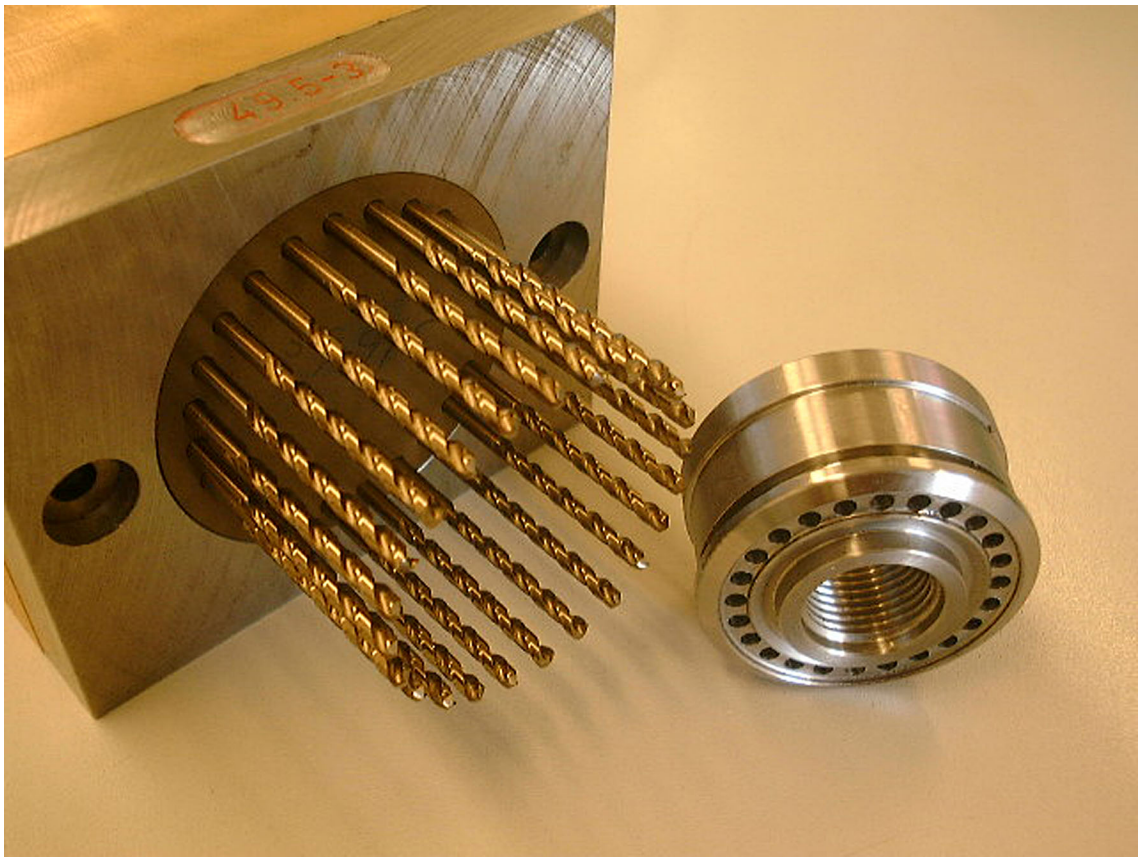
Computing the shape of flexible shafts

Thomas ten Cate



Institute for Mathematics and Computing Science

RuG

Bachelor thesis

Computing the shape of flexible shafts

Thomas ten Cate

Supervisors:
A.E.P. Veldman and H. Bekker
University of Groningen
Institute for Mathematics and Computing Science
P.O. Box 800
9700 AV Groningen
The Netherlands

May 2007

Abstract

This thesis concerns the shape of flexible shafts or rods in three dimensions, whose position and derivative are constrained at the endpoints, but whose length is allowed to change freely. This problem has a practical application in drill heads made by the customer, Koese Engineering.

Two models are presented based on the physical aspects of the system. The first model is more elegant and more widely applicable; the second was developed specifically for the customer's particular situation.

Neither system of equations can be solved analytically. A combination of several numerical algorithms is used to compute the shape of a shaft under absolute precision requirements.

Finally, the design and development of a software program is presented that allows the customer to quickly enter his data and extract the results.

Contents

1 Introduction **9**
1.1 Objectives . . . 9
1.2 Outline . . . 10

2 Geometric model **11**
2.1 Description of a drill head . . . 11
2.2 Geometric model . . . 12

3 Model of a bending shaft **17**
3.1 Parametrisation along the shaft . . . 17
3.1.1 Boundary conditions . . . 17
3.1.2 Elastic bending . . . 18
3.1.3 Forces and moments . . . 19
3.1.4 Differential equations . . . 20
3.2 Parametrisation in z . . . 21
3.2.1 Boundary conditions . . . 21
3.2.2 Differential equations . . . 21
3.2.3 The circular arc . . . 22

4 Algorithms **25**
4.1 Algorithm overview . . . 25
4.2 Error metric . . . 26
4.3 Numerical integration . . . 26
4.3.1 Integration step size . . . 27
4.4 Solving for the parameters . . . 28
4.5 Interpolation . . . 28

5 Requirements **31**
5.1 Functional requirements . . . 31
5.2 Nonfunctional requirements . . . 32

6 User interface design **33**
6.1 Menu bar . . . 33
6.2 Main dimensions . . . 34
6.3 Side view . . . 35
6.4 Top view . . . 35

6.4.1 Top bearings . . . 36
6.4.2 Shafts . . . 37
6.4.3 Guiding block holes . . . 37
6.4.4 Support bearing holes . . . 37

7 Technical design **39**
7.1 Language and tools . . . 39
7.2 Overview of software structuring . . . 40
7.2.1 Maths . . . 40
7.2.2 Model . . . 40
7.2.3 GUI . . . 41
7.2.4 Tests . . . 42

8 Tests and Results **43**
8.1 Comparison to the circle . . . 43
8.2 Comparison to accurate computations . . . 43
8.3 Real-world tests . . . 46

9 Conclusion **49**

A Pseudocode algorithms **53**
A.1 The Broyden algorithm . . . 53

Chapter 1

Introduction

Koese Engineering [1] is a company whose core business is the manufacturing of drills for industrial purposes. These drills are specifically designed for high-speed production, typically taking only a few seconds to drill dozens of holes at once.

Koese specialises in drilling many closely-spaced holes in one go. Because the distance between the holes is often smaller than the holes themselves, there is no room for gearwheels to drive each drill separately. Koese's patented solution for this problem is to drive the drills using flexible shafts that bend outwards from the block containing the drills, creating space for a large gearwheel in the centre that drives all shafts.

The flexible shafts will bend when pressure is put onto the drills. If a shaft bends too much, it will break. Therefore the shafts are supported at a few points (typically one to three) by a bronze plate with holes in it through which the shafts run.

1.1 Objectives

For years, Koese has constructed the aforementioned bronze plates simply by putting the shafts in the right position and then measuring where the holes in the bronze plates needed to be. This is a very time-consuming task.

The objective of this project is to analyse the problem mathematically and come up with a way to compute the positions of these holes in advance. To easily use the resulting algorithm, a computer program will be written that performs the necessary computations numerically. This program will have a graphical user interface that facilitates easy input and readout of data.

A secondary objective is to make the program into a useful aid when deciding upon the dimensions of the machine. The important factor here is the minimum bending radius that occurs in any shaft, because this corresponds to maximum curvature and therefore the highest probability of breaking.

1.2 Outline

We will start in chapter 2 by describing and modelling the situation. This chapter is relevant for all other chapters, so it should be read carefully.

The next two chapters are primarily of interest to the mathematically oriented reader. The equations necessary for computing the shaft shapes will be derived in chapter 3. The algorithms for numerically solving the equations are treated in chapter 4.

Three chapters on the development of the program follow; these are obviously primarily intended for readers from the field of computing science, most notably software engineering. The requirements are discussed in chapter 5. Chapter 6 presents the design of the user interface based upon these requirements, and chapter 7 discusses the implementation of this interface and the underlying algorithms.

Test results acquired with this program are given in chapter 8, leading up to the conclusion in chapter 9.

Chapter 2

Geometric model

The first section of this chapter, section 2.1, describes in detail how a Koese drill head is built up physically. In section 2.2 this is then translated into the mathematical domain by constructing a geometrical interpretation.

2.1 Description of a drill head

We will describe the structure of a Koese drill head from bottom to top. (Without loss of generality, we can assume that the machine drills holes downward. Most Koese machines do this, but it is not a requirement.) The word *drill head* in this context means everything the project is involved with, and so does not include, for example, the engine that drives the machine or the supports it is standing on.

Refer to figure 2.1 for an overview picture of a drill head designed to drill 18 holes at once.

At the bottom, there is the product to be drilled. Above that, the drills protrude from the *guiding block*, which is simply a block of metal with holes in it to guide the drills. The drills can slide freely through the guiding block in the axial (vertical) direction.

From the top of the guiding block the *flexible shafts* emerge. Their direction is initially exactly vertical, but they bend away from each other as we get higher up. In the following, when the word "shaft" is used without further specification, the flexible shaft is meant.

Along the way up, a shaft runs through the holes in a number of bronze plates called *support bearings*, mounted horizontally to support the shafts.

Where the flexible shaft ends at the top, it changes into a normal, *rigid shaft*, running further upwards, but not exactly vertical. The angle between the rigid shaft and the vertical is typically in the order of 10 degrees, and equal for all shafts. Sometimes there is no rigid shaft, but we model this by simply setting its length to zero.

At the top end of the rigid shaft, it is supported by a ball bearing called the *top bearing* (to distinguish it from the support bearing mentioned before). Somewhere along the rigid shaft, a small gearwheel is mounted. The rigid shafts are placed in such a way that these gearwheels are on a circle. In the centre of this circle is a large gearwheel, that drives the small gearwheels on all rigid shafts simultaneously.

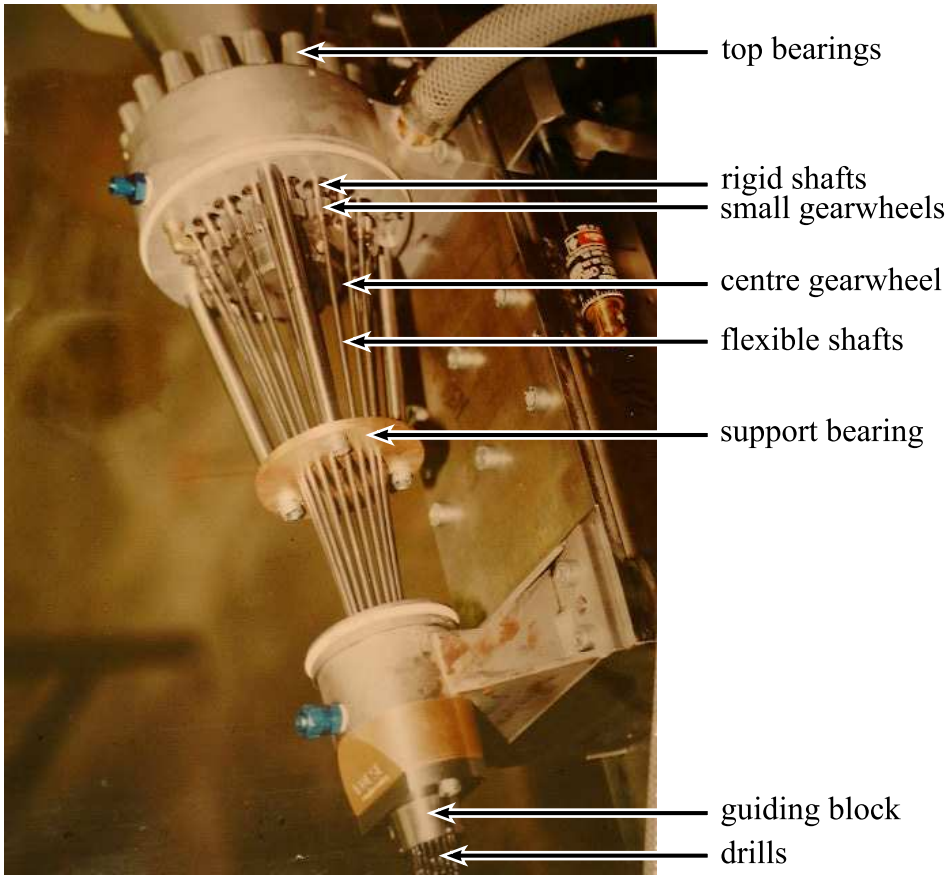


Figure 2.1: A Koese drill head with 18 shafts and one support bearing.

2.2 Geometric model

In this section, the coordinate system will be described and several important constants will be introduced that are used throughout the rest of the document.

As we are not concerned with the drills or the guiding block at the bottom, we place the origin of our coordinate system at the top of the guiding block, where the flexible shafts start. The z-axis runs upward from there, whereas the x-axis can be thought of to run to the right and the y-axis to the back. A side view in the direction of the positive y-axis is given in figure 2.2.

H is the *height* of the drill head, measured along the z-axis, but note that this includes only the part with the flexible shafts.

α is the angle between the rigid shaft and the z-axis, and is therefore called the *rigid shaft angle*. It also determines the tangent at the top of the flexible shaft.

R is the so-called *pure bending radius* of the shafts. It follows from H and α, because $R = H/\sin\alpha$. If the shape of a shaft follows a circle, which indeed does happen in the 'ideal' case as we will see later, then R is the radius of this circle.

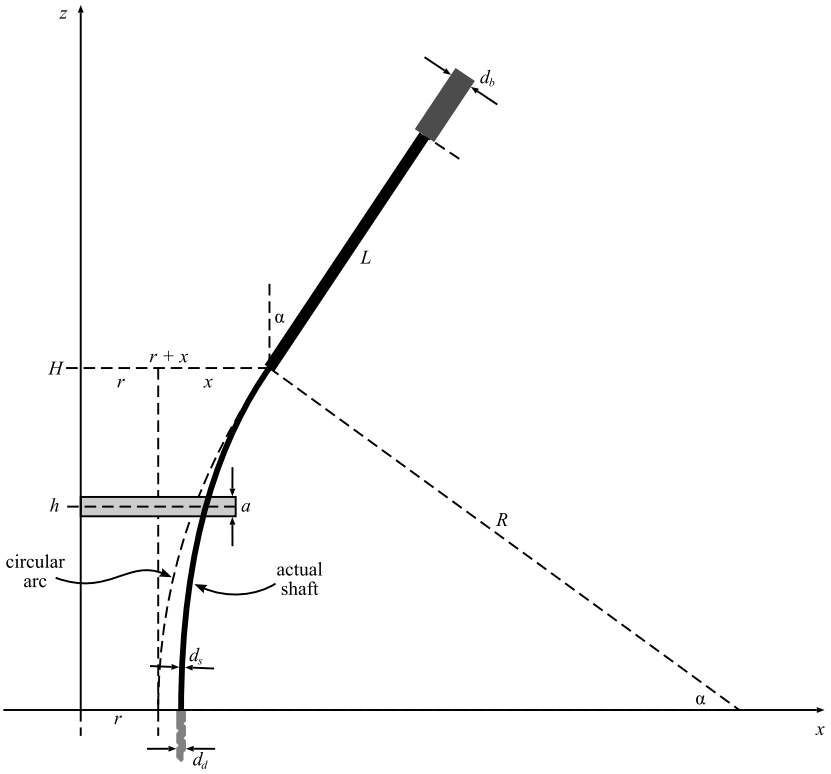


Figure 2.2: A schematic model of a drill head. One shaft is shown that lies in the xz-plane.

x is the *fan radius*, and can be computed as $x = R - R\cos\alpha$. It is the distance between the bottom and the top of a circular arc, projected onto the xy-plane, and therefore amounts to the space gained by using flexible shafts.

L is the length of the rigid shaft. It is 0 in case there are no rigid shafts.

r is the so-called *average radius*. It is not actually any average in the strict sense. The name 'average radius' was chosen because r is usually chosen such that it is close to the average distance of a hole to the origin, in order to minimise the distance from the bottom of a shaft to the circle defined by r.

h is the height of a support bearing, measured to its centre. We discuss all support bearings independently of each other, so an index (e.g. h_1, h_2) is not needed.

a is the thickness of a support bearing.

d_b is the diameter of the top bearings, again equal for all bearings.

d_s is the diameter of the flexible shaft, and can be assumed to be equal for all flexible shafts. It also determines the size of the holes in the support bearing.

d_d is the diameter of the drills, and therefore also of the drilled holes.

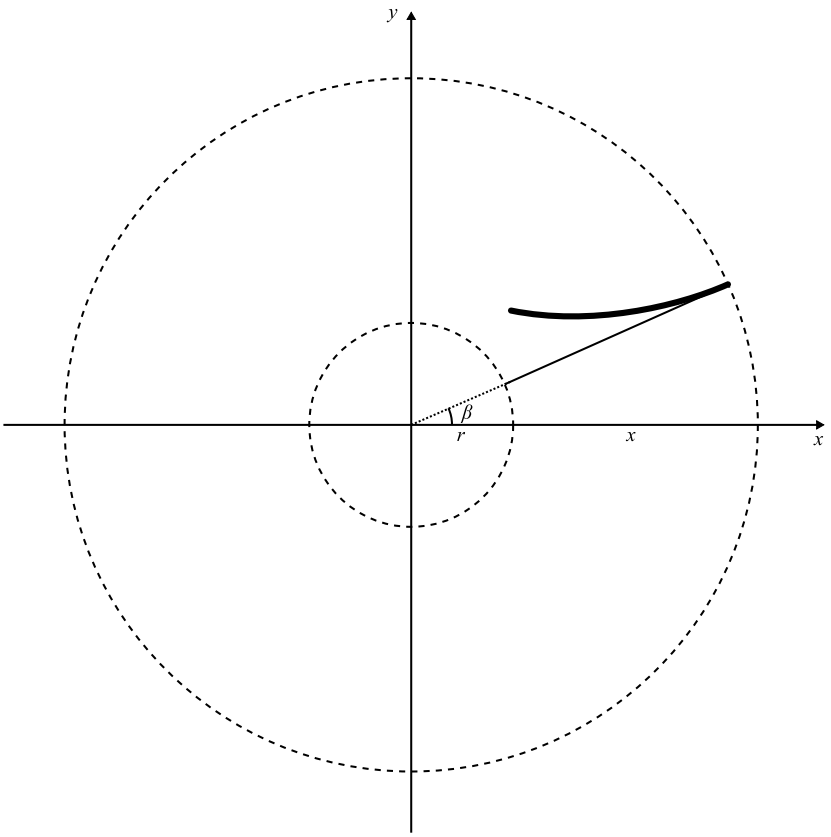


Figure 2.3: The drill head model seen from the top. The rigid shaft and top bearing are not shown.

The top view is shown in figure 2.3. Here we see a shaft that does not lie in the xz-plane. Note that r and $r + x$ sweep out a circle. The tops of the flexible shafts are all on the circle with radius $r + x$.

β is the angle between the x-axis and the tangent vector at the top of the shaft. It is different for each shaft. In practical situations, the shafts are often equally spaced, e.g. $\beta = 0°, 15°, 30°, \ldots$.

Where the shaft intersects a support bearing, it does so at a certain point (x, y, h) in a certain direction, defined in the spherical coordinates seen in figure 2.4.

γ is the *azimuth* at the point of intersection, defined to be the counterclockwise angle from the x-axis to the projection of the tangent vector onto the xy-plane.

δ is the *elevation*[1] at the point of intersection, which is the angle between the tangent and the z-axis.

[1]The terms 'azimuth' and 'elevation' are borrowed from astronomy. However, in astronomy the word 'elevation' usually refers to the angle between the vector and the xz-plane.

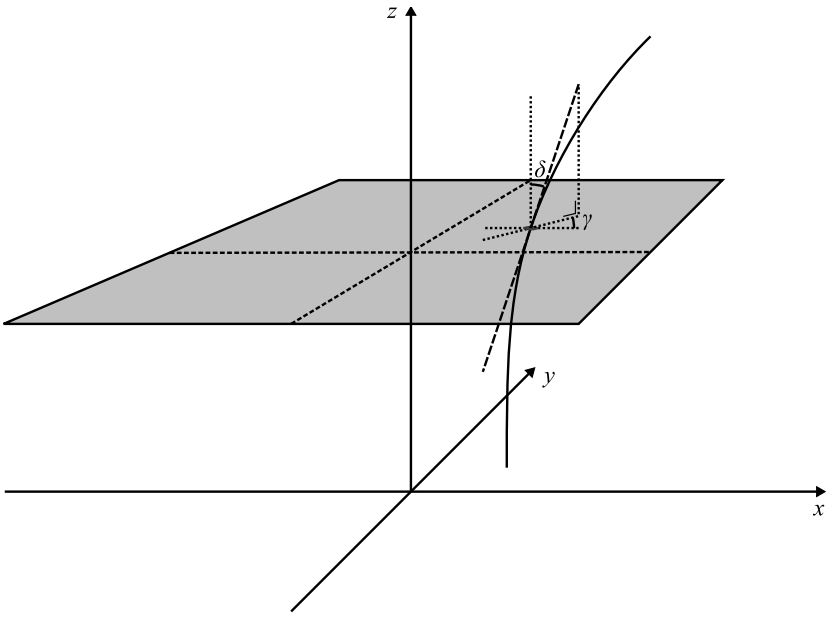


Figure 2.4: A shaft intersecting a support bearing.

Chapter 3

Model of a bending shaft

In order to determine the placement of the holes in a support bearing, we need to know the shape that a given shaft will assume. In this section, physical modelling will be used to result in a set of non-linear differential equations that describe the shape of the shaft, given the position and tangent at its endpoints.

The required system of differential equations is essentially derived in [2], and section 3.1 is mostly a rephrasing of this derivation. The result is a six-dimensional system. However, for this specific purpose, it turns out to be possible to reduce this to a four dimensional system, at the cost of slightly more complex equations. This result is presented in section 3.2.

3.1 Parametrisation along the shaft

We describe the shape of the shaft as a curve c of length L_b in three-dimensional space:

$$\begin{aligned} c : [0, L_b] &\longrightarrow \mathbb{R}^3, \\ c : t &\longmapsto \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}. \end{aligned}$$

Because t parametrises the length along the shaft, the derivative of c with respect to t must have magnitude 1:

$$\|c'(t)\| = \sqrt{x'(t)^2 + y'(t)^2 + z'(t)^2} = 1. \tag{3.1}$$

3.1.1 Boundary conditions

In order to fit into the drill head, certain conditions on c must be satisfied. The start and end positions are fixed, as is the direction (tangent, or simply first derivative) at the start and end point. If the hole to be drilled is at (b_x, b_y), then observing figures 2.2 and 2.3 and applying

some trigonometry, we get the following boundary conditions:

$$\begin{cases} c(0) = \begin{pmatrix} b_x \\ b_y \\ 0 \end{pmatrix}, & c(L_b) = \begin{pmatrix} (r+x)\cos\beta \\ (r+x)\sin\beta \\ H \end{pmatrix}, \\ c'(0) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, & c'(L_b) = \begin{pmatrix} \sin\alpha\cos\beta \\ \sin\alpha\sin\beta \\ \cos\alpha \end{pmatrix}. \end{cases} \tag{3.2}$$

Note that the length L_b of the shaft is *not* known.

3.1.2 Elastic bending

The shaft takes on the shape that it does because forces and bending moments are exerted on it. We assume that the shaft is very thin compared to its length, that its cross section is constant and that it is made of a uniform material. All of these assumptions hold very well in practice. This allows us to use the Bernoulli-Euler equation[1], that is often used in engineering disciplines, as the basic model of bending:

$$K(t) = -\frac{M(t)}{EI}. \tag{3.3}$$

Here, $M(t)$ is the bending moment of the shaft at a the point $c(t)$, and $K(t)$ is the curvature, here defined as

$$K(t) = \frac{c'(t) \times c''(t)}{\|c'(t)\|^3} = c'(t) \times c''(t). \tag{3.4}$$

Note that both M and K are vectors. E and I are physical parameters (the elasticity modulus and the moment of inertia, respectively). However, these do not influence the shape of the shaft, so we set both to 1, resulting in

$$K(t) = -M(t). \tag{3.5}$$

Because $\|c'(t)\| = 1$ by (3.1), we know that c' does not change in magnitude. This implies that the second derivative $c''(t)$ is perpendicular to $c'(t)$ at every point:

$$\begin{aligned} c'(t) \cdot c''(t) &= \begin{pmatrix} x'(t) \\ y'(t) \\ z'(t) \end{pmatrix} \cdot \begin{pmatrix} x''(t) \\ y''(t) \\ z''(t) \end{pmatrix} \\ &= x'(t)x''(t) + y'(t)y''(t) + z'(t)z''(t) \\ &= \frac{\mathrm{d}}{\mathrm{d}t}(\frac{1}{2}x'(t)^2 + \frac{1}{2}y'(t)^2 + \frac{1}{2}z'(t)^2) \\ &= \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}(x'(t)^2 + y'(t)^2 + z'(t)^2) \\ &= \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}1 \\ &= 0. \end{aligned}$$

[1] In the literature, this equation is often found in different forms, for example in two dimensions and parametrised in the x-coordinate. The equation is presented here in its most generic form, working in either two or three dimensions and not constraining one spatial coordinate to be a function of another.

To 'invert' the cross product (3.4), we use the following lemma: if $p = q \times r$ and $q \perp r$, then $\|q\|^2 r = p \times q$. The proof is neither difficult nor interesting and can be found in the literature. Filling in $p = K(t)$, $q = c'(t)$ and $r = c''(t)$, and using $\|c'(t)\| = 1$, gives us

$$c''(t) = K(t) \times c'(t),$$

and combining with (3.4) yields:

$$c''(t) = -M(t) \times c'(t) = c'(t) \times M(t). \tag{3.6}$$

3.1.3 Forces and moments

We imagine that the shaft is first fixed at the top (that is, at the bottom of the rigid shaft) and allowed to float freely. Barring gravity, it will of course be straight. To force the shaft into its desired position, a force and bending moment are exerted on the bottom of the shaft, where it intersects the $z = 0$ plane; see figure 3.1. Note that the eventual length of the shaft is not known: it is allowed to shift freely at the bottom in the z direction.

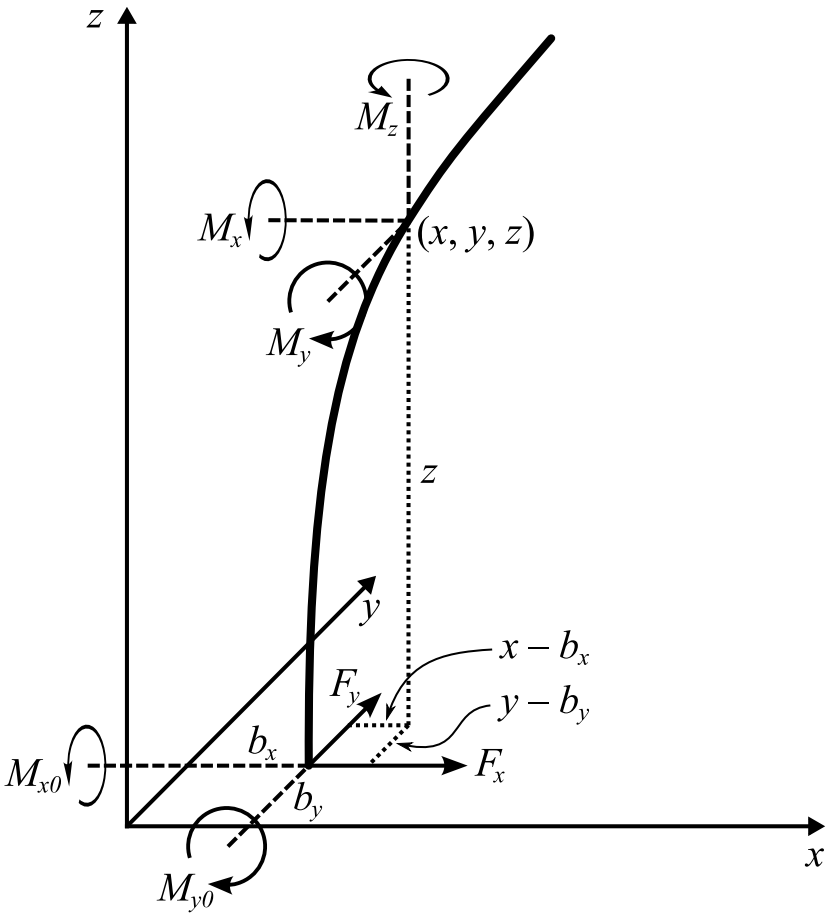


Figure 3.1: The forces and moments acting on the shaft.

The force is split into scalar components F_x and F_y. There is no force in the z direction, because it is assumed that no pressure is being put on the drills. Similarly, the moment is

split into M_{x0} and M_{y0}. There is no moment around the z-axis because there is no torsion being applied to the drills. At this point, F_x, F_y, M_{x0} and M_{y0} are unknown. These need to be chosen in such a way that the bottom of the shaft ends up in the desired position with the desired tangent.

The bending moment 'felt' at a point $c(t)$ along the shaft is the sum of the moment at the bottom and the moment resulting from the force applied at the bottom. The moment resulting from this force is found by multiplying the force with the length of its lever arm, which is equal to $z(t)$ in this case. This results in a non-zero moment around the z-axis as well; see again figure 3.1.

From this, taking care to get the signs correct, and leaving out the dependency on t for readability, we deduce that the moment acting on any part of the shaft is given by

$$M = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \begin{pmatrix} M_{x0} + zF_y \\ M_{y0} - zF_x \\ (y - b_y)F_x - (x - b_x)F_y \end{pmatrix}. \tag{3.7}$$

3.1.4 Differential equations

Plugging (3.7) into (3.6), we find

$$\begin{aligned} \begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} &= \begin{pmatrix} y'M_z - z'M_y \\ z'M_x - x'M_z \\ x'M_y - y'M_x \end{pmatrix} \\ &= \begin{pmatrix} y'((y - b_y)F_x - (x - b_x)F_y) - z'(M_{y0} - zF_x) \\ z'(M_{x0} + zF_y) - x'((y - b_y)F_x - (x - b_x)F_y) \\ x'(M_{y0} - zF_x) - y'(M_{x0} + zF_y) \end{pmatrix}. \end{aligned}$$

This is a three-dimensional system of second-order ordinary differential equations. It can be rewritten to a six-dimensional system of first-order ordinary differential equations in the standard way:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}x &= x', \\ \frac{\mathrm{d}}{\mathrm{d}t}y &= y', \\ \frac{\mathrm{d}}{\mathrm{d}t}z &= z', \\ \frac{\mathrm{d}}{\mathrm{d}t}x' &= y'((y - b_y)F_x - (x - b_x)F_y) - z'(M_{y0} - zF_x), \\ \frac{\mathrm{d}}{\mathrm{d}t}y' &= z'(M_{x0} + zF_y) - x'((y - b_y)F_x - (x - b_x)F_y), \\ \frac{\mathrm{d}}{\mathrm{d}t}z' &= x'(M_{y0} - zF_x) - y'(M_{x0} + zF_y). \end{cases}$$

These equations contain the four unknowns F_x, F_y, M_{x0} and M_{y0}. In the boundary conditions (3.2) a fifth unknown, the length of the shaft L_b, is stated. It would appear that we are dealing with a system of five variables and six constraints, yet it turns out to have a solution. The reason is that the sixth constraint is satisfied automatically because of (3.1): our solution is parametrised in the length along the shaft. Therefore, if the end conditions for x' and y' are satisfied, then, by (3.1), so is the end condition for z'.

In general, it is not possible to solve these equations analytically. An approach to a numerical solution is briefly described in [2]; however, matters are complicated by the extra constraint (3.1) and the fact that the integration length L_b is unknown. In the next section, a simpler system is derived which has only four dimensions and no constraint on the parametrisation, and eliminates L_b entirely.

3.2 Parametrisation in z

Because in practical situations, a shaft will never 'double back' onto itself, it is safe to assume that every plane perpendicular to the z-axis is intersected at most once. In other words, the shape of the shaft can also be seen as a function mapping a z-coordinate between 0 and H to its corresponding x- and y-coordinates. For consistency, we retain the third coordinate z but keep in mind that this is simply the identity function:

$$\begin{aligned} c : [0, H] &\longrightarrow \mathbb{R}^3, \\ c : z &\longmapsto \begin{pmatrix} x(z) \\ y(z) \\ z \end{pmatrix}. \end{aligned}$$

Also note

$$c'(z) = \begin{pmatrix} x'(z) \\ y'(z) \\ z'(z) \end{pmatrix} = \begin{pmatrix} x'(z) \\ y'(z) \\ 1 \end{pmatrix}, \qquad c''(z) = \begin{pmatrix} x''(z) \\ y''(z) \\ z''(z) \end{pmatrix} = \begin{pmatrix} x''(z) \\ y''(z) \\ 0 \end{pmatrix},$$

and therefore

$$\|c'(z)\| = \sqrt{x'(z)^2 + y'(z)^2 + 1},$$

which is no longer constant as in (3.1). This renders some of its corollaries invalid.

3.2.1 Boundary conditions

The boundary conditions (3.2) take on a different form in the new parametrisation, which can again easily be derived with some trigonometry:

$$\begin{cases} c(0) = \begin{pmatrix} b_x \\ b_y \\ 0 \end{pmatrix}, & c(H) = \begin{pmatrix} (r+x)\cos\beta \\ (r+x)\sin\beta \\ H \end{pmatrix}, \\ c'(0) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, & c'(H) = \begin{pmatrix} \tan\alpha\cos\beta \\ \tan\alpha\sin\beta \\ 1 \end{pmatrix}. \end{cases} \tag{3.8}$$

3.2.2 Differential equations

The Bernoulli-Euler equation (3.3) remains intact, because both K and M are reparametrised. The curvature K, however, can no longer be written in the elegant form of (3.4), but instead

becomes

$$\begin{aligned} K(z) &= \frac{c'(z) \times c''(z)}{\|c'(z)\|^3} \\ &= \frac{c'(z) \times c''(z)}{(x'(z)^2 + y'(z)^2 + 1)^{\frac{3}{2}}} \\ &= (x'(z)^2 + y'(z)^2 + 1)^{-\frac{3}{2}} \begin{pmatrix} y'(z)z''(z) - z'(z)y''(z) \\ z'(z)x''(z) - x'(z)z''(z) \\ x'(z)y''(z) - y'(z)x''(z) \end{pmatrix} \\ &= (x'(z)^2 + y'(z)^2 + 1)^{-\frac{3}{2}} \begin{pmatrix} -y''(z) \\ x''(z) \\ x'(z)y''(z) - y'(z)x''(z) \end{pmatrix}. \end{aligned}$$

Combining this with $K(z) = -M(z)$ from (3.5) and plugging in M from (3.7) gives us

$$(x'(z)^2 + y'(z)^2 + 1)^{-\frac{3}{2}} \begin{pmatrix} -y''(z) \\ x''(z) \\ x'(z)y''(z) - y'(z)x''(z) \end{pmatrix} = - \begin{pmatrix} M_{x0} + zF_y \\ M_{y0} - zF_x \\ (y - b_y)F_x - (x - b_x)F_y \end{pmatrix}.$$

This can again be rewritten into a system differential equations, by solving it for y'' and z''. We do not need the third component, so we leave it out on both sides.

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}x &= x', \\ \frac{\mathrm{d}}{\mathrm{d}t}y &= y', \\ \frac{\mathrm{d}}{\mathrm{d}t}x' &= (-M_{y0} + zF_x)(x'(z)^2 + y'(z)^2 + 1)^{\frac{3}{2}}, \\ \frac{\mathrm{d}}{\mathrm{d}t}y' &= (M_{x0} + zF_y)(x'(z)^2 + y'(z)^2 + 1)^{\frac{3}{2}}. \end{cases} \tag{3.9}$$

These are the final differential equations that will be used in the program. The four unknown parameters are F_x, F_y, M_{x0} and M_{y0}. The integration length is now known to be H and there are no secondary constraints.

These equations are not known to be analytically solvable. They can easily be linearised by neglecting the $x'(z)$ and $y'(z)$ terms, which will be small in practical cases. The linear system can then be solved analytically and results in a third-degree polynomial for both x and y. This is one of the approaches taken in [2] to estimate the maximum curvature, producing results with an error of a few percent.

For this application however, a more precise result is required. Therefore, the nonlinear system will have to be solved using numerical methods, which are described in the next chapter. First, however, we will discuss a situation which does allow an exact solution.

3.2.3 The circular arc

As might be expected from the law of conservation of energy, in this case bending energy, the shaft will prefer a shape with as little bending as possible. If its constraints allow, it will take on the shape of a perfectly circular arc. In this section we will show that this is indeed a solution of the aforementioned equations.

For simplicity, we assume that the shaft is in the xz-plane, to the right of the z axis, like the one in figure 2.2. This assumption is equivalent to $\beta = 0$. The proof can be extended to

the general case, but this only leads to more complex computations without delivering any more insights. Such a circular shaft in the xz plane is described by

$$\begin{aligned} x(z) &= r + R - \sqrt{R^2 - z^2}, \\ y(z) &= 0. \end{aligned}$$

To allow a perfectly circular shape, the boundary conditions need to be just right. This is realised by setting

$$\begin{aligned} b_x &= r, \\ b_y &= 0. \end{aligned}$$

See again figure 2.2.

The y part is trivial: everything remains 0 everywhere. We will therefore focus on the x part. We start by differentiating:

$$x'(z) = \frac{z}{\sqrt{R^2 - z^2}}.$$

Now we can check without much difficulty that the boundary conditions in (3.8) are indeed satisfied.

To verify that x conforms to the differential equations (3.9) we differentiate x' again, using the product rule:

$$\begin{aligned} x''(z) &= \frac{1}{\sqrt{R^2 - z^2}} + \frac{z^2}{(R^2 - z^2)^{\frac{3}{2}}} \\ &= \frac{1}{\sqrt{R^2 - z^2}} \left(\frac{z^2}{R^2 - z^2} + 1 \right) \\ &= \frac{1}{R} \sqrt{\frac{R^2}{R^2 - z^2}} \left(\frac{z^2}{R^2 - z^2} + 1 \right) \\ &= \frac{1}{R} \left(\frac{z^2}{R^2 - z^2} + 1 \right)^{\frac{1}{2}} \left(\frac{z^2}{R^2 - z^2} + 1 \right) \\ &= \frac{1}{R} \left(\frac{z^2}{R^2 - z^2} + 1 \right)^{\frac{3}{2}}. \end{aligned}$$

Now we note that $\frac{z^2}{R^2 - z^2} = x'(z)^2$ and $0 = y'(z)^2$, so we get

$$x''(z) = \frac{1}{R} \left(x'(z) + y'(z) + 1 \right)^{\frac{3}{2}}.$$

To make this satisfy (3.9), we choose $M_{y0} = -\frac{1}{R}$ and $F_x = 0$, resulting in

$$x''(z) = (-M_{y0} + zF_x) \left(x'(z) + y'(z) + 1 \right)^{\frac{3}{2}},$$

which proves that a circular arc is indeed a solution.

Chapter 4

Algorithms

Eventually, the customer is interested in two things: the placement of the holes in a support bearing, and the minimum bending radius of a shaft. For estimating the minimum bending radius, a simple formula has been produced [2] which works very well in practice. The focus of this project will therefore be on computing the positions of the holes in the support bearings, and the angle under which these need to be drilled. These values correspond to $x(h)$, $y(h)$, $x'(h)$ and $y'(h)$ in the modelling of section 3.2.

4.1 Algorithm overview

If the values of the four parameters F_x, F_y, M_{x0} and M_{y0} are given, then equations (3.9) can be solved using numerical integration. This will give the position and tangent at certain points along the shaft, and in particular at the top end of the shaft, $z = H$. The numerical integration is discussed in detail in section 4.3.

We then need to determine the values of the four parameters such that these position and tangent match those of the boundary conditions (3.8). Essentially, we are dealing with a function

$$\begin{aligned} S : \mathbb{R}^4 &\longrightarrow \mathbb{R}^4, \\ S : (F_x, F_y, M_{x0}, M_{y0}) &\longmapsto (x(H), y(H), x'(H), y'(H)), \end{aligned}$$

where the value of S is determined by the numerical integration process, and we want to solve

$$S(F_x, F_y, M_{x0}, M_{y0}) = (x_e, y_e, x'_e, y'_e), \tag{4.1}$$

where x_e, y_e, x'_e, y'_e are the constants from the boundary condition (3.8) at H. This solving can be done with an off-the-shelf algorithm. This algorithm is discussed in section 4.4.

Note that the combination of a numerical integrator and an equation solver results in what is essentially a so-called 'shooting method': fire a 'shot' for certain values of the parameters, see where we end up, adjust the parameters and try again, until the shot is on (or close enough to) the target.

Once the values of these parameters have been determined we can run the numerical integration one more time to get a list of points along the shaft. In general there will be no

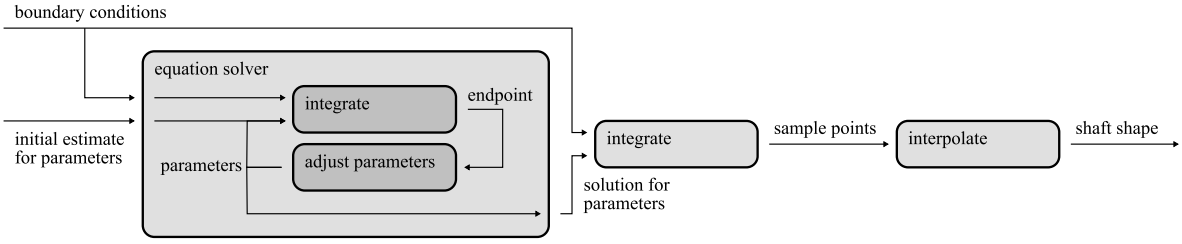


Figure 4.1: A schematic overview of the algorithm that computes the shaft shape.

point at exactly the height h where we want to determine the values of x, y, x' and y', so we will need some form of interpolation. This is treated in section 4.5.

Figure 4.1 gives a schematic overview of the algorithm discussed above.

4.2 Error metric

Numerical computations like this will never be precisely accurate. A way to measure the error in the result is needed. As an error metric, we will use the Euclidian distance between the real location $(x^*(h), y^*(h), h)$ and the computed location $(x(h), y(h), h)$ of the hole. Both are at a height of $z = h$, so this distance is computed in the horizontal plane. Given a value for the tolerance τ, we want to achieve the following:

$$\sqrt{(x(h) - x^*(h))^2 + (y(h) - y^*(h))^2} \leq \tau.$$

In practice, τ will be around 0.1 mm for H in the order of 10^3 mm. It is possible to define τ as a percentage of H if desired.

Rather arbitrarily, we use the same measure and the same τ for the derivative. This provides our second constraint:

$$\sqrt{(x'(h) - x^{*\prime}(h))^2 + (y'(h) - y^{*\prime}(h))^2} \leq \tau.$$

4.3 Numerical integration

The most well-known and very frequently used algorithm for solving first-order ordinary differential equations is the fourth-order Runge-Kutta (RK4) method [3, p. 278], which seems like a good choice at first sight.

However, the RK4 algorithm uses a fixed step size which has to be specified in advance. This may not be a good idea if we want precise control over the error. The adaptive Runge-Kutta-Fehlberg (RKF) method [3, pp. 285–286], on the other hand, incorporates a minimum and a maximum step size, and will adaptively adjust the actual step size used. This step size is based on an estimate of the local truncation error, produced by running an RK4 and an RK5 method side by side.

On the other hand, the differential equations that we are dealing with are quite smooth, and derivatives are relatively small. Experiments with RKF revealed that it nearly always decides to use the maximum allowed step size. The extra overhead produced by the RK5 method is therefore worthless. For this reason, eventually RK4 was chosen for the numerical integration procedure.

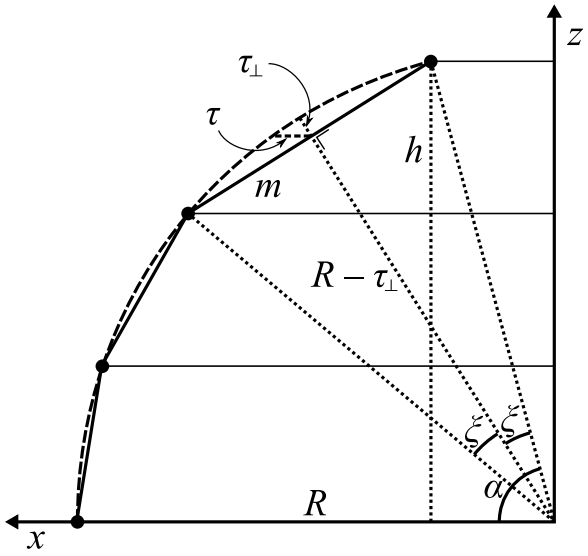


Figure 4.2: A sketch to determine the step size for the numerical integration.

4.3.1 Integration step size

For determining an appropriate step size for the RK4 algorithm, we assume that the resulting points are on a circle. In practice they will indeed be close to a circular arc. We further assume that they are interpolated in a linear fashion, which, as we will see later, is a worst-case scenario.

Now we can use figure 4.2 to determine the maximum step size. Because the points are distributed evenly on the z-axis, the largest error will occur at the top, in the last segment, where the slope of the shaft is largest with respect to the z axis. Because this is precisely the error that we want to control, we call this distance τ. We assume that the largest error occurs close to the midpoint of the segment, as can be seen in the figure, so we define m as half the length of this segment. We define $\tau_\perp$ to be the distance from the midpoint to the circle.

From the Pythagorean theorem, followed by some trigonometry, we get:

$$\begin{aligned}\tau_\perp &= R - \sqrt{R^2 - m^2} \\ &= R - \sqrt{R^2 - (R\sin\xi)^2} \\ &= R - R\sqrt{1 - \sin^2\xi}.\end{aligned}$$

Solving for ξ yields:

$$\xi = \operatorname{asin}\sqrt{\frac{2\tau\cos\alpha}{R} - \frac{\tau^2\cos^2\alpha}{R^2}}.$$

Some more trigonometry gives us h:

$$h = R\sin\alpha - R\sin(\alpha - 2\xi),$$

into which we can plug in ξ to produce

$$h = R\sin\alpha - R\sin\left(\alpha - 2\,\mathrm{asin}\sqrt{\frac{2\tau\cos\alpha}{R} - \frac{\tau^2\cos^2\alpha}{R^2}}\right),$$

which is the step size to use for the RK4 algorithm.

4.4 Solving for the parameters

Now that we have a way to map the four parameters F_x, F_y, M_{x0} and M_{y0} to the resulting situation at the endpoint $x(H)$, $y(H)$, $x'(H)$, $y'(H)$, we need a way to solve the nonlinear system of equations (4.1), thereby sending the endpoint towards its desired position. A numerical equation solver has to be used here.

Because the derivative of S is unknown, Newton's method is useless here. (Note that even S itself cannot be written analytically, because it is the result of a numerical process.) In one-dimensional systems this problem can be solved by using the secant method. There exists a generalization of the secant method to higher dimensions, known as *Broyden's method*. Please refer to A.1 for the pseudocode of this algorithm. Instead of requiring the exact derivative (Jacobian matrix), it employs an estimate, which is updated in each iteration. Broyden's algorithm is the one that we will use.

The algorithm given in [3] was slightly modified to be able to meet absolute precision requirements, in terms of τ, instead of stopping as soon as the change in the result drops below a certain threshold. For details, again see A.1.

In Step 9 of this algorithm, a matrix A is updated using a term multiplied by $\frac{1}{p}$. However, the algorithm does not mention what to do in case $p = 0$. One case where p can become 0 is when the first estimate is already correct. This happened in one of the unit tests, where the shape of a straight shaft along the z-axis is computed. Skipping the update of A is a correct approach for these situations, but might not handle cases where p becomes 0 for some other reason. As the book [3] does not mention anything about this, it will probably be very rare or even impossible.

As the error in the endpoint of the RK4 output will usually be an ascending function of the integration length z, we could assume that the largest error is in the last point, where we can control it. If this were always true, we could control the global error by setting the tolerance of the Broyden algorithm to τ, resulting in errors less than τ everywhere else along the shaft. However, sometimes the largest error turns out to be somewhere else. Because in practice the algorithm turns out to converge very fast, we can afford to set its tolerance to $\frac{\tau}{10}$. This should ensure that the Broyden solver does not become the precision bottleneck of the algorithm.

4.5 Interpolation

The series of points resulting from the numerical integration in section 4.3 can be interpolated in a piecewise linear fashion, that is, with straight line segments between the points. However, the curve that represents the shape of a shaft is very smooth. It can therefore be expected that an interpolation using some form of polynomial will yield much more accurate results.

A Hermite spline is a commonly used method of interpolating data points where the derivative is known in every point. Some confusion exists over the precise definition of "Hermite spline"; we will therefore use the more concise term "piecewise cubic spline". Because the numerical integration process produces a numerical value for the derivative in every point, using piecewise cubic interpolation is the obvious choice. To each interval between two points, a cubic polynomial (four degrees of freedom) is fitted that matches the values and the derivatives in both endpoints (four constraints). Unlike many other spline fitting algorithms, the interpolation in one interval does not depend on any information outside its interval, making this interpolation very robust. The linearised version of the problem also produces a third-order polynomial as the solution (see section 3.2.2 and [2]), further supporting the choice for a cubic interpolating function.

Cubic splines are usually created in two dimensions, but they can easily be extended to three dimensions by working in the xz and yz projection planes and producing polynomials for $x(z)$ and $y(z)$ separately.

Because a support bearing has a certain thickness, the question remains whether to compute the hole position at the top, in the centre or at the bottom of the bearing. The arbitrary choice has been made to compute the position and angle in the centre of the bearing, thereby defining a cilindrical hole. The curvature of the shaft over this relatively short distance is neglected.

Chapter 5

Requirements

By this point, an algorithm has been developed that can be used to compute the positions of the holes that need to be drilled in the support bearings. The next step is to design a program that allows the user to work efficiently with this algorithm. This chapter highlights the requirements that the program must meet, along with the reason that these requirements were posed.

The requirements are labelled. These labels are referred to in the subsequent sections on design.

5.1 Functional requirements

The end goal of the program is to produce a view on the pattern of holes to be drilled in the support bearings. For this, the following data are needed:

FR1 — the main dimensions of the drill head,

FR2 — the location (and diameter) of the holes to be drilled,

FR3 — the location (and diameter) of the top bearings,

FR4 — the assignment between shafts and holes,

FR5 — the diameter of shafts,

FR6 — the location and thickness of the support bearings.

The UI must allow the user to input each of these. (The diameters in braces are not strictly necessary, but are included to give the user a more complete view of the situation, to prevent creating overlapping holes or top bearings.)

The UI must be able to produce a view of the output, consisting of the patterns of holes in each of the support bearings, in three formats:

FR7 — on-screen in a graphical form, to check whether everything "looks okay",

FR8 — printable in a tabular form, to input the data manually into some machine,

FR9 — digital in AutoCAD DXF format [4], to be imported into a CAD program like SolidWorks [5].

Finally, the program must support the following auxiliary features, needed to be able to use the program efficiently:

FR10 — saving a design to a file and loading it back from the file,

FR11 — configuring the tolerance τ as described before.

5.2 Nonfunctional requirements

Below is a selection of the most important nonfunctional requirements for the program, along with their rationale.

NFR12 — Because the program will be used on an irregular basis, and will often not be used for months, the interface must have a high degree of (re)discoverability.

NFR13 — The user interface must respond quickly to the user's actions to keep the user from performing the action again.

NFR14 — Incorrect output will result in lost time, money and effort because an incorrect support bearing will be produced. The utmost care should be taken to avoid this.

NFR15 — The program must run on an average Windows XP machine as is used by Koese Engineering.

Chapter 6

User interface design

In this chapter, the graphical user interface (UI) of the program is described in some detail, and the rationale behind this design is presented.

In general, the user (an employee of Koese Engineering) will have a fixed hole pattern that needs to be drilled, as specified by the customer of Koese Engineering. This pattern will in general be the first input of the program, and will not often be modified later. The other inputs may follow in any order.

It may be sensible to construct the interface in such a way that the hole pattern is input first, and is not easy to modify later on. This approach would result in a less cluttered interface, because the controls to change the hole pattern will be hidden, but this also means that parts of the user interface are hidden at any point. This makes the UI less discoverable (*NFR12*), and therefore the decision was made to put all controls into a single window. The user can then immediately see what's available.

The most intuitive way to specify a hole pattern (*FR2*), top bearings (*FR3*), shaft assignments (*FR4*) and support bearings (*FR6*) is visually. The first three can best be done in a top view of the drill head; support bearings need a side view. In addition, we need a way to specify the main dimensions (*FR1*). Although this could be done visually in a side view, this would result in a very nonstandard and possibly counter-intuitive control, so simple numerical edit boxes are preferred.

The result of this approach can be seen in figure 6.1. The main areas in this design will be explained in detail in the following sections.

6.1 Menu bar

This is a standard menu bar as found in many (Windows and non-Windows) programs.

The File menu contains the items to create a new drill head, to load, save (*FR10*) and print (*FR8*) drill head designs, to export the design to a DXF file (*FR9*), and to exit the program.

The Edit menu only contains a Settings item, which calls up a dialog in which the shaft shape error tolerance can be configured (*FR11*). This is considered an 'advanced' feature and the default setting should be fine in most cases.

The Help menu contains an item to toggle tool tips (*NFR12*) on or off (in case the user knows the program well enough and they become annoying) and an About item displaying

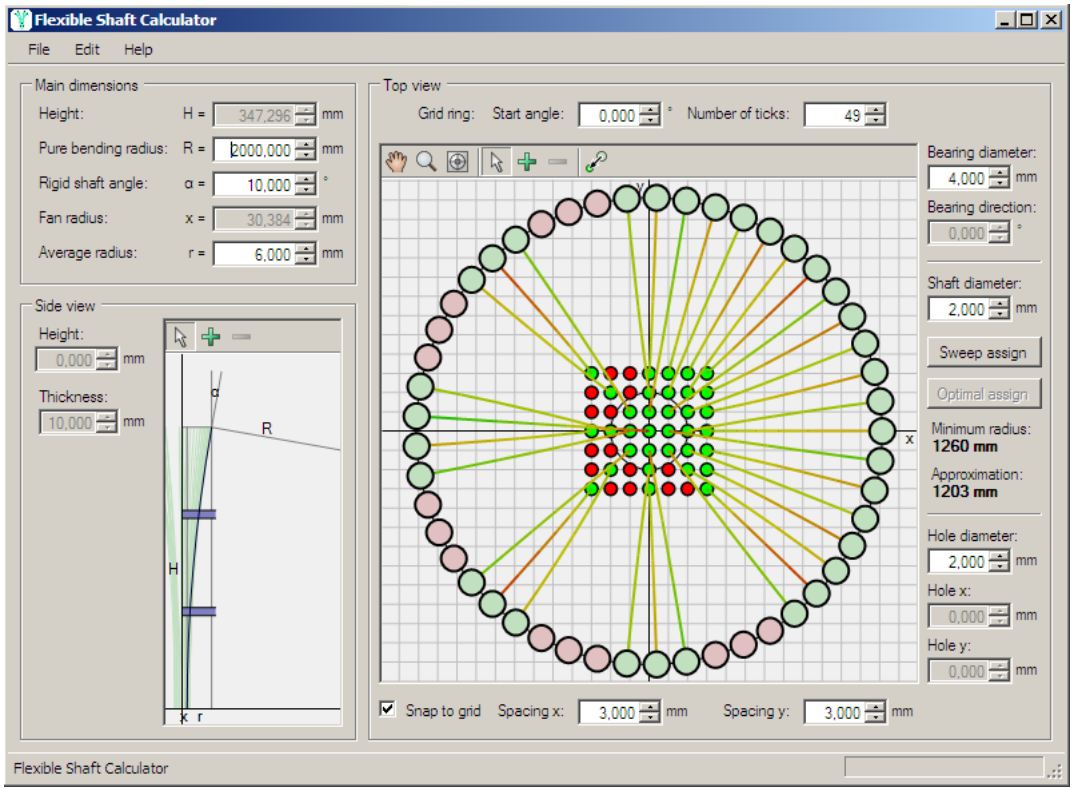


Figure 6.1: A screen shot of the final program, showing a drill head design in which some shafts have not yet been assigned.

information about the program. No on-line help is provided, because a printed manual will be produced.

6.2 Main dimensions

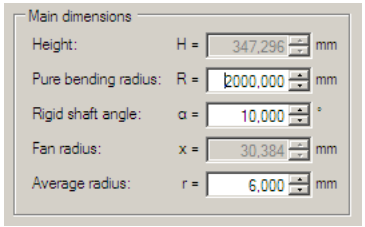


Figure 6.2: The edit boxes used to enter the main dimensions of the drill head.

The main dimensions area (figure 6.2) allows the user to configure some of the main dimensions (*FR1*). Because the first four of these dimensions are interdependent, the two that are grayed out are computed automatically. It proved to be very difficult to come up with a design that allows the user to enter any combination of dimensions that is sufficient to

compute the others, without sacrificing usability.

The labels H, R etcetera are used to refer to the side view.

6.3 Side view

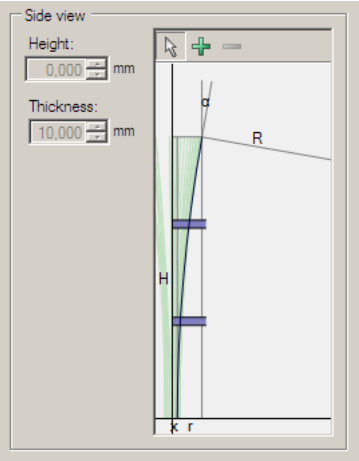


Figure 6.3: The side view of the drill head, showing two support bearings.

The side view (figure 6.3) shows the meanings of the symbols as used in the main dimensions area. The only manipulations that can be done in this area relate to the support bearings.

The look and behaviour is similar to that found in many graphics and CAD programs. The side view contains a tool bar with the tools needed to manipulate the items in the view. In this case, the 'arrow tool' allows support bearings to be selected and dragged up and down, the 'add tool' allows creating new support bearings and the 'delete tool' removes the currently selected bearing (*FR6*).

When a bearing is selected, the boxes at the left become available, and the height and thickness of the selected bearing can be specified numerically. Moreover, selecting a bearing will show in the top view (see section 6.4) the location of the holes in that bearing. Because the position of the holes is shown at the top as well as the bottom of the bearing, the support bearings are drawn with a black line at the top and bottom, indicating those planes of intersection.

6.4 Top view

As indicated before, the top view (figure 6.4) is where most of the editing takes place. Three types of parts of the drill head are manipulated here: top bearings, shafts and guiding block holes. These will be discussed in the following subsections. The controls related to the top bearings are placed at the top, those related to holes are placed at the bottom, and those related to shafts are placed in the middle. This provides a subtle clue as to the purpose of controls, as well as making them easier to find.

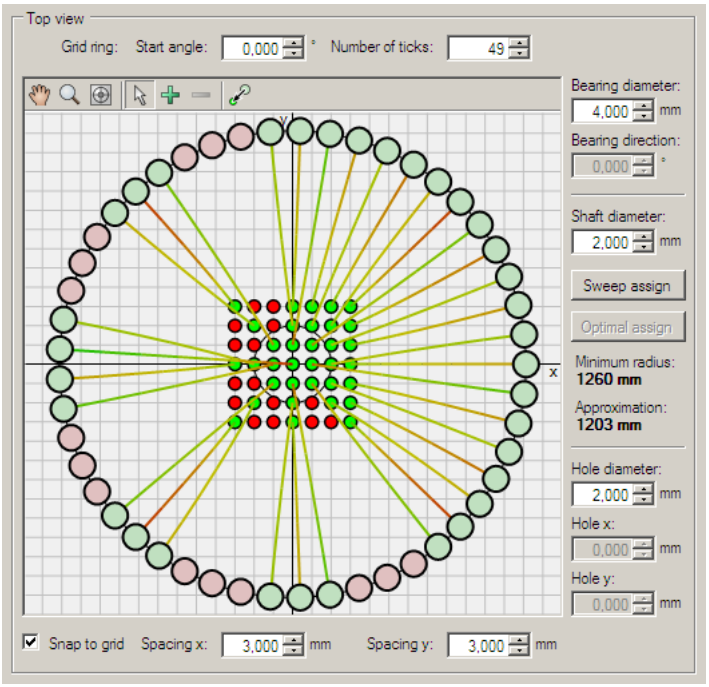


Figure 6.4: The top view of the drill head, showing 49 holes and top bearings, of which 34 are connected to each other.

Because the top view can become cluttered at times, it can be zoomed in using the 'zoom tool' for a better view. It can then be scrolled using the 'pan tool', resembling the hand tool which is used in programs like Adobe Reader. The third tool in the row, the 'reset view tool', resets the view to show the entire drill head.

6.4.1 Top bearings

The top bearings in a drill head are often spaced equally. For this purpose, a grid is provided on the ring of top bearings. The number of ticks and the rotation of this grid can be configured using the boxes at the top.

Bearings that are connected to a shaft are shown in light green; unconnected bearings have a light red colour. This allows the user to easily spot the bearings that are still unused.

Creating top bearings (*FR3*) is done using the 'add tool' available from the tool bar. The user can click on the top ring to create a bearing at the closest grid point. He can also drag to create a range of bearings in one action, something that will often be useful.

Bearings can be moved around using the 'arrow tool'. Multiple bearings can be selected using the Shift or Control keys; they will retain their distance to each other while they are moving.

The diameter of all bearings can be set in the box to the right. The position of the currently selected bearing can be entered below that, in case the grid does not provide the required accuracy.

Removing bearings is done by selecting them and pressing the Delete key or clicking the 'remove' button on the tool bar, similar to many drawing programs.

6.4.2 Shafts

A shaft cannot exist on its own: it is always a connection between a bearing and a hole. The 'add shaft' tool (last on the tool bar) is used to create such a connection by dragging between the bearing and the hole (*FR4*). An existing connection can be severed by dragging into an empty area.

Holes and bearings that are connected with a shaft are shown in green; unconnected ones are shown in red. This allows for easy checking whether the shaft assignment is complete.

Shafts themselves are coloured according to their curvature. The largest curvature in the drill head (corresponding to the most likely point of failure) is shown in red; the smallest in green and intermediate values in shades ranging, via yellow, in between red and green. In this way, the most likely point of failure can be identified and improved if possible.

The diameter of all shafts can be configured in the box to the right (*FR5*). The shafts are not drawn using this diameter, because that would make the top view extremely cluttered. When a support bearing is selected in the left view, however, the support bearing holes shown in the top view *are* drawn using this diameter (section 6.4.4).

Below this, two buttons appear that can be used to assign shafts automatically. The 'sweep assign' is a heuristic currently used by Koese, which assigns top bearings to holes in the order they are encountered by a clock-like sweep line. The 'optimal assign' should compute the optimal shaft assignment (resulting in the smallest maximum curvature), but this function is not (yet?) implemented and therefore the button is grayed out.

The precise value of the smallest bending radius (corresponding to the largest curvature) is shown to the right of the top view, along with a value computed using an approximation formula that has been used by Koese for years. If these numbers differ too much, this may indicate a potential problem (*NFR14*).

6.4.3 Guiding block holes

The holes to be drilled are shown as circles. These are added, modified and removed just like top bearings (*FR2*). They can be snapped to a grid consisting of horizontal and vertical lines, with a distance configurable below the top view.

Holes that have a shaft running to them are shown in bright green; those that are not yet connected are bright red. In this way the user can easily see which holes still need attention.

The diameter of all holes can be set to the right of the view. Holes are drawn using this diameter.

When a hole is selected, the boxes indicating its position become available. For multiple selected holes, the box for the x-coordinate is available only if the x-coordinate is the same for all selected holes; the same holds for the y-coordinate.

6.4.4 Support bearing holes

To avoid clutter in the top view, we do not draw all holes in support bearings in the top view. Instead, we draw only the holes in the currently selected support bearing (*FR7*). In contrast to the internal computations, which compute the position of the hole at the centre of the bearing, the top view shows the position of the hole at the top and bottom of the bearing, with a line connecting them (figure 6.5). This gives the user a better 'feel' for the

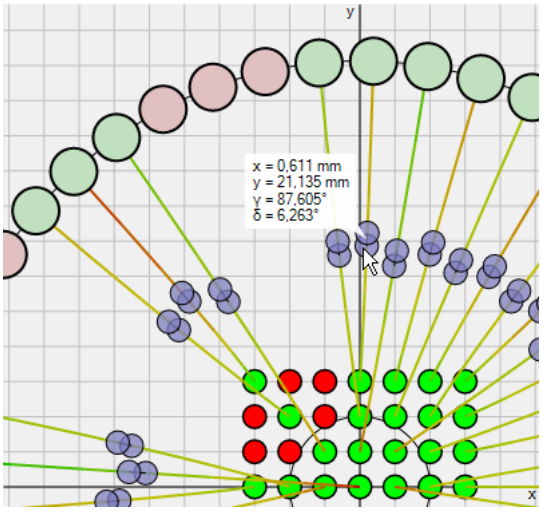


Figure 6.5: The top view, zoomed in, showing around a dozen holes in the currently selected support bearing, as well as a pop-up balloon showing the exact coordinates of one of these holes.

angle (azimuth and elevation) of the hole, and allows for better visual checking whether the holes overlap.

When hovering the mouse over such a hole, a 'balloon' appears with the precise coordinates and angle of that hole. The point of the balloon points exactly to the hole position at the centre of the bearing, indicating the location that is referred to by the coordinates inside the balloon.

Chapter 7

Technical design

This chapter will discuss the overall structure of the program and elaborate on some of the design choices made.

7.1 Language and tools

Numerical algorithms can be implemented in nearly any programming language, and for many languages (most notably C) pre-built libraries exist. However, it was not clear from the start what the precise status of the program would be, which ruled out any library under GPL or a similar license. Implementing the algorithms by hand will give more control and should not be too much work, given the pseudocode from the book [3]. The choice of language, therefore, need not depend on library availability.

The most important properties of a language are the speed with which code can be developed, and the maintainability of the resulting code. Both considerations rule out archaic low-level languages like C and C++, which require manual memory management and do not, for example, provide array boundary checking. This makes programming in those languages tricky, dangerous, error-prone and time-consuming. Although execution speed is not essential, because we expect the algorithms to be fast, a scripting language like Python or Perl might be too slow for the task. Higher level (partially) compiled languages, like Java or C#, do not suffer from these problems (*NFR13*). Java, however, has many annoyances, and C# is in general a much more pleasant language in the author's humble opinion.

Another important consideration is the graphical user interface that is to be built. Implementing a GUI without a visual development environment will take a disproportionate amount of time and effort. A visual tool needs to be available for the language and toolkit used.

The program only needs to run on modern Windows systems (*NFR15*), and therefore a portable language is not needed. Indeed, portability may be bad for usability, because portable GUI toolkits usually do not employ the native Windows controls, instead re-inventing the wheel in order to roll on all platforms.

Taking into consideration the previous discussions, C# [6] was chosen as the language of implementation, using .NET's [7] Windows Forms[1] [9] in Microsoft Visual Studio 2005 [10] to produce the graphical user interface. For the unit tests, the logical choice is NUnit [11].

[1] Despite the name, a port to UNIX platforms is well under way, in the form of the Mono project [8]. At the time of writing, Mono is nearly complete enough to run the constructed application.

7.2 Overview of software structuring

It is nearly always a good idea to separate the frontend (user interface) from the backend (actual implementation) of the program. In this case it is essential. Without proper separation of responsibilities, it is impossible to implement automated unit tests to guarantee even a little bit of correctness (*NFR14*) of the backend code. The backend can again be split up into two: the actual model, and the numerical algorithms. The program itself is therefore split up into three namespaces: Maths (section 7.2.1), Model (section 7.2.2) and Gui (section 7.2.3). A fourth package, Tests (section 7.2.4), provides the unit tests and code to produce the tables from section 8.

7.2.1 Maths

The Maths namespace contains all general-purpose classes for matrix algebra and numerical algorithms.

The classes Matrix and Vector implement real-valued matrices and vectors of arbitrary dimensions. They also define overloaded operators and some useful methods like Inverse and Norm.

The class Function represents an abstract function in the mathematical sense, with real-valued vectors as input and output. It has an abstract method to evaluate the function value, and implements a method to compute an approximation to the Jacobian matrix in a given point. The abstract class DifferentialEquation represents a first-order ordinary differential equation. This is a special kind of Function where the dimension of the domain is one higher (for the time variable t) than the dimension of the codomain.

Taking a DifferentialEquation as one of its arguments, the static method Solve in the class RK4 performs the numerical integration discussed in section 4.3.

The static method Solve in the class BroydenSolver uses the algorithm from section 4.4 to solve $F(x) = 0$ for a certain Function F.

The class PiecewiseCubicSpline3D is initialised with a list of points and will interpolate these points using the algorithm from section 4.5. It then allows readout of the spline's x and y position at a given z-coordinate.

This namespace further contains some utility classes, like Set (representing an unordered collection of elements, without duplicates, which is strangely not implemented in the .NET library) and Point2D and Point3D, which are specialized Vectors representing points in the plane and in space.

7.2.2 Model

The actual model classes can best be described using an UML diagram as in figure 7.1.

The main container class is DrillHead. It contains sets of objects for TopBearing, Shaft, GuidingBlockHole and SupportBearing, which all directly correspond to the physical objects discussed in section 2. All these classes inherit from DrillHeadPart, whose main purpose is to maintain a pointer to the parent DrillHead. (It also assigns each part a unique identifier which is used during loading and saving.)

A shaft can be associated with a top bearing and a guiding block hole, but it does not have to be: while the drill head is being constructed in the GUI we want to be able to use

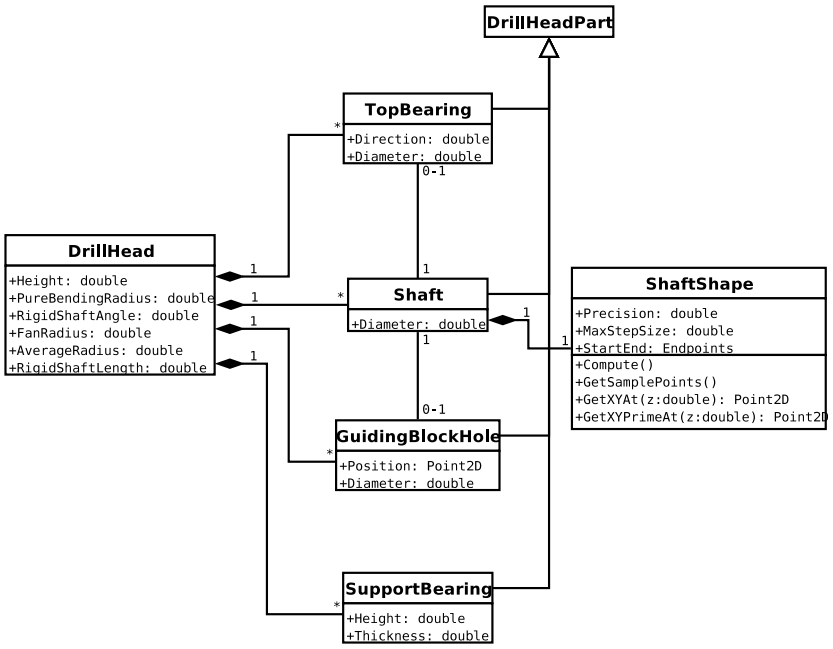


Figure 7.1: A UML diagram of the model classes.

'dangling' shafts which are not yet connected to anything. However, we allow only dangling shafts at the bottom end: there is a one-to-one mapping between shafts and top bearings. This simplifies implementation.

Shaft shape

ShaftShape is the class that actually drives the numerical algorithms to produce the estimated shape of the shaft. This is separate from Shaft to facilitate easier testing: in this way, we can test ShaftShape without the need to construct a DrillHead with its components.

A ShaftShape computes its shape by the procedure described in section 4.1.

First, it lets the BroydenSolver solve using a Function called ShaftErrorFunc. This function takes a four-dimensional vector containing F_x, F_y, M_{x0} and M_{y0} and returns a four-dimensional vector containing the difference between the desired and actual endpoint, which we want to be zero. This is exactly what the BroydenSolver does when given this function.

The ShaftErrorFunc performs the numerical integration using the RK4 class on the differential equation (3.9), which is encapsulated in the class ShaftDiffEq. It takes the last point returned by the RK4 algorithm and computes the error for each of the four dimensions.

Finally, the parameters returned by the BroydenSolver are fed into the RK4 solver once more to produce the final list of sample points, which are then interpolated using PiecewiseCubicSpline3D to produce the final interpolation.

7.2.3 GUI

The Gui namespace contains all classes for the graphical user interface. It contains mostly code generated by the IDE, and the parts that are written manually are straightforward and

uninteresting for the most part (for example, "if the value in the text field for the drill head height is modified, set the height of the drill head in the model to the specified value").

The nontrivial parts are in the abstract Manipulator class and its descendants. This class is a graphical control consisting of a canvas and a toolbar. Tools are represented by ManipulatorTool objects. These objects can add event listeners to the canvas, thereby handling mouse events. Concrete implementations of tools can be added to the toolbar in classes derived from Manipulator. Because the tools themselves will only work on a specific subclass of Manipulator, generics are used to define an abstract class GenericManipulatorTool<M>, where M must be a subclass of Manipulator. All tools derive from this class, filling in a specific Manipulator type for M.

Two concrete Manipulators are used in the program. The first is the SideManipulator, which displays the side view of a drill head, and contains tools to add, move and remove support bearings.

The second and most important Manipulator is the TopManipulator, which shows the top view of the drill head. It contains tools to add, move and remove guiding block holes and top bearings, and to connect them with shafts. When a support bearing is selected in the SideManipulator, the TopManipulator will show the positions of the holes in this particular support bearing for every shaft.

7.2.4 Tests

Unit tests are written using the NUnit framework [11]. This framework works similarly to other xUnit frameworks, allowing a test suite to first set up some objects, then to run some code using these objects, and finally (or in between) check whether the actual result matches the expected result.

Current tests include standalone tests for most of the numeric algorithms and many shaft shape tests. The matrix algebra classes are not tested separately, but are heavily used in all other algorithms and are thereby tested implicitly.

To test the numerical algorithms, examples from [3] are used. To test the shaft shape classes, several different approaches are used; please see chapter 8 for a description. The unit tests are a superset of the tests and results presented therein.

Chapter 8

Tests and Results

The program has been subjected to several automated tests. The results are compared to a circle for certain cases where this is known to be correct. For other cases, the results are compared to a computation from the same program, but done with very high accuracy.

Moreover, the results have been compared to real-world measurements done by Koese.

8.1 Comparison to the circle

If the bottom of the shaft is exactly on the 'average radius' circle, we know that it will take on the shape of a circular arc. In table 8.1, four of these cases are tested. The first set of dimensions are from a drill head Koese was designing at the time of writing; the other three are also realistic sets of dimensions and were taken from [2].

The errors ϵ and ϵ' were found by measuring the deviation of the computed spline from the real circle at 100 points, equally spaced on the z-axis. For ϵ, the Euclidean distance in the xy-plane was computed. For ϵ' the Euclidean norm of the difference between the derivatives was used.

It is immediately clear from the τ/ϵ column, which should be greater than 1 everywhere, that the errors ϵ in the result are within the specified error bound τ. Looking at τ/ϵ', we see that the error ϵ' in the derivative is even orders of magnitude smaller than τ.

8.2 Comparison to accurate computations

For shafts that do not assume a circular shape, we need a precise computation to compare the results to. Table 8.2 gives some of these cases. The three examples from [2] were reused here; for brevity, the first example from section 8.1 is omitted. The shift with respect to the ideal position is (b_x, b_y).

The 'exact' result was acquired by setting $\tau = 1 \cdot 10^{-6}$ (one nanometre). This resulted in a shaft shape consisting of 2000 to 3000 points in all cases. The errors ϵ and ϵ' were found by measuring the deviation in the xy-plane at 100 points, similar to the circular cases.

We observe that these more difficult cases do not pose a problem for the algorithm either. The error ϵ stays within its bound τ, and again ϵ' is orders of magnitude smaller than τ.

Precision τ	Points	Error in position ϵ	τ/ϵ	Error in derivative ϵ'	τ/ϵ'
$H = 258.8$, $R = 1000.0$, $\alpha = 15°$					
$1 \cdot 10^{+1}$	2	$6.1 \cdot 10^{-1}$	$1.6 \cdot 10^{+1}$	$1.0 \cdot 10^{-2}$	$9.8 \cdot 10^{+2}$
$1 \cdot 10^{+0}$	5	$6.5 \cdot 10^{-2}$	$1.5 \cdot 10^{+1}$	$1.4 \cdot 10^{-3}$	$7.1 \cdot 10^{+2}$
$1 \cdot 10^{-1}$	11	$4.2 \cdot 10^{-3}$	$2.4 \cdot 10^{+1}$	$8.6 \cdot 10^{-5}$	$1.2 \cdot 10^{+3}$
$1 \cdot 10^{-2}$	32	$3.6 \cdot 10^{-5}$	$2.8 \cdot 10^{+2}$	$7.4 \cdot 10^{-7}$	$1.3 \cdot 10^{+4}$
$1 \cdot 10^{-3}$	98	$3.6 \cdot 10^{-5}$	$2.8 \cdot 10^{+1}$	$7.3 \cdot 10^{-7}$	$1.4 \cdot 10^{+3}$
$1 \cdot 10^{-4}$	306	$3.8 \cdot 10^{-8}$	$2.7 \cdot 10^{+3}$	$7.7 \cdot 10^{-10}$	$1.3 \cdot 10^{+5}$
$1 \cdot 10^{-5}$	965	$3.8 \cdot 10^{-8}$	$2.7 \cdot 10^{+2}$	$7.6 \cdot 10^{-10}$	$1.3 \cdot 10^{+4}$
$H = 180.0$, $R = 800.2$, $\alpha = 13°$					
$1 \cdot 10^{+1}$	2	$2.7 \cdot 10^{-1}$	$3.7 \cdot 10^{+1}$	$6.4 \cdot 10^{-3}$	$1.6 \cdot 10^{+3}$
$1 \cdot 10^{+0}$	4	$2.1 \cdot 10^{-2}$	$4.8 \cdot 10^{+1}$	$6.5 \cdot 10^{-4}$	$1.5 \cdot 10^{+3}$
$1 \cdot 10^{-1}$	9	$1.1 \cdot 10^{-3}$	$9.2 \cdot 10^{+1}$	$3.1 \cdot 10^{-5}$	$3.2 \cdot 10^{+3}$
$1 \cdot 10^{-2}$	25	$7.4 \cdot 10^{-6}$	$1.4 \cdot 10^{+3}$	$2.3 \cdot 10^{-7}$	$4.3 \cdot 10^{+4}$
$1 \cdot 10^{-3}$	75	$7.4 \cdot 10^{-6}$	$1.4 \cdot 10^{+2}$	$2.1 \cdot 10^{-7}$	$4.7 \cdot 10^{+3}$
$1 \cdot 10^{-4}$	235	$7.4 \cdot 10^{-6}$	$1.4 \cdot 10^{+1}$	$2.1 \cdot 10^{-7}$	$4.7 \cdot 10^{+2}$
$1 \cdot 10^{-5}$	741	$8.3 \cdot 10^{-9}$	$1.2 \cdot 10^{+3}$	$2.4 \cdot 10^{-10}$	$4.2 \cdot 10^{+4}$
$H = 347.0$, $R = 1998.3$, $\alpha = 10°$					
$1 \cdot 10^{+1}$	2	$2.4 \cdot 10^{-1}$	$4.2 \cdot 10^{+1}$	$2.8 \cdot 10^{-3}$	$3.6 \cdot 10^{+3}$
$1 \cdot 10^{+0}$	4	$1.0 \cdot 10^{-2}$	$9.9 \cdot 10^{+1}$	$1.6 \cdot 10^{-4}$	$6.3 \cdot 10^{+3}$
$1 \cdot 10^{-1}$	10	$3.4 \cdot 10^{-4}$	$3.0 \cdot 10^{+2}$	$5.0 \cdot 10^{-6}$	$2.0 \cdot 10^{+4}$
$1 \cdot 10^{-2}$	30	$3.4 \cdot 10^{-4}$	$3.0 \cdot 10^{+1}$	$5.0 \cdot 10^{-6}$	$2.0 \cdot 10^{+3}$
$1 \cdot 10^{-3}$	90	$1.4 \cdot 10^{-6}$	$7.0 \cdot 10^{+2}$	$2.1 \cdot 10^{-8}$	$4.8 \cdot 10^{+4}$
$1 \cdot 10^{-4}$	282	$1.4 \cdot 10^{-6}$	$7.0 \cdot 10^{+1}$	$2.1 \cdot 10^{-8}$	$4.8 \cdot 10^{+3}$
$1 \cdot 10^{-5}$	890	$8.1 \cdot 10^{-10}$	$1.2 \cdot 10^{+4}$	$1.2 \cdot 10^{-11}$	$8.3 \cdot 10^{+5}$
$H = 750.0$, $R = 3930.6$, $\alpha = 11°$					
$1 \cdot 10^{+1}$	3	$6.9 \cdot 10^{-1}$	$1.5 \cdot 10^{+1}$	$3.8 \cdot 10^{-3}$	$2.6 \cdot 10^{+3}$
$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.6 \cdot 10^{-4}$	$3.8 \cdot 10^{+3}$
$1 \cdot 10^{-1}$	15	$1.4 \cdot 10^{-3}$	$7.1 \cdot 10^{+1}$	$9.7 \cdot 10^{-6}$	$1.0 \cdot 10^{+4}$
$1 \cdot 10^{-2}$	45	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+3}$	$5.1 \cdot 10^{-8}$	$2.0 \cdot 10^{+5}$
$1 \cdot 10^{-3}$	139	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+2}$	$4.9 \cdot 10^{-8}$	$2.1 \cdot 10^{+4}$
$1 \cdot 10^{-4}$	436	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+1}$	$4.9 \cdot 10^{-8}$	$2.1 \cdot 10^{+3}$
$1 \cdot 10^{-5}$	1377	$5.5 \cdot 10^{-9}$	$1.8 \cdot 10^{+3}$	$3.7 \cdot 10^{-11}$	$2.7 \cdot 10^{+5}$

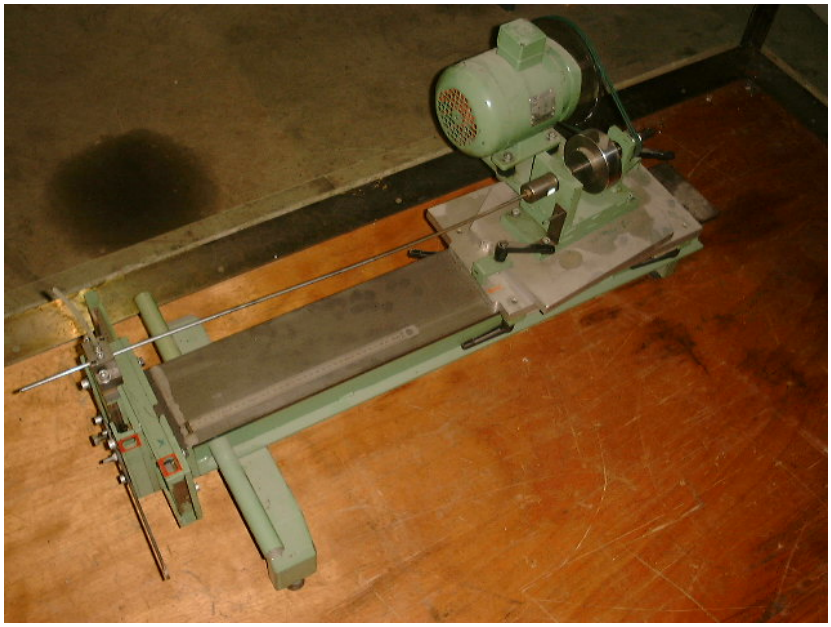
Table 8.1: Circular test cases, tested against the shape of a real circle.

b_x, b_y	Precision τ	Points	Posèrror ϵ	τ/ϵ	Dir. error ϵ'	τ/ϵ'
$H = 180.0$, $R = 800.2$, $\alpha = 13°$						
$b_x = 1.5$, $b_y = 1.5$	$1 \cdot 10^{+0}$	4	$2.1 \cdot 10^{-2}$	$4.8 \cdot 10^{+1}$	$6.5 \cdot 10^{-4}$	$1.5 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$7.4 \cdot 10^{-6}$	$1.4 \cdot 10^{+3}$	$2.3 \cdot 10^{-7}$	$4.3 \cdot 10^{+4}$
	$1 \cdot 10^{-4}$	235	$7.4 \cdot 10^{-6}$	$1.4 \cdot 10^{+1}$	$2.1 \cdot 10^{-7}$	$4.7 \cdot 10^{+2}$
$b_x = -1.1$, $b_y = -1.1$	$1 \cdot 10^{+0}$	4	$2.2 \cdot 10^{-2}$	$4.6 \cdot 10^{+1}$	$5.6 \cdot 10^{-4}$	$1.8 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$2.2 \cdot 10^{-5}$	$4.6 \cdot 10^{+2}$	$5.4 \cdot 10^{-7}$	$1.8 \cdot 10^{+4}$
	$1 \cdot 10^{-4}$	235	$1.4 \cdot 10^{-11}$	$7.3 \cdot 10^{+6}$	$5.5 \cdot 10^{-11}$	$1.8 \cdot 10^{+6}$
$b_x = 4.1$, $b_y = 4.1$	$1 \cdot 10^{+0}$	4	$2.3 \cdot 10^{-2}$	$4.3 \cdot 10^{+1}$	$8.2 \cdot 10^{-4}$	$1.2 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$4.9 \cdot 10^{-4}$	$2.0 \cdot 10^{+1}$	$1.7 \cdot 10^{-5}$	$6.1 \cdot 10^{+2}$
	$1 \cdot 10^{-4}$	235	$1.9 \cdot 10^{-6}$	$5.3 \cdot 10^{+1}$	$6.2 \cdot 10^{-8}$	$1.6 \cdot 10^{+3}$
$b_x = 1.5$, $b_y = 1.5$	$1 \cdot 10^{+0}$	4	$2.1 \cdot 10^{-2}$	$4.9 \cdot 10^{+1}$	$6.4 \cdot 10^{-4}$	$1.6 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$9.9 \cdot 10^{-4}$	$1.0 \cdot 10^{+1}$	$2.8 \cdot 10^{-5}$	$3.5 \cdot 10^{+2}$
	$1 \cdot 10^{-4}$	235	$7.7 \cdot 10^{-6}$	$1.3 \cdot 10^{+1}$	$2.2 \cdot 10^{-7}$	$4.6 \cdot 10^{+2}$
$b_x = -1.1$, $b_y = -1.1$	$1 \cdot 10^{+0}$	4	$2.2 \cdot 10^{-2}$	$4.6 \cdot 10^{+1}$	$5.7 \cdot 10^{-4}$	$1.8 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$1.4 \cdot 10^{-5}$	$7.2 \cdot 10^{+2}$	$3.4 \cdot 10^{-7}$	$2.9 \cdot 10^{+4}$
	$1 \cdot 10^{-4}$	235	$1.6 \cdot 10^{-7}$	$6.3 \cdot 10^{+2}$	$4.2 \cdot 10^{-9}$	$2.4 \cdot 10^{+4}$
$b_x = 4.1$, $b_y = 4.1$	$1 \cdot 10^{+0}$	4	$2.1 \cdot 10^{-2}$	$4.8 \cdot 10^{+1}$	$7.5 \cdot 10^{-4}$	$1.3 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	25	$4.7 \cdot 10^{-4}$	$2.1 \cdot 10^{+1}$	$1.6 \cdot 10^{-5}$	$6.3 \cdot 10^{+2}$
	$1 \cdot 10^{-4}$	235	$2.6 \cdot 10^{-6}$	$3.8 \cdot 10^{+1}$	$8.5 \cdot 10^{-8}$	$1.2 \cdot 10^{+3}$
$H = 347.0$, $R = 1998.3$, $\alpha = 10°$						
$b_x = 20.0$, $b_y = 20.0$	$1 \cdot 10^{+0}$	4	$1.0 \cdot 10^{-2}$	$9.9 \cdot 10^{+1}$	$1.6 \cdot 10^{-4}$	$6.3 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	30	$3.4 \cdot 10^{-4}$	$3.0 \cdot 10^{+1}$	$5.0 \cdot 10^{-6}$	$2.0 \cdot 10^{+3}$
	$1 \cdot 10^{-4}$	282	$1.4 \cdot 10^{-6}$	$7.0 \cdot 10^{+1}$	$2.1 \cdot 10^{-8}$	$4.8 \cdot 10^{+3}$
$b_x = 10.0$, $b_y = 10.0$	$1 \cdot 10^{+0}$	4	$1.3 \cdot 10^{-2}$	$7.7 \cdot 10^{+1}$	$1.2 \cdot 10^{-4}$	$8.4 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	30	$6.1 \cdot 10^{-4}$	$1.6 \cdot 10^{+1}$	$6.4 \cdot 10^{-6}$	$1.6 \cdot 10^{+3}$
	$1 \cdot 10^{-4}$	282	$1.7 \cdot 10^{-11}$	$5.9 \cdot 10^{+6}$	$4.2 \cdot 10^{-11}$	$2.4 \cdot 10^{+6}$
$b_x = 30.0$, $b_y = 30.0$	$1 \cdot 10^{+0}$	4	$7.9 \cdot 10^{-2}$	$1.3 \cdot 10^{+1}$	$1.6 \cdot 10^{-3}$	$6.2 \cdot 10^{+2}$
	$1 \cdot 10^{-2}$	30	$2.0 \cdot 10^{-5}$	$5.1 \cdot 10^{+2}$	$5.9 \cdot 10^{-7}$	$1.7 \cdot 10^{+4}$
	$1 \cdot 10^{-4}$	282	$2.0 \cdot 10^{-5}$	$5.1 \cdot 10^{+0}$	$4.9 \cdot 10^{-7}$	$2.0 \cdot 10^{+2}$
$b_x = 20.0$, $b_y = 20.0$	$1 \cdot 10^{+0}$	4	$1.0 \cdot 10^{-2}$	$9.7 \cdot 10^{+1}$	$1.7 \cdot 10^{-4}$	$5.9 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	30	$1.6 \cdot 10^{-4}$	$6.1 \cdot 10^{+1}$	$2.0 \cdot 10^{-6}$	$5.0 \cdot 10^{+3}$
	$1 \cdot 10^{-4}$	282	$1.7 \cdot 10^{-6}$	$5.7 \cdot 10^{+1}$	$2.3 \cdot 10^{-8}$	$4.4 \cdot 10^{+3}$
$b_x = 10.0$, $b_y = 10.0$	$1 \cdot 10^{+0}$	4	$1.6 \cdot 10^{-2}$	$6.1 \cdot 10^{+1}$	$1.8 \cdot 10^{-4}$	$5.6 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	30	$3.1 \cdot 10^{-4}$	$3.2 \cdot 10^{+1}$	$4.4 \cdot 10^{-6}$	$2.3 \cdot 10^{+3}$
	$1 \cdot 10^{-4}$	282	$5.1 \cdot 10^{-6}$	$2.0 \cdot 10^{+1}$	$7.1 \cdot 10^{-8}$	$1.4 \cdot 10^{+3}$
$b_x = 30.0$, $b_y = 30.0$	$1 \cdot 10^{+0}$	4	$5.2 \cdot 10^{-3}$	$1.9 \cdot 10^{+2}$	$1.3 \cdot 10^{-4}$	$7.6 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	30	$6.8 \cdot 10^{-5}$	$1.5 \cdot 10^{+2}$	$1.3 \cdot 10^{-6}$	$7.5 \cdot 10^{+3}$
	$1 \cdot 10^{-4}$	282	$5.4 \cdot 10^{-7}$	$1.9 \cdot 10^{+2}$	$1.0 \cdot 10^{-8}$	$9.6 \cdot 10^{+3}$
$H = 750.0$, $R = 3930.6$, $\alpha = 11°$						
$b_x = 20.0$, $b_y = 20.0$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.6 \cdot 10^{-4}$	$3.8 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+3}$	$5.1 \cdot 10^{-8}$	$2.0 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+1}$	$4.9 \cdot 10^{-8}$	$2.1 \cdot 10^{+3}$
$b_x = 17.5$, $b_y = 17.5$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.5 \cdot 10^{-4}$	$4.0 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$9.6 \cdot 10^{-6}$	$1.0 \cdot 10^{+3}$	$6.3 \cdot 10^{-8}$	$1.6 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$9.6 \cdot 10^{-6}$	$1.0 \cdot 10^{+1}$	$6.3 \cdot 10^{-8}$	$1.6 \cdot 10^{+3}$
$b_x = 22.5$, $b_y = 22.5$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.8 \cdot 10^{-4}$	$3.6 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$5.3 \cdot 10^{-6}$	$1.9 \cdot 10^{+3}$	$4.1 \cdot 10^{-8}$	$2.4 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$5.3 \cdot 10^{-6}$	$1.9 \cdot 10^{+1}$	$3.7 \cdot 10^{-8}$	$2.7 \cdot 10^{+3}$
$b_x = 20.0$, $b_y = 20.0$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.6 \cdot 10^{-4}$	$3.8 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+3}$	$5.1 \cdot 10^{-8}$	$2.0 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$7.2 \cdot 10^{-6}$	$1.4 \cdot 10^{+1}$	$4.9 \cdot 10^{-8}$	$2.1 \cdot 10^{+3}$
$b_x = 17.5$, $b_y = 17.5$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.5 \cdot 10^{-4}$	$4.0 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$9.5 \cdot 10^{-6}$	$1.0 \cdot 10^{+3}$	$6.3 \cdot 10^{-8}$	$1.6 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$9.5 \cdot 10^{-6}$	$1.0 \cdot 10^{+1}$	$6.2 \cdot 10^{-8}$	$1.6 \cdot 10^{+3}$
$b_x = 22.5$, $b_y = 22.5$	$1 \cdot 10^{+0}$	6	$3.6 \cdot 10^{-2}$	$2.8 \cdot 10^{+1}$	$2.8 \cdot 10^{-4}$	$3.6 \cdot 10^{+3}$
	$1 \cdot 10^{-2}$	45	$5.3 \cdot 10^{-6}$	$1.9 \cdot 10^{+3}$	$4.2 \cdot 10^{-8}$	$2.4 \cdot 10^{+5}$
	$1 \cdot 10^{-4}$	436	$5.3 \cdot 10^{-6}$	$1.9 \cdot 10^{+1}$	$3.8 \cdot 10^{-8}$	$2.7 \cdot 10^{+3}$

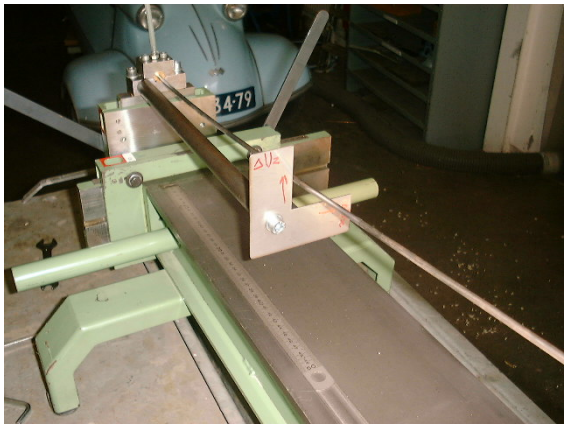
Table 8.2: Non-circular test cases, tested against very precise computation.

8.3 Real-world tests

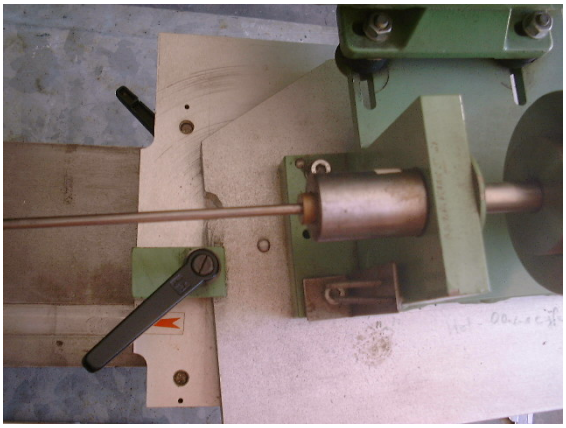
To verify that the computed results agree with observations in the real world, some measurements were performed by Koese. These were done in a special measuring set-up (figure 8.1), but the coordinates presented here are converted to the same system as used in the previous sections.



(a) The complete set-up.



(b) The 'bottom' of the shaft and the measuring bracket.



(c) The 'top' of the shaft where the angle can be set.

Figure 8.1: The measuring set-up used to acquire the real-world measuring results.

For a fixed height of 700, the angle of the shaft was varied on one side, as well as its endpoint on the other side. For each configuration, the x- and y-coordinates of the centreline of the shaft were measured halfway, at $h = 350$.

The measurement results can be seen side by side with the computations in table 8.3.

Note that, even though one decimal appears after the decimal point, the offset and measured position are only accurate up to 1 millimetre in each direction. The maximum acceptable error would therefore be $\sqrt{2} \approx 1.41$, and we see that it remains below this value in all cases.

Angle	Offset		Measured		Computed		Error
α (°)	b_x	b_y	v_x	v_y	w_x	w_y	ϵ
$H = 700$, $h = 350$							
5	15.6	0.0	15.6	0.0	15.4	0.0	0.16
5	0.6	0.0	8.1	0.0	7.9	0.0	0.15
12	23.6	0.0	31.1	0.0	30.1	0.0	0.92
12	38.1	0.0	38.1	0.0	37.5	0.0	0.62
12	35.6	30.0	36.6	15.0	36.2	15.1	0.40
12	23.6	40.0	30.6	21.0	30.1	20.2	0.95
12	3.6	40.0	20.6	20.0	20.0	20.2	0.58
5	-39.4	40.0	-12.4	21.0	-12.2	20.0	0.99
5	0.6	40.0	7.6	20.0	7.9	20.0	0.33

Table 8.3: Real-world test cases.

In fact, Koese has already used the program to produce two support bearings for a machine. Initially, one of them was accidentally mirrored in the production process, but after correcting this problem the holes turned out to be in the right place.

Chapter 9

Conclusion

The bending of shafts, rods or beams is an elementary problem in mathematical physics, but is often only treated in a linearised form in two dimensions. The addition of a third dimension and the removal of the linearisations complicate the problem.

Equations can be derived using a physical model involving the Bernoulli-Euler equation. Two parametrisations of the bending shaft have been presented, one more general and elegant, but more complex to use, the other more specific to this particular application and therefore posing a problem that is easier to solve.

The resulting differential equations are analytically unsolvable. A shooting method is able to produce a numerical solution. The method involves a fourth-order Runge-Kutta integration and a Broyden multidimensional equation solver, followed by a piecewise cubic polynomial interpolation.

It can be hard to estimate the the error in any numerical approximation, making it difficult to keep the error within a specified bound. Using assumptions that hold fairly well in practical situations, it has been shown that the results from the provided algorithm remain within the specified error bound in test cases derived from practical situations.

Based on a set of functional and nonfunctional requirements, a user-friendly user interface has been designed to access these underlying algorithms. A technical design was made to facilitate easy maintenance of the program. The program was implemented in C#.

Unit tests have shown, to a certain degree, the correctness of the program. Moreover, the program has already shown its use and usability in practice.

Bibliography

[1] *Koese Engineering: Multi Compact Drilling, Drilling Dotcodes, Micro-EDM and Prototyping*. URL `http://www.koese.nl/`.

[2] Hartono, Sri Subarinah, Sudi Prayitno, F.P.H. van Beckum (supervisor). *Curvature of Flexible Shafts in Multiple Compact Drilling*. 1996. Unpublished.

[3] Richard L. Burden, J. Douglas Faires. *Numerical Analysis* (Brooks/Cole, 2001), seventh ed.

[4] *DXF: Drawing Exchange Format*. URL `http://www.autodesk.com/techpubs/autocad/acadr14/dxf/index.htm`.

[5] *SolidWorks – 3D Mechanical Design and 3D CAD Software*. URL `http://www.solidworks.com/`.

[6] *The C# Language*. URL `http://msdn2.microsoft.com/en-us/vcsharp/aa336809.aspx`.

[7] *.NET Framework Developer Center*. URL `http://msdn2.microsoft.com/en-us/netframework/default.aspx`.

[8] *Mono*. URL `http://www.mono-project.com/`.

[9] *Windows Forms*. URL `http://msdn2.microsoft.com/en-us/netframework/aa497342.aspx`.

[10] *Visual Studio 2005 Developer Center*. URL `http://msdn2.microsoft.com/en-us/vstudio/default.aspx`.

[11] *NUnit*. URL `http://www.nunit.org/`.

Appendix A

Pseudocode algorithms

A.1 The Broyden algorithm

This is the pseudocode version of Broyden's method, taken from [3, pp. 623–624] and adapted to meet a precision requirement in terms of the absolute error instead of the step size. The addition consists of lines 13–15. These replace a check on the magnitude of s after line 22.

The algorithm attempts to find an approximate solution to $F(x) = 0$, where $F : \mathbb{R}^n \to \mathbb{R}^n$.

Input: function F, initial estimate x, tolerance ϵ, maximum number of iterations N
Output: approximate solution x

```
    A_0 := J(x) { J(x) denotes the Jacobian of F in x }
 5  v := F(x)
    A := A_0^{-1} { matrix inverse }
    s := -Av
    x := x + s
    k := 2
10  while k ≤ N do
        w := v
        v := F(x)
        if ‖v‖ ≤ ε then
            return x
15      end if
        y := v - w
        z := -Ay
        p := -s^⊤ z { ^⊤ denotes transposition }
        u^⊤ := s^⊤ A
20      A := a + (1/p)(s + z)u^⊤
        s := -Av
        x := x + s
        k := k + 1
    end while
25  fail { maximum number of iterations exceeded }
```