NTNU
Innovation and Creativity

Advanced Filtering in Intuitive Robot Programming

Tore Martin Madsø Hauan

Master of Science in Physics and Mathematics
Submission date: June 2006
Supervisor: Harald Hanche-Olsen, MATH
Co-supervisor: Trygve Thomessen, PPM AS

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

Intuitive Robot Programming is a process to program industrial robots by tracking a task performed manually by a skilled operator. The tracking results in a lot of data which needs to be reduced before it is sent to the robot. The task of this diploma is to describe the mathematics in the context of reducing such data without considerably altering the path.

Assignment given: 13. January 2006
Supervisor: Harald Hanche-Olsen, MATH

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology,
Mathematics and Electrical Engineering

MASTER'S THESIS

FOR

STUD.TECHN. Tore Martin Madsø Hauan

FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

NTNU

Date due: June 9, 2006

Discipline: Numerics

Title: "Advanced Filtering in Intuitive Robot Programming"

Purpose of the work: Describe the mathematics in the context of reducing the data in programming an industrial robot.

This diploma thesis is carried out as a collaboration beetween PPM AS and the Department of Mathematical Sciences under guidance of the Managing Director of PPM Ph.D. Trygve Thommesen, and førsteamanuensis Harald Hanche-Olsen.

Trondheim, January 13, 2006.

Trond Digernes
Instituttleder
Dept. of Mathematical Sciences

Harald Hanche-Olsen
Førsteamanuensis
Dept. of Mathematical Sciences

Preface

Five years of studies are almost finished. This report documents this last semester's work, and completes my master's degree. After all these years it is nice to finally see that mathematics has applications beyond the books and exams.

To get the opportunity to be a part of the development of solutions that might improve the industrial sector both economically and with respect to the working conditions for the employees, is satisfying. That said, I wish to thank the people at PPM for giving me this opportunity, especially Ph.D. Trygve Thommesen and M.Sc. John Einar Reitan, which have been the external contributors to this work.

I also want to thank Harald Hanche-Olsen for his guidance during this project, and for being the source of inspiration as lecturer in various courses throughout the study.

Trondheim, June 08, 2006

Tore Martin Madsø Hauan

Abstract

This text deals with the problem of reducing multi-dimensional data in the context of programming an industrial robot. Different ways to treat the positional and orientational data are discussed, and algorithms for each of these are developed and tested on various generated datasets. The outcome of the work was an algorithm expressing the position as three polynomials, one for each coordinate, and the orientation is then reduced with respect to given tolerances in Euler Angles. The resulting algorithm turned out to reduce a physical dataset with 97%. It was concluded that it is very satisfying to be able to reduce a set with this amount without loosing vital information.

Contents

1 Introduction

Todays society is paying more and more attention to health, environment and safety for the workforce. Especially in the industrial sector, employees are exposed to tasks that might cause immediate danger, or result in longterm injuries. To prevent this, employers may want to automate several tasks, but as with anything else, it is a matter of profitability and feasibility. The main problem for companies operating with smaller production runs, is that the programming of the robots is too timeconsuming to make it competitive to manual work.

PPM AS, founded December 31. 2000 by Ph.D. Trygve Thomessen and M.Sc. Per Kristian Sannæs, is developing a software solution for increasing the profitability for investments in industrial robots, even for small production runs. One of the properties of this software, is to considerably decrease the time it takes for reprogramming the robot. The software will be both user-friendly and efficient, because of the possibility to write and test the new programming in a Virtual Environment, while the robot still is in production [13]. The method is called "Intuitive Robot Programming" (IRP). The idea is when the software is ready to be launched, it is so intuitive that the operator which performs the task today, will be able to teach the robot that will be taking over his task(s). Some examples of the automated tasks are arc welding, illustrated in figure 1, assembling and sealing.

This text discusses the challenges in reducing the amount of data needed in training an industrial robot. It will focus on the mathematics and suggest one or several algorithms for solving each part of this process. Section 2 describes IRP in more detail, before section 3 look at the necessary theory from both mathematics and robotics needed in this context. The next sections contain the generated datasets, algorithms and the results of the algorithms run on the given data. Section 7 tests the final algorithm on a real dataset, before the text is summed up in discussion and conclusion.

Figure 1: Arc Welding

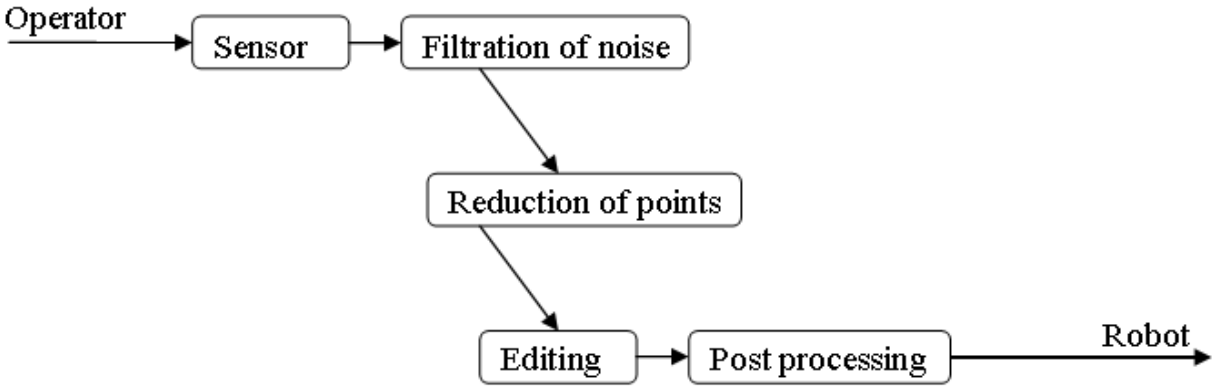


Figure 2: Intuitive Robot Programming

2 Intuitive Robot Programming

The IRP method is shown in figure 2. The process starts with the registration of the movements of a skilled operator, who performs the task manually. The registration results in a large set of data that needs to be filtered and reduced before it is edited to complete motions, translated into the internal language of the robot and sent to the robot. In the following sections the details of each part of the process will be presented.

2.1 Task acquisition

In IRP, the motions of the skilled operator are tracked at a rate about 100 Hz. The variety of sensors is large, but the three main types are described in the following table:

Mechanical / Dynamical sensors These sensors can be defined as those which measure mechanical quantities. Some examples of these are: force, torque, displacement, rotation, pressure, mass and density, among many others. Mechanical quantities are measured indirectly through their effect on sensor parameters which also are mechanical [14].

Magnetic sensors These sensors are, in one way or another, associated with the laws and effects of magnetic or electromagnetic fields. The latter utilize the laws of induction, and the former includes sensors that use the influence from a magnetic field on certain material properties [15].

Optical sensors The main property of these sensors is the ability to recognize structures based on one or several light sources. Say that an object's shape or motion is going to be tracked by an optical sensor. Then one or several LEDs are placed on the object and the sensor can obtain the desired information by recording the position of these LEDs. See e.g. [2]. Optical sensors have existed for quite some time. Telescopes is one of these, but the ones for industrial usage which are considerably younger, are still at the research and development level.

2.2 Filtration of noise

Due to the high frequency of the sampling, different kinds of noise are inevitable. Here are some of the sources:

Operator No matter how skilled the operator is, his movements will, due to the high frequency, be registered with some irregularities. Especially with movements around corners, one registers

many outliers. Another thing is that the filter ought to remove errors that may originate from a trembling hand.

Sensor The sensor readings themselves can be distorted by the surroundings. E.g. a magnetic sensor will be useless if the environment consists of large barriers of metal.

Smart filters must be constructed to deal with these problems. The filters look at subsets of the data, and based on means or other probability models like the gaussian, they could remove outliers from the subset. Another way of removing this erratic data is to look ahead in the dataset, and try to find a pattern that way. More on the Look-Ahead-filter in appendix C.

2.3 Reduction of points

After filtering there are still redundant data according to what's needed to program the robot to handle. Take the example of a operator that welds along a straight line. When this is tracked at the above mentioned rate, it results in a large number of unneccesary points. This is because if the robot is to move along the same line, it only need to be told where to start and stop. Exactly the same can be said about a movement along an arc. The robot only needs three points, the start, the end, and one control-point in the middle. This is so, because the robot has programs for interpolating along straight lines and arcs. This leaves us with the main problem for this diploma. How do one transform a large set of points to standard geometrical elements?
In section 5, this text focuses on algorithms that solve the above problem and in section 6 the algorithms are tested on particular datasets.

2.4 Editing

At this point in the process there is some recorded path(s) which is to be tested and/or edited. Whether the path(s) came from teaching the robot manually, was generated from a CAD program or originates from a sensor, is not the main concern at this point. The editing can be viewed as two separate tasks, although they both are being performed in the same software solution. The software solution is called SPORT^{OPEN}, and was developed by Andor [1]. The interface is shown in figure 3, and it clearly has similarities with CAD systems.
The first task concerns the editing of the path(s). When doing this, the program gives the worker the following options:

Import One can import paths, robots, and other objects.

Add/Modify glue The imported paths can be glued together by a glue function. This is by default a straigth line between the endpoint of path1 and starting point of path2, but this can be modified into e.g. a circular motion.

Animate One can animate the imported robot to follow one of the paths imported or several of them glued together to a larger path. The animation may include interaction with the other imported objects. This way one can verify that the robot does as intended in the new program without having to remove it from production, and there is no risk for crashing with the robot's surroundings either.

After this is done, the worker selects which type of robot will be doing the actual task. Different robots has different ways of solving a typical task, e.g. moving along a curved line. This means that the path generated in SPORT^{OPEN} will be cut into different segments depending on which type of robot that will be performing the task.

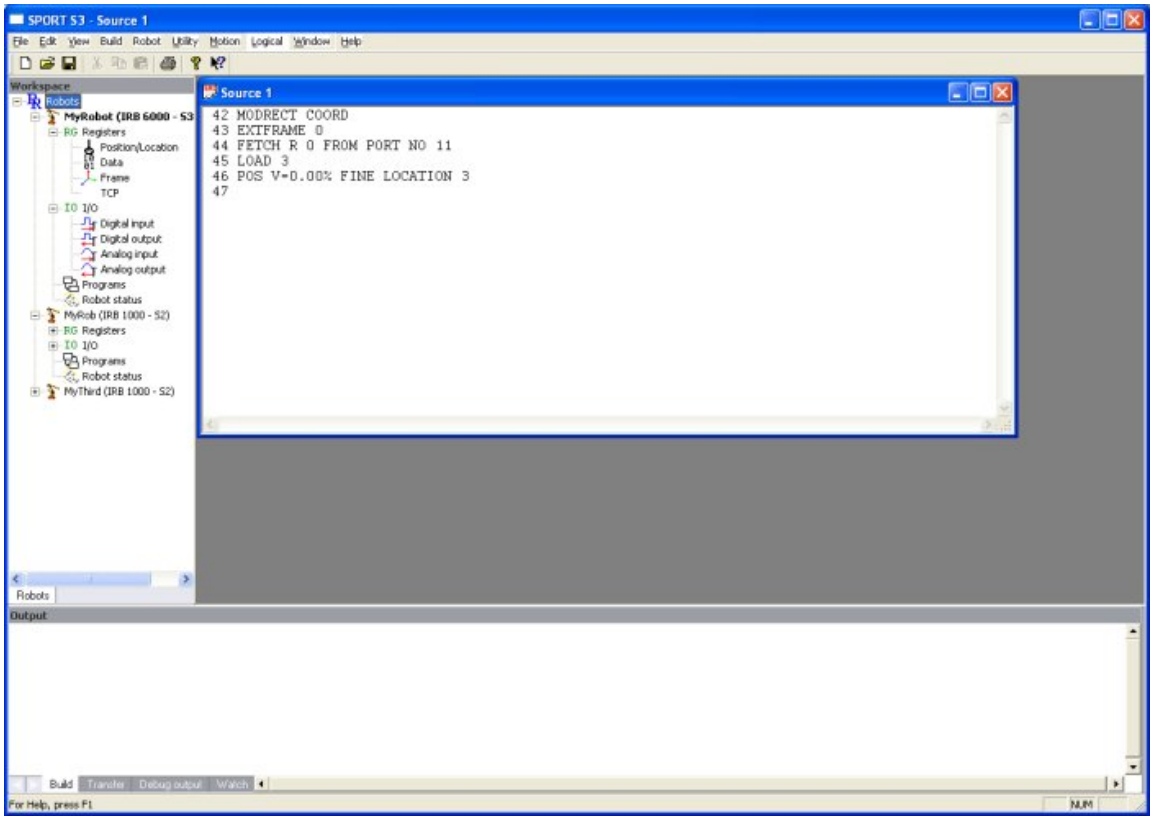


Figure 3: The current userinterface of SPORT^{OPEN}

2.5 Post processing

Based on the previous sections one finally has a complete program to send to the robot. The actual sending is done via Ethernet, RS232 or similar communication standards. What makes this point interesting is the variety of languages between robot manufacturers. Table 1 illustrates some examples.

Manufacturer	Language
NACHI	SLIM
ABB	Rapid
Motoman	Inform II
KUKA	KRL
GMF - Fanuc	TP/Karel 5

Table 1: Some robot manufacturers and their robot languages

Attempts to standardize the languages have been made, and two general robot languages are Industrial Robot Language (IRL) and Intermediate Code for Robots (ICR). The compiler basics of IRL is Pascal with facilities for the programming of robots. ICR is an intermediate code language that high level languages can be compiled into. [4]

3 Theory

3.1 Kinematics

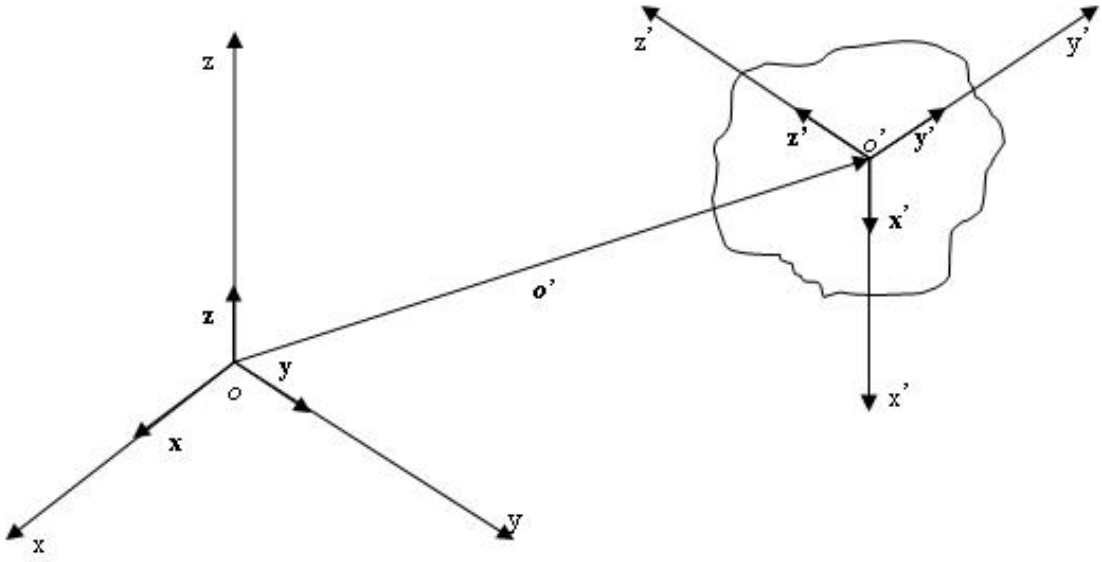


Figure 4: Position and orientation of a rigid body

A rigid body is described in space by its position and orientation with respect to a reference frame. The position is given by the 3×1 vector

$$o' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix} \tag{1}$$

With reference to figure 4, one can express the orientation of the rigid body as follows

$$\begin{aligned} \boldsymbol{x}' &= x'_x\boldsymbol{x} + x'_y\boldsymbol{y} + x'_z\boldsymbol{z} \\ \boldsymbol{y}' &= y'_x\boldsymbol{x} + y'_y\boldsymbol{y} + y'_z\boldsymbol{z} \\ \boldsymbol{z}' &= z'_x\boldsymbol{x} + z'_y\boldsymbol{y} + z'_z\boldsymbol{z} \end{aligned} \tag{2}$$

This leads to the Rotation Matrix

$$\mathbf{R} = \begin{bmatrix} \boldsymbol{x}' & \boldsymbol{y}' & \boldsymbol{z}' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}'^T\boldsymbol{x} & \boldsymbol{y}'^T\boldsymbol{x} & \boldsymbol{z}'^T\boldsymbol{x} \\ \boldsymbol{x}'^T\boldsymbol{y} & \boldsymbol{y}'^T\boldsymbol{y} & \boldsymbol{z}'^T\boldsymbol{y} \\ \boldsymbol{x}'^T\boldsymbol{z} & \boldsymbol{y}'^T\boldsymbol{z} & \boldsymbol{z}'^T\boldsymbol{z} \end{bmatrix} \tag{3}$$

This matrix is orthogonal, i.e. $\mathbf{R}^T\mathbf{R} = \mathbf{I}$ and $\mathbf{R}^T = \mathbf{R}^{-1}$. Any rotation can be expressed by a rotation matrix, and consecutive rotations are obtained by multiplying the $\mathbf{R}$'s for each rotation. The drawback for these matrices are redundancy. Each rotation matrix has nine elements, but to specify a given rotation one only need four parameters, three to specify the axis to rotate about, and

one for the angle of rotation. Based on this angle/axis representation one can derive the following equation[1]

$$\mathbf{R}(\theta, \boldsymbol{r}) = \begin{bmatrix} r_x^2(1-c_\theta)+c_\theta & r_x r_y(1-c_\theta)-r_z s_\theta & r_x r_z(1-c_\theta)+r_y s_\theta \\ r_x r_y(1-c_\theta)+r_z s_\theta & r_y^2(1-c_\theta)+c_\theta & r_y r_z(1-c_\theta)-r_x s_\theta \\ r_x r_z(1-c_\theta)-r_y s_\theta & r_y r_z(1-c_\theta)+r_x s_\theta & r_z^2(1-c_\theta)+c_\theta \end{bmatrix} \tag{4}$$

This rotation matrix has the property $\mathbf{R}(-\theta, -\boldsymbol{r}) = \mathbf{R}(\theta, \boldsymbol{r})$, resulting in nonuniqueness for this representation, which also leads to problems when solving the inverse problem. These drawbacks can be overcome by using unit quaternions instead of the angle/axis notation. More on quaternions in section 3.5.

3.2 Elementary rotations and Euler Angles

Any rotation can be expressed as a composition of three elementary rotations. An elementry rotation rotates the reference-frame around one of the coordinate axes. The rotation is defined positive when it is counter-clockwise. Figure 5 show an elementary rotation of α degrees about the z-axis.

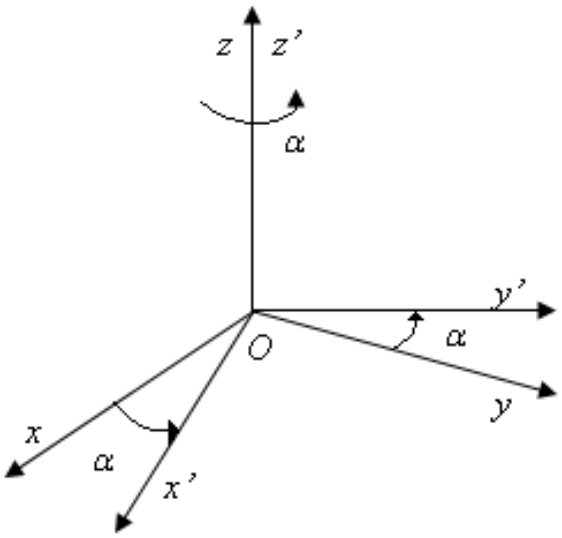


Figure 5: Elementary rotation of α degrees about the z-axis

The standard convention is to let the reference frame be given by the three coordinate axis x, y, z. Figures for the x- and y-axis similar to figure 5 can be used to deduce that the elementary rotations are given as the three matrices (5)-(7).

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{5}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{6}$$

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

[1] c_θ and s_θ is short for $\cos\theta$ and $\sin\theta$ respectively.

When a general rotation is given in the angle-axis representation as (4), the rotation is given by four parameters, three for the axis and one for the angle of rotation around this axis. With Euler Angles this can be reduced to the minimal representation of three parameters. Here the vector $\Phi = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}$ gives the rotation around the three coordinate axis. What is important to note here is that the rotation is defined by a composition of three elementary rotations, and not necessarily three different elementary rotations. ZYZ is a completely valid set of rotation axis for the Euler Angles, but ZYY clearly is not because $\mathbf{R}_y(\beta_2)\mathbf{R}_y(\beta_1) = \mathbf{R}_y(\beta_1 + \beta_2)$ which the following computation illustrates.

$$\begin{aligned} \mathbf{R}_y(\beta_1)\mathbf{R}_y(\beta_2) &= \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_2 \end{bmatrix} \\ &= \begin{bmatrix} c_1c_2 - s_1s_2 & 0 & c_1s_2 + s_1c_2 \\ 0 & 1 & 0 \\ -(s_1c_2 + c_1s_2) & 0 & c_1c_2 - s_1s_2 \end{bmatrix} = \begin{bmatrix} c_{1+2} & 0 & s_{1+2} \\ 0 & 1 & 0 \\ -s_{1+2} & 0 & c_{1+2} \end{bmatrix} = \mathbf{R}_y(\beta_1 + \beta_2) \end{aligned} \tag{8}$$

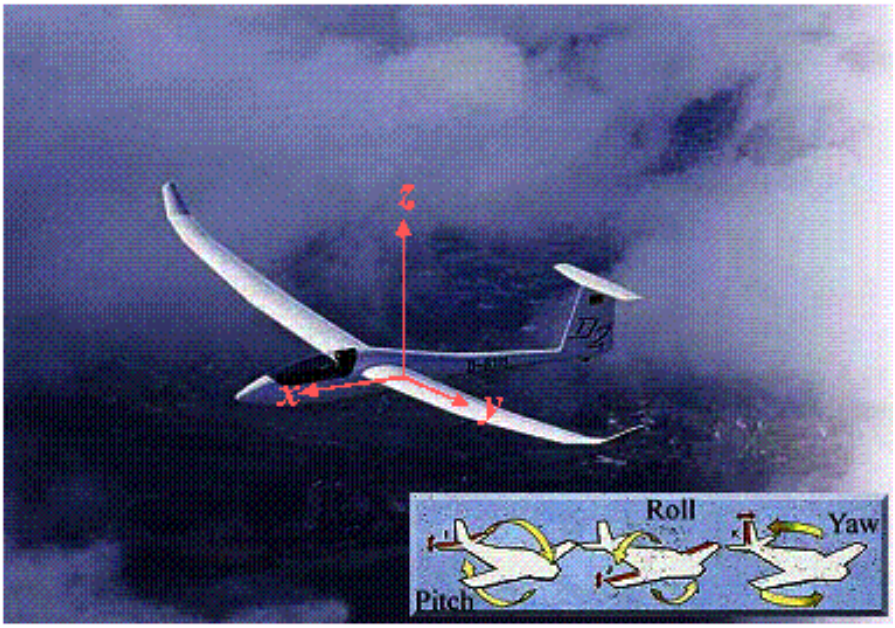


Figure 6: The Roll-Pitch-Yaw rotations around the aircrafts fixed frame

In the (aero)nautical field the commonly used set of Euler angles are ZYX, also known as *Roll-Pitch-Yaw* angles. In this system the rotations are defined with respect to a fixed frame attached to the (air)crafts center of mass. The frame and rotations' axis is shown in figure 6 and the rotation matrix based on the *Roll-Pitch-Yaw* angles is given as

$$\begin{aligned} \mathbf{R}(\Phi) &= \mathbf{R}_z(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi) \\ &= \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ s_\phi c_\theta & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix} \end{aligned} \tag{9}$$

For a general rotation matrix the *Roll-Pitch-Yaw* angles[2] are given by[3]

$$\begin{aligned} \phi &= \text{Atan2}(r_{21}, r_{11}) \\ \theta &= \text{Atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}), |\theta| \leq \frac{\pi}{2} \\ \psi &= \text{Atan2}(r_{32}, r_{33}) \end{aligned} \tag{10}$$

3.3 Homogeneous Transformations

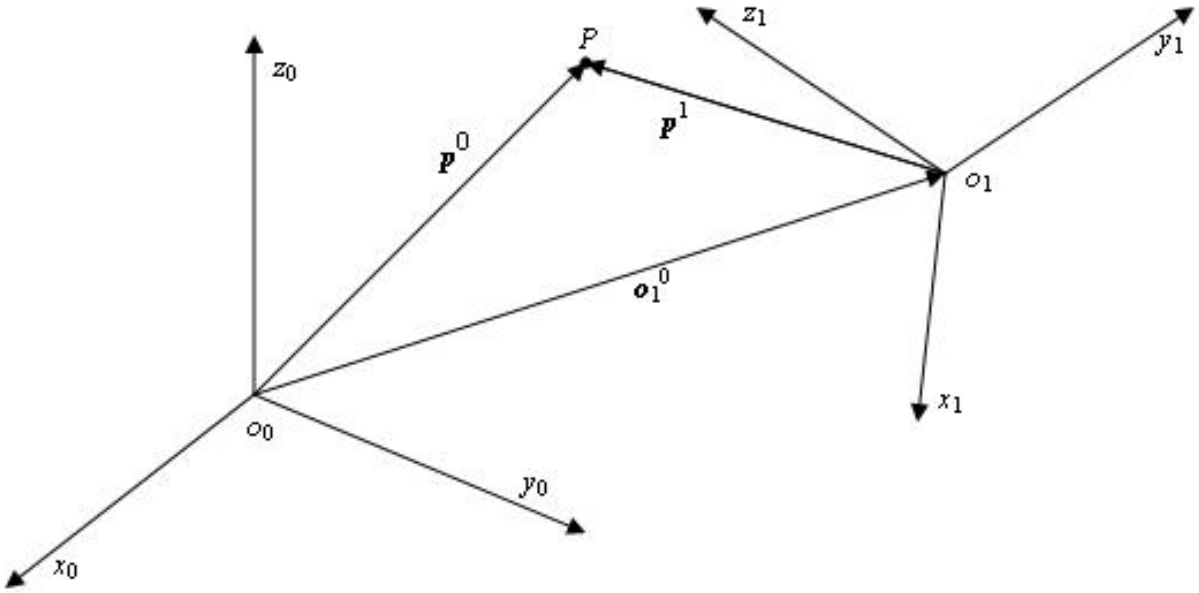


Figure 7: Representation of a point P in several frames

The main goal for this section is to obtain a mathematical expression for the kinematics of an arbitrary manipulator or robot. The point P in figure 7 can be expressed in both frame 0 and frame 1, but because it is the same point in space, the two representations must be equal. This leads to[4]

$$\mathbf{p}^0 = \mathbf{o}_1^0 + \mathbf{R}_1^0 \mathbf{p}^1 \tag{11}$$

Now one can introduce the homogeneous transformation matrix

$$\mathbf{A}_1^0 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{o}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{12}$$

Then (11) can be written as

$$\tilde{\mathbf{p}}^0 = \mathbf{A}_1^0 \tilde{\mathbf{p}}^1 \tag{13}$$

where we have used the homogeneous representation of the vector $\mathbf{p}$, i.e.

$$\tilde{\mathbf{p}} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{14}$$

[2]For another set of axes the rotation matrix would be different and hence also the inverse problem would have a different solution.

[3]The Atan2(y,x) computes the arctangent of y/x, but utilizes the sign of each argument to determine which quadrant the angle belongs to. The function is implemented in *math.h* in the standard C/C++ Library.

[4]The superscript denotes the frame the components are referenced too, and the subscript what frame is transformed into the reference system.

With the homogeneous transformation matrices and the Denavit Hartenberg convention, see appendix D, one can form the transformation matrix $\mathbf{T}_e^b(\mathbf{q})$ which gives the transformation from the *end-effector*-frame (e) to the *base*-frame (b) as a function of the joint variables $\mathbf{q}$.

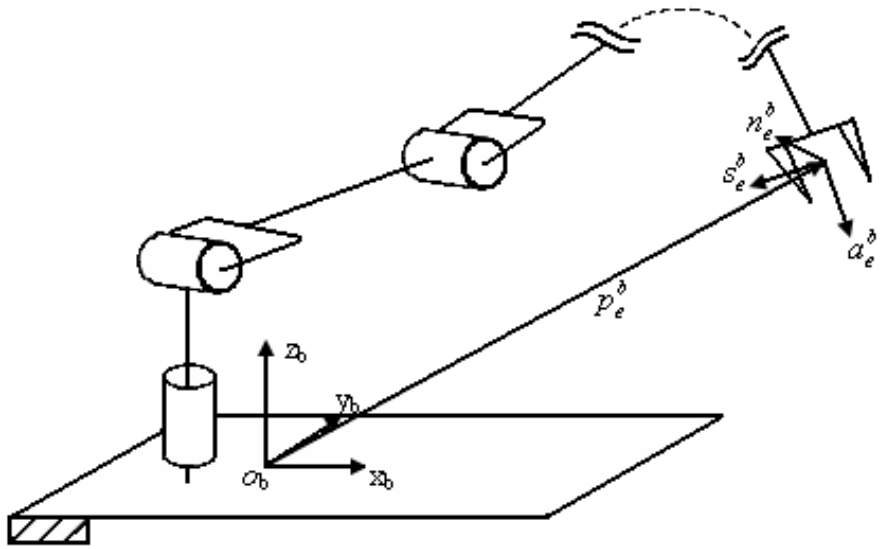


Figure 8: Vital coordinate frames of a manipulator

3.4 Coordinate systems

The various sets of coordinates are referenced to different coordinate systems. It is vital to the process that it is known which system the coordinates being studied is referenced to. If coordinates from different systems are to be compared, one of them has to be transformed to the other cordinate system. This should be done by the transformation matrices discussed in section 3.3. As examples of coordinate systems a brief list of the ones encountered in this context is given.

World This system is the most general system. The origin could be defined anywhere and the operator will set the origin of this frame where he find it useful.

Manipulator Figure 8 show the vital frames of a usual manipulator. The two frames shown is the base frame with subscript b, and the end-effector frame[5], or tool center frame with subscript e. The transformation beetween these two is usually done internally using the Denavit Hartenberg convention see appendix D. The transformation can be expressed as the rotation matrix R_e^b and the translation vector p_e^b giving the homogeneous transformation matrix

$$T_e^b = \begin{bmatrix} R_e^b & p_e^b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} n_e^b & s_e^b & a_e^b & p_e^b \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

[5]The axis in this frame is chosen as follows: a indicates the *approach* direction to the object, s is normal to a in the *sliding* plane, and n is the *normal* that completes the right-handed frame.

As an example of calculations with these matrices, consider three frames of a manipulator. Let them be base-frame, end-effector-frame and the third a j-th-frame which could be any frame beetween the two first. Assume now that the transformation from base- to j-frame T_j^b, and from the j- to end-effector-frame T_e^j is known. Then the transformation from base to end-effector is given by the following computation

$$T_e^b = T_j^b T_e^j = \begin{bmatrix} R_j^b & p_j^b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_e^j & p_e^j \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_e^b & p_e^b \\ 0 & 1 \end{bmatrix} \tag{16}$$

Similar calculations can be done for any pair of transformations, not necessarily internally on a manipulator.

3.5 Quaternions

The quaternion algebra was discovered by Sir William Rowan Hamilton, as he was searching for a three dimensional variant of the complex numbers. When he realized he needed three complex components instead of just two, he was so pleased that he scratched the fundamental formula into the stone of the Brougham Bridge.[16]

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{17}$$

$$\mathbf{ij} = \mathbf{k} = -\mathbf{ji}, \quad \mathbf{jk} = \mathbf{i} = -\mathbf{kj}, \quad \mathbf{ki} = \mathbf{j} = -\mathbf{ik}. \tag{18}$$

Every quaternion can be represented by one real, and three complex parameters. Two important representations are the linear combination and the (scalar, vector),

$$q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} = (\eta, \epsilon) \tag{19}$$

Any two quaternions can be added together componentwise, i.e.

$$q_1 + q_2 = (a_1 + a_2) + (b_1 + b_2)\mathbf{i} + (c_1 + c_2)\mathbf{j} + (d_1 + d_2)\mathbf{k} \tag{20}$$

Since the real numbers are commutative under addition i.e. $a + b = b + a$, the same applies for the quaternions. In algebraic terminology this means the set of quaternions with the binary operator $+$ denoted $\langle R, + \rangle$, is an abelian group[6].
By multiplying two quaternions q_1 and q_2, represented as polynomials or vectors, one can derive the following product formula

$$q_1 \otimes q_2 = \left(\eta_1\eta_2 - \epsilon_1^T\epsilon_2, \eta_1\epsilon_2 + \eta_2\epsilon_1 + \epsilon_1 \times \epsilon_2\right) \tag{21}$$

and with this rule it is easy to show that multiplication is associative. This means $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in \mathbb{H}$ where $\mathbb{H}$ is the set of quaternions, in memory of Sir Hamilton. One can also show the **left** and **right distributive laws** which say

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$
$$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$$

The quaternion **ring** is now established due to the following definition [7]

Definition 3.5.1. *A **ring** $\langle R, + \rangle$ is a set R together with two binary operations $+$ and $\cdot$, which we call* addition *and* multiplication*, defined on R such that the following axioms are satisfied:*

[6] A group $\langle G, * \rangle$ is a set G, closed under its binary operation $*$, i.e. if $a, b \in G \Rightarrow a * b \in G$. The operator must be associative; existence of identity element i.e. $\forall x \in G \exists e \in G$ s.t. $x * e = e * x = x$; and existence of inverse elements i.e. $\forall a \in G \exists a' \in G$ s.t. $a * a' = a' * a = e$.

$\mathfrak{R}_1$ *$\langle R, + \rangle$ is an abelian group.*

$\mathfrak{R}_2$ *Multiplication is associative.*

$\mathfrak{R}_3$ *For all $a, b, c \in R$, the left and right distributive laws hold.*

In the definition of a group the existence of an identity element is required. For the binary operator $+$ this is called the identity. In the quaternion ring the zero quaternion has this property. But also for the binary operator $\cdot$ there is such an element. The multiplicative identity element is called **unity**. The element $(1, 0, 0, 0)$ has this property with respect to the multiplication defined in (21). Based on the same equation it is easy to verify that the multiplicative inverse of a quaternion is its conjugate, and hence all quaternions except the zero quaternion has multiplicative inverses. Equation (18) states that quaternion multiplications does not commute, i.e. $q_1 q_2 \neq q_2 q_1$ generally. This discussion has shown that the quaternions define the strictly skew field $\langle \mathbb{H}, +, \otimes \rangle$ due to the following definition [7]

Definition 3.5.2. *Let* R *be a ring with unity* $1 \neq 0$*. An element* u *in* R *is a* ***unit*** *of* R *if it has a multiplicative inverse in* R*. If every nonzero element of* R *is unit, then* R *is a* ***division ring*** *(or* ***skew field****). A* ***field*** *is a commutative division ring. A noncommutative division ring is called a* ***strictly skew field****.*

The elements in $\mathbb{H}$ have properties like the conjugate and norm just like the complex numbers.

$$\bar{q} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k} \tag{22}$$

$$|q| = a^2 + b^2 + c^2 + d^2 \tag{23}$$

One important multiplicative subgroup of the quaternions is the unit quaternions $\langle \mathbb{H}, \otimes \rangle$, i.e. those with $|q| = 1$. They form only a sub**group** because they are not closed under addition which is shown by the following computation

$$\mathbf{i} + \mathbf{j} = (0, 1, 0, 0) + (0, 0, 1, 0) = (0, 1, 1, 0) \tag{24}$$

where it should be clear that $(0, 1, 1, 0)$ is not a unit quaternion because $|(0, 1, 1, 0)| = \sqrt{2} \neq 1$.
The unit quaternions can represent rotations in 3-dimensional space. For a given quaternion (η,ϵ), the axis of rotation equals ϵ and the angle of rotation θ are given by:

$$\theta = 2 \cos^{-1}(\eta) \tag{25}$$

The rotation matrix based on the quaternion (η,ϵ) is given by:

$$\mathbf{R}(\eta, \epsilon) = \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x \epsilon_y - \eta \epsilon_z) & 2(\epsilon_x \epsilon_z + \eta \epsilon_y) \\ 2(\epsilon_x \epsilon_y + \eta \epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y \epsilon_z - \eta \epsilon_x) \\ 2(\epsilon_x \epsilon_z - \eta \epsilon_y) & 2(\epsilon_y \epsilon_z + \eta \epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix} \tag{26}$$

For the inverse problem, the following formula gives the quaternion based on any rotation matrix [11]:

$$\eta = \frac{1}{2} \sqrt{r_{11} + r_{22} + r_{33} + 1} \tag{27}$$

$$\epsilon = \frac{1}{2} \begin{bmatrix} \operatorname{sgn}(r_{32} - r_{23}) \sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \operatorname{sgn}(r_{13} - r_{31}) \sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \operatorname{sgn}(r_{21} - r_{12}) \sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \tag{28}$$

3.6 Localization of roots

In the 3D-algorithm which will be discussed in section 5.2 the challenge of localization of the roots of a polynomial is encountered. More specific: Is the real zeros, if any, localized on a given interval? This section will present some of the theory associated with the determination of those x satisfying the equation

$$f(x) = \sum_{i=0}^{n} a_i x^i = 0 \quad \text{where } a_m \neq 0. \tag{29}$$

First look at these two theorems [10] and [9].

Theorem 3.6.1 (Descartes). *In equation* (29) *the number of positive roots equal the number of times the sign changes in the sequence*

$$a_0, a_1, \ldots, a_n$$

or a multiple of two less. The number of negative roots is found in the same way by considering $f(-x)$.

Theorem 3.6.2 (Budan and Fourier). *Given the real polynomial* $f(x)$ *of degree* n, *let* V_x *represent, for any real* x, *the number of variations of sign in the sequence*

$$f(x), f'(x), \ldots, f^{(n)}(x).$$

Then if $\alpha < \beta$ *and* $f(\alpha)f(\beta) \neq 0$, *the number of real zeros of* $f(x)$ *on the interval* (α, β) *is not greater than* $V_\alpha - V_\beta$ *and can differ only by an even integer.*

Especially the theorem of Budan and Fourier is interesting because it says something about the existence of roots on an interval. The problem is that there might be a multiple of two less, and if one find that there is 2 or 4 roots on a given interval one does not have much information at all.

Definition 3.6.3. *A sequence of real polynomials*

$$f_0(x), f_1(x), \ldots, f_m(x) \tag{30}$$

will be said to form a Sturm sequence *in the weak sence on the interval* $[\alpha, \beta]$ *in case the following is true:*

1. *No two consecutive polynomials in the sequence vanish simultaneously on the interval.*
2. *If* $f_j(r) = 0$ *for* $j < m$, *then* $f_{j-1}(r)f_{j+1}(r) < 0$.
3. *Throughout the interval* $[\alpha, \beta]$, $f_m(x) \neq 0$.
4. *If it is also true that if* $f_0(r) = 0$, *then* $f_0'(r)f_1(r) > 0$, *then it will be called a* Sturm sequence in the strict sense, *or just a Sturm sequence, for* $f(x) = f_0(x)$.

If the degree of $f_1(x)$ is not greater than that of $f_0(x)$, then the remaining polynomials of a Sturm sequence can be formed by the Euclidean algorithm, thus,

$$\begin{array}{rcl} f_0(x) & = & q_1(x)f_1(x) - f_2(x), \\ f_1(x) & = & q_2(x)f_2(x) - f_3(x), \\ \cdots & = & \cdots \\ f_{m-2}(x) & = & q_{m-1}(x)f_{m-1}(x) - f_m(x) \end{array} \tag{31}$$

where $f_m(x)$ is a polynomial, possibly a constant, which nowhere vanishes on the interval (α, β).

Theorem 3.6.4 (Sturm). *If* $f_1(x) \equiv f_0'(x)$ *and the sequence* (30) *is formed as in* (31) *where* $f_m(x)$ *retais a fixed sign on the interval* (α, β), *then the number of times* $f_0(x)$ *vanishes on the interval* (α, β) *is exactly equal to* $V_\alpha - V_\beta$ (*that is, a zero of whatever multiplicity is counted only once*).

4 Data considerations

When the operator is going to teach a robot a specific task he has some knowledge of the path. This info can be used to improve the tracking and/or filter the dataset for errors. The following subsections discuss some aspects of the alternative paths.

4.1 Free tracking

This is the general case where the path that is tracked is of a general three-dimensional form. In this case the sensor must track both position and orientation of the tool along the path. This results in a dataset $\{(x, y, z, q[4], t)_i | i = 1 \dots N\}$, a three-dimensional position, an orientation represented by a unit quaternion and a timestamp for each sampling. The challenge for the algorithm that try to reduce the total number of points is not to remove the structure of the path. Ideally it is not supposed to cut corners, but for straight lines only two points are needed. In addition the information about the orientation of the tool has to be preserved during the reduction.

4.2 Planar objects

At this point it is known that the path to be followed is along a planar object or surface, but with coordinates given in three dimensions. Now the idea is as follows: If one only track the position for the path along this object, then one could have an algorithm to insert a normal coordinate frame at each point. After this is done one could set different tolerances in each of the normal frame's axis, and generate motions based on this or check if new trackings with orientation is legal or not.
The tracking is done by hand so the set will generally not be perfectly planar. There are two ways to deal with this challenge:

1. Predefining of the plane. The operator can use his metainformation about the object, e.g. this is a plane, and give three or four external points to define the plane. After the tracking, each point in the set can be projected into the predefined plane.

2. Another way to deal with it is to use a selection of points from the set to estimate the objects plane.

The 2D-Algorithm in section 5.1 focuses on the second method, but the mathematics in the projection routine will be equal for both methods.

4.3 The datasets

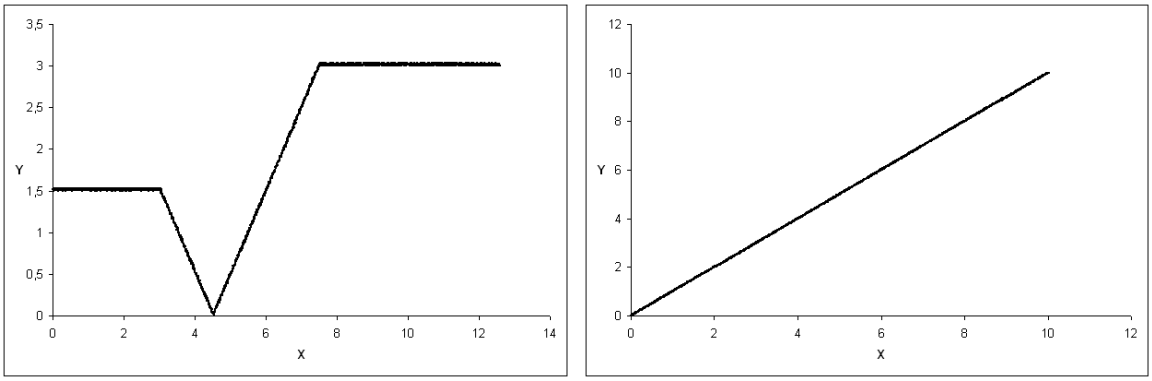
During the development and testing of the filtration algoritms various test-sets were used. These sets were generated in a non-physical environment and are therefore dimensionless. Visualization of multidimensional sets are not easy but the selected plots show the main theme before and after running the various algorithms on the sets.

4.3.1 The 3D-sets

For the 3D-algorithm two datasets were used; *corners set* and *line set*. The sets originally contained 3000 points in (x, y, z) coordinates. At each point a small random error was added to represent the error that might occur when tracking a human motion. Figures 9(a) and 9(b) show the x-y-plane of the sets, and the same plane is used in the results section.

4.3.2 The orientation set

For the orientation algorithms a set called angles were used for testing. It was a set of 1000 points on the form $(x, y, z, q[4], t)$, thought x, y and z were not considered in the filtering. Its first Euler



(a) The corners set (b) The line set

Figure 9: The original x-y-plane of the datasets used during testing of the 3D algorithm.

angle (ϕ) is shown as a function of time in figure 10. The Euler angle is based on the quaternion at each point translated via a rotation matrix by equation (26) and then into a set of Euler angles by the equations given in (10).

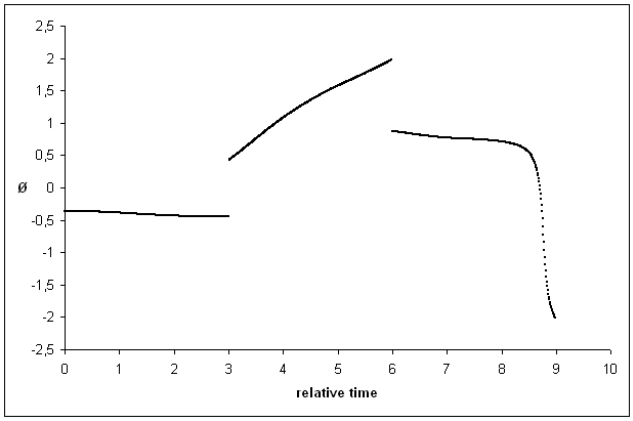
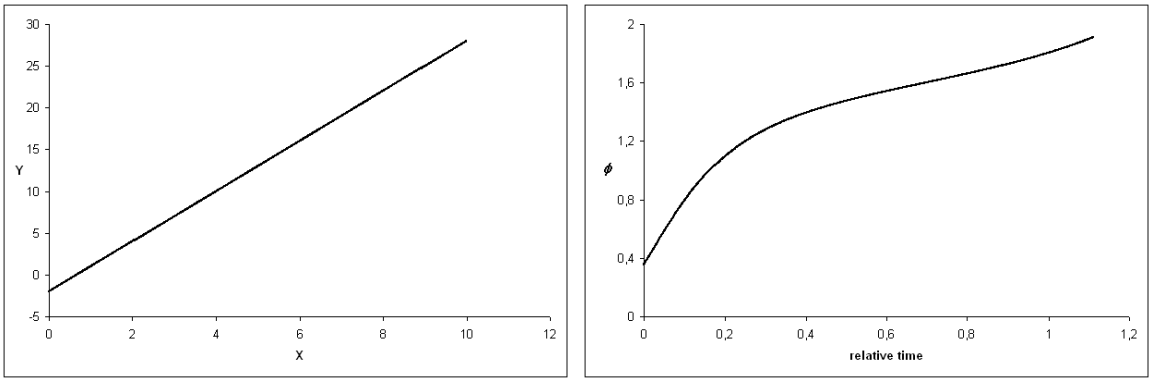


Figure 10: The original $\phi(t)$ for the angles set.

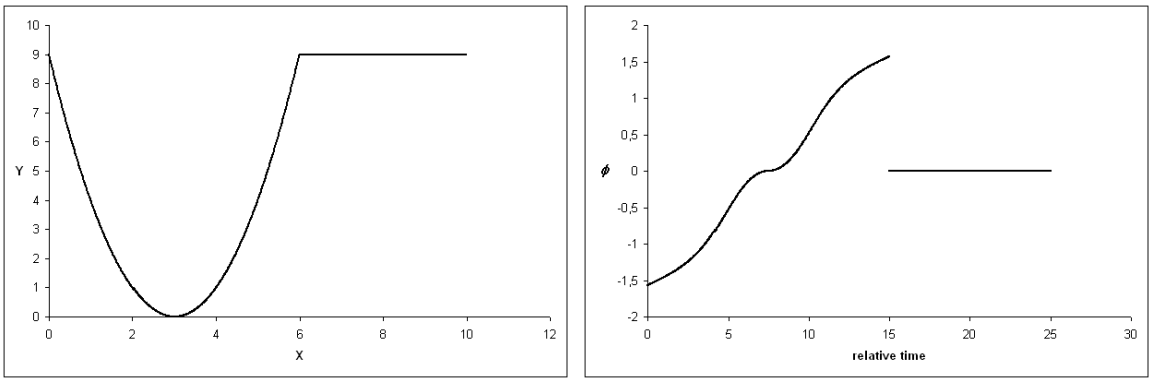
4.3.3 The joining sets

During testing of the joining algorithm two sets were used. The first called *simple* and the other one *complex*. These sets consisted of 1000 points on the form $(x, y, z, q[4], t)$. In the plotting of these sets, the XY-plane and the first Euler angle (ϕ) is shown as a function of time. See figures 11 and 12.



(a) XY-plane (b) $\phi(t)$

Figure 11: Original plot of the *simple* set used in testing of the joining algorithm



(a) XY-plane (b) $\phi(t)$

Figure 12: Original plot of the *complex* set used in testing of the joining algorithm

5 Algorithms

5.1 2D-Algorithm

This algorithm will find the actual plane of a given two-dimensional object based upon a selection of points from a tracking along the border of the surface. After determining the plane it will insert a normal coordinate frame at each point on the given path, illustarted in figure 13. Each of these frames will be given as a unit quaternion, representing the transformation from an external world-frame, to the normal-frame at each point.

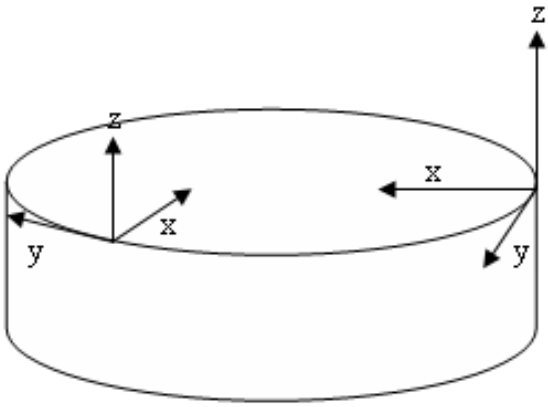


Figure 13: The normal-frame. X-axis: Normal to the path. Y-axis: Tangential to the path. Z-axis: Normal to the plane.

5.1.1 Sorting vectors/planes

The equation for a plane through the point $p = (x_0, y_0, z_0)$ with normal $\mathbf{n} = \langle a, b, c \rangle$ is given by

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0 \tag{32}$$

For a given point $p_i = (x_i, y_i, z_i)$ outside the plane the equation give a positive or negative result, and hence the point can be categorized as above or below the given plane. Given a set of vectors defining approximately the same plane, they can be sorted by using the quicksort (appendix A) due to how many points they have above or below.

5.1.2 Projecting points into planes

In this procedure the normal vector of the plane $\mathbf{n} = \langle a, b, c \rangle$ and one point in the plane $p_0 = (x_0, y_0, z_0)$ is known. The point $p_1 = (x_1, y_1, z_1)$ will be projected into the plane given by $\mathbf{n}$ containing p_0. To simplify the equations the coordinate system is selected in a way so that p_0 is translated to the origin, and the point p_1 will then be translated to another point called b. The normal vector stays the same. Figure 14 illustrates the situation. The idea is to find the vector p and subtract this from b. From [12] it is known that the projection p of a point b onto a line given by the vector a is

$$p = \bar{x}a = \frac{a^T b}{a^T a} a \tag{33}$$

The projection p is the component along the normal-vector. Subtracting p from b gives the component of b in the plane normal to $\mathbf{n}$. Translating the coordinates back to their original, gives the final

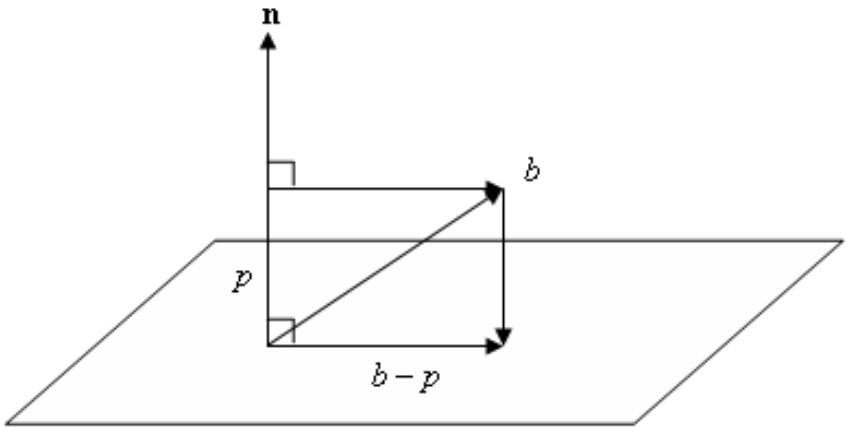


Figure 14: The projection of p_1 into the plane given by normal-vector $\mathbf{n}$

expression for $\hat{p}_1$, i.e. the projection of p_1 in the plane normal to $\mathbf{n}$.

$$\hat{p}_1 = b - p + p_0 = b - \frac{n^T b}{n^T n} n + p_0 = p_1 - \frac{n^T (p_1 - p_0)}{n^T n} n \tag{34}$$

Here the last equality uses the fact that $b = p_1 - p_0$.

5.1.3 The algorithm

To determine the best aproximation to the plane the algorithm selects the one in the middle in the array of sorted planes, also known as the median.

1. Select 5 vectors from the first point in the set to points around first quarter of the set, and another 5 from the first point to the last quarter in the set.

2. Cross each first-quarter vector with a last-quarter vector all being a candidate for the plane's normal vector.

3. The 5 normal vectors can be sorted with the method in 5.1.1. The assumption here is that the best approximation will have the same number of points above as below hence the median is the best one.

4. The data of the set can now be projected into this plane as discussed in section 5.1.2.

5. Calculate the path's tangent vector, i.e. the vector beetween two consecutive points under the assumption that they are close enough.

6. Since the XYZ-frame is a righthanded system, the following holds: $\mathbf{y} \times \mathbf{z} = \mathbf{x}$. In other words, knowing the normal Z-axis, and the Y-axis at each point, the X-axis can be calculated by the cross-product between Y and Z.

5.1.4 An alternative approach

It is clear that the algorithm described above has limitations. The normal vector is calculated using only a selection of the given set. If the selection is not representative for the whole set, the plane will not be approximated as well as it should. The best would be to find a way to use the entire set for the approximation of the normal vector. The following method which is based on the single value

decomposition (SVD) accomplishes this.
The given set can be written as the matrix A where each datapoint is a comlumn in A.

$$\mathbf{A} = \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ z_1 & \cdots & z_n \end{bmatrix} \tag{35}$$

This matrix can be decomposed into $\mathbf{A} = \mathbf{U\Sigma V}$ where the columns of $\mathbf{U}(3 \times 3)$ are eigenvectors of $\mathbf{AA^T}$ and the rows of $\mathbf{V}(n \times n)$ are eigenvectors of $\mathbf{A^TA}$. The r singular values on the diagonal of $\mathbf{\Sigma}(3 \times n)$ are the square roots of the nonzero eigenvalues of both $\mathbf{AA^T}$ and $\mathbf{A^TA}$ [12].

$$\mathbf{AA^T} = \mathbf{U\Sigma V V^T \Sigma^T U^T} = \mathbf{U\Sigma\Sigma^T U^T} = \mathbf{U} \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix} \mathbf{U^T} \tag{36}$$

In this case the matrix $\mathbf{A}$ is a set of vectors which ideally lie in a plane. To define a plane, only two linearly independent[7] vectors are needed. This is called the rank of the matrix, and the vectors are said to *span* what in this case is a plane. When $\mathbf{A}$ is expressed as $\mathbf{U\Sigma V}$ the rank of $\mathbf{U}$ equal the rank of $\mathbf{A}$ and they span the same space. The two fist columns in $\mathbf{U}$ therefore span the plane.
In the ideal case when $\mathbf{A}$ consist of points in the x-y-plane the third row is zero and hence the third singular value in $\mathbf{\Sigma}$ is zero. When the point almost lie in one plane the singular value will be close to zero. Post multiplication the above equation with $\mathbf{U}$ give

$$\mathbf{AA^TU} = \mathbf{U\Sigma\Sigma^T} \quad \text{or} \quad \mathbf{BU} = \mathbf{U\Lambda} \tag{37}$$

i.e. an eigenvector, eigenvalue problenm. If $\sigma_3^2 = 0$ the last column of the matrix on the right hand side would be the zero vector. But this would mean that the third column vector of $\mathbf{U}$ spans the Nullspace of $\mathbf{AA^T}$, which is the same as the normal vector to the plane spanned by the two first columns.
The problem of finding the normal vector og the plane has been transformed into finding the third column of $\mathbf{U}$ in the single value decomposition. An implementation of this can be found in appendix F.1.

5.2 3D-Algorithm

This algorithm is the three-dimensional extension of the Paraperp algorithm (discussed in [8]) with the alternative tolerance procedure. Figure 15 show the flow of the algorithm and selected parts of it are discussed in more detail below.

5.2.1 The System of Equations

The input to the 3D-Algorithm consists of a set of points given in the coordinates $\{x, y, z\}$. To find an approximation, the algorithm expresses each coordinate as a polynomial $x_t = x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$. For each coordinate the first four points give the following system of equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{38}$$

which can be solved exactly for the parameters $\{a_i | i = 1, \ldots, 4\}$. Having n points is the same as n equations, resulting in an $n \times 4$ matrix $\mathbf{A}$ in the system $\mathbf{Ax} = \mathbf{b}$, and we can only get an approximate

[7] A set of vectors $\{v_j\}$ are linearly independent if the sum $c_1 v_1 + \cdots + c_k v_k$ is zero only when the coefficients $c_1 = c_2 = \cdots = c_k = 0$.

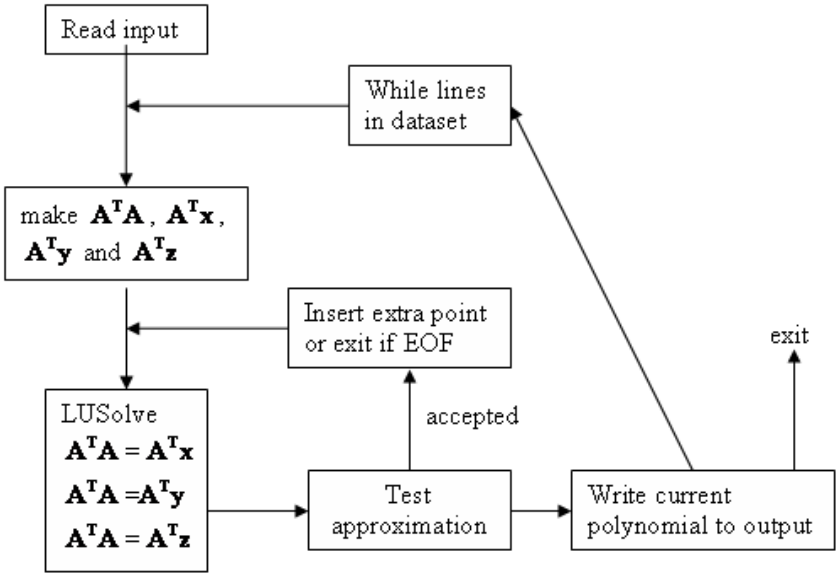


Figure 15: Flow of the 3D-Algorithm

solution of the system. This solution is given by the normal equations:

$$\mathbf{A^TAx = A^Tb} \tag{39}$$

In the 3D-Algorithm this system is solved by LU-Factorization a method which is assumed known to the reader. Having found the solution $\mathbf{x} = \{a_i\}$ one can check the approximation with the given tolerance criteria.

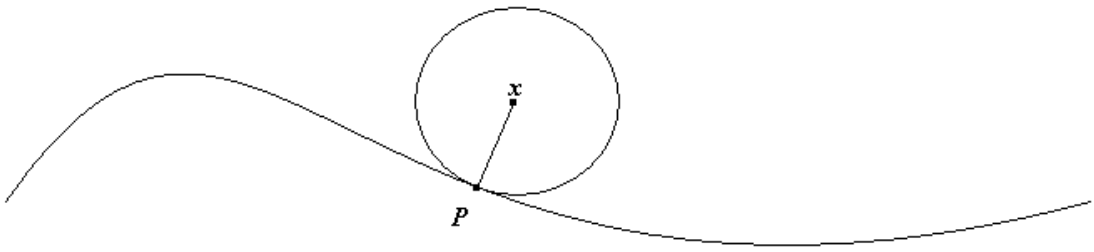


Figure 16: The tolerance criteria for the 3D-algorithm

5.2.2 The Tolerance Routine

Figure 16 shows the approximation curve and a sphere around a datapoint x, in 3-dimensional space. As the figure illustrates the sphere touches the curve in a single point P. Here the tangent of the curve lies in the tangent plane of the sphere. This means that the radius of the sphere is tangent to the curve, and hence, it is the smallest distance from x to the line. Based upon these facts the

tolerance routine will be a question of whether or not the curve is inside the sphere. The expression for a sphere with origin (x_0, y_0, z_0), and the parametrization of the curve is shown in equations (40) through (43).

$$\begin{aligned} r^2 &= (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 && (40)\\ x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 && (41)\\ y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^3 && (42)\\ z(t) &= c_0 + c_1 t + c_2 t^2 + c_3 t^3 && (43) \end{aligned}$$

Inserting each polynomial into the equation for the sphere, we get the sextic polynomial

$$\begin{aligned} f(t) &= (a_3^2 + b_3^2 + c_3^2)t^6 + 2(a_2a_3 + b_2b_3 + c_2c_3)t^5 \\ &+ \left[2(a_1a_3 + b_1b_3 + c_1c_3) + a_2^2 + b_2^2 + c_2^2\right] t^4 \\ &+ 2\left[a_3(a_0 - x_0) + b_3(b_0 - y_0) + c_3(c_0 - z_0) + a_1a_2 + b_1b_2 + c_1c_2\right] t^3 \\ &+ \left[2(a_2(a_0 - x_0) + b_2(b_0 - y_0) + c_2(c_0 - z_0)) + a_1^2 + b_1^2 + c_1^2\right] t^2 \\ &+ 2\left[a_1(a_0 - x_0) + b_1(b_0 - y_0) + c_1(c_0 - z_0)\right] t + (a_0 - x_0)^2 + (b_0 - y_0)^2 + (c_0 - z_0)^2 \end{aligned} \quad (44)$$

whose roots can be calculated by Newton's method. In [3] the following derivation is given: Suppose that $f \in C^2[a, b]$. Let $\bar{x} \in [a, b]$ be an approximation to p such that $f'(\bar{x}) \neq 0$ and $|p - \bar{x}| < \epsilon$ for some $\epsilon > 0$. The first Taylor polynomial for $f(x)$ expanded about $\bar{x}$ is then

$$f(x) = f(\bar{x}) + (x - \bar{x})f'(\bar{x}) + \frac{(x - \bar{x})^2}{2} f''(\xi(x)) \quad (45)$$

where $\xi(x)$ lies between x and $\bar{x}$. With $f(p) = 0$ and $x = p$ equation (45) becomes

$$0 = f(\bar{x}) + (p - \bar{x})f'(\bar{x}) + \frac{(p - \bar{x})^2}{2} f''(\xi(p)) \quad (46)$$

Newton's method is derived on the assumption that since $|p - \bar{x}| < \epsilon$ the term involving $(p - \bar{x})^2 \ll \epsilon$, so

$$0 \approx f(\bar{x}) + (p - \bar{x})f'(\bar{x}) \quad (47)$$

and solving for p then gives

$$p \approx \bar{x} - \frac{f(\bar{x})}{f'(\bar{x})} \quad (48)$$

Newton's method starts with a initial approximation p_0 and generates the sequence $\{p_n\}_{n=0}^{\infty}$, by

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{for } n \geq 1 \quad (49)$$

In this case, finding a solution to $f(t) = 0$ means that the curve at least touches the sphere, and the tolerance critera is satisfied. If Newton's method fails there exists only complex solutions which are of no interest in this context.

5.2.3 Adding a new point to the system

When the approximation satisfies the tolerance criteria, the algorithm will to add another point from the set to the existing system. Here the mathematics behind this is shown. Given $\mathbf{A^T A}$, $\mathbf{A_{n+1}^T A_{n+1}}$ is found from the following calculations

$$\mathbf{A_{n+1}^T A_{n+1}} = \begin{bmatrix} \mathbf{A^T} & a_{n+1}^T \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ a_{n+1} \end{bmatrix} = \mathbf{A^T A} + a_{n+1}^T a_{n+1} \quad (50)$$

where $a_{n+1} = \begin{bmatrix} 1 & n+1 & (n+1)^2 & (n+1)^3 \end{bmatrix}$ resulting in

$$a_{n+1}^T a_{n+1} = \begin{bmatrix} (n+1)^0 & (n+1)^1 & (n+1)^2 & (n+1)^3 \\ (n+1)^1 & (n+1)^2 & (n+1)^3 & (n+1)^4 \\ (n+1)^2 & (n+1)^3 & (n+1)^4 & (n+1)^5 \\ (n+1)^3 & (n+1)^4 & (n+1)^5 & (n+1)^6 \end{bmatrix} \tag{51}$$

which is easy to calculate. In a similar manner the new $\mathbf{A^T b}$ is given by

$$\mathbf{A_{n+1}^T b} = \left[\mathbf{A^T b}\right] + \begin{bmatrix} (n+1)^0 x_{n+1} \\ (n+1)^1 x_{n+1} \\ (n+1)^2 x_{n+1} \\ (n+1)^3 x_{n+1} \end{bmatrix} \tag{52}$$

5.3 Orientation

Not only the position, filtered in section 5.2, is important when controlling a robot arm. Especially in welding the orientation, i.e. the direction the tool is ponting, is an essential part of the task at hand. By discussing some reduction criteria the following sections will focus on this challenge.

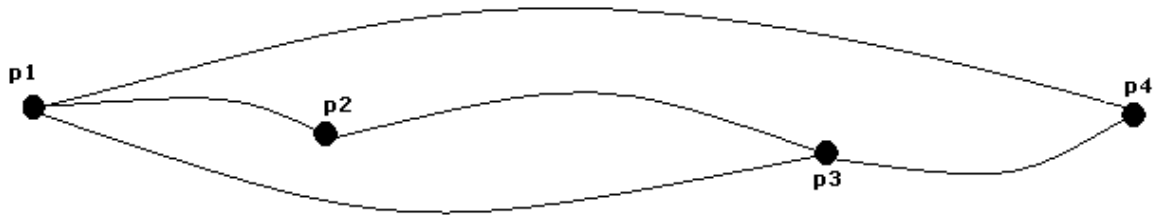


Figure 17: Points in dataset to be filtered

5.3.1 The problem

The orientation of the robot arm is represented by a unit quaternion, see section 3.5. This results in four additional coordinates to consider during the filtration which is the high level problem.
During its movement it is desired that a robot has as little angular acceleration as possible along its path. In other words a movement with constant velocity. Figure 17 show four points in a dataset which will be filtered. At each of these point the set has 8 coordinates, a three-dimensional position a unit quaternion giving a coordinate frame transformation, and finally a timestamp. Between each of the points there is a difference in the orientation. This results in an angular velocity when the robot moves between them. The velocity between two ponts A and B will be denoted by ω_{AB}. In the example figure we assume that the orientation in point p_3 differs considerably from the other three. This could be expressed in the following way

$$\omega_{34} > \omega_{14} + \delta\omega \tag{53}$$

where $\delta\omega$ is some tolerance. When p_3's orientation differs from that of p_1 and p_4 it has to be taken into the filtered set. On the other hand for p_2 we assumed it had approximately the same orientation

as p_1, resulting in

$$\omega_{23} \approx \omega_{13} \quad \text{when } \omega_{12} \approx 0 \tag{54}$$

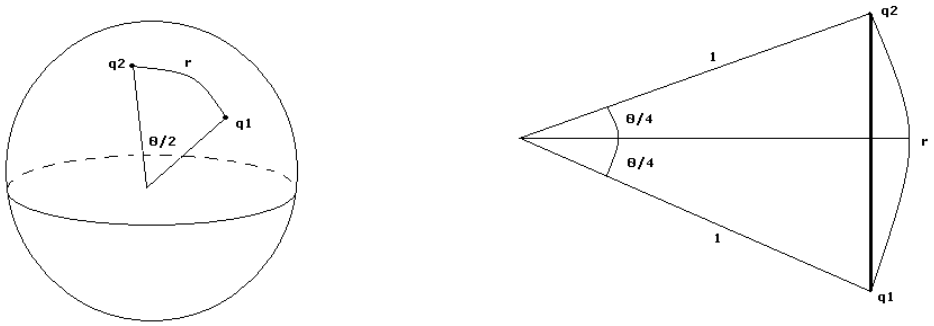
and hence p_2 could be removed from the set[8].

5.3.2 Measuring distance between unit Quaternions

Let a 4-dimensional sphere represent the group of unit quaternions. See figure 18(a). Given two elements on this sphere q_1 and q_2, there exists an unit quaternion r, representing the transformation of q_1 to q_2. Expressed mathematically[9]

$$\begin{aligned} r \otimes q_1 &= q_2 \\ r &= q_2 \otimes \bar{q_1} \end{aligned}$$

Now the angle of r, i.e. $\theta_r = 2 \arccos \eta_r$, equals the arclength between q_1 and q_2 when the angle is given in radians. This is so because the unit quaternions are on the surface of a unit sphere. As figure 18(b) illustrates, the Euclidean distance between the quaternions is given by the expression $2 \sin(\theta_r/4)$.



(a) 4D unit sphere (b) Euclidean distance

Figure 18: Measuring distances in the unit quaternion group

5.3.3 The Orientation Algorithm

Throughout the text this algorithm may also be referenced to as the quaternion reduction algorithm. The procedure for the algorithm will be recursive and work in the following way:

1. Determine angular velocity ω_{0N} between starting and ending point based on the Euclidean distance.
2. Compare this global velocity to local velocities ω_{0i}
3. If ω_{0i} exceeds a given tolerance, divide the large interval into 2 subintervals connected at i
4. Repeat the above process for each subinterval, i.e.
   - Determine semiglobal angular velocity
   - Compare local velocities on subinterval, and if necessary divide into smaller subintervals

[8]The position has not been considered so the statement would be wrong in general.
[9]The symbol $\otimes$ was defined as the quaternion-product in equation (21).

5.4 Euler reduction

The results from the testing of the quaternion reduction algorithm shown in section 6.2 were not promising. In most applications however the quaternions are not user friendly for the robot operator. Mainly because they are not physically related. An approach via Euler Angles was suggested for the reduction of orientation. This section will present such an algorithm.

5.4.1 Theoretical aspects

As it was for the orientation algorithm, figure 18(a) is the starting point for this algorithm as well, but this time the physical meaning of the transformation quaternion r is considered. Any quaternion can be transformed into a rotation matrix by equation (26), and from there into a set of Euler angles by equation (10). No matter which one of these representations are chosen, the physical interpretation of them is the same.
The quaternion for each point in the set is an mathematical way to express the orientation of the coordinate system of the tool at that point. The quaternion r, represents the orientation of the second coordinate system (q_2) expressed in the coordinates of the first system (q_1). Using Euler angles for the representation gives the rotation about the three axis to transform the first coordinate system into the second.
The difference between the two timestamps at q_1 and q_2 and the Euler angles make it possible to compute the angular velocity that was mentioned as part of the problem in section 5.3.1.
Uncertainties about how operator will state the tolerances, gave rise to two algorithms.

5.4.2 Tolerance given as a velocity

The input of this recursive routine is an interval with datapoints.
It starts by calculating the transformation quaternion r from start to end. This quaternion is then expressed as a vector of three Euler angles, and dividing each one of these by the difference in time beetween start and end, give a vector of global velocities ω_{global}. In the same way the algorithm now computes the vector of local velocities ω_{loc} beetween two consecutive points on the inner interval. Now the tolerance criteria

$$\omega_{global} - \mathrm{TOL} < \omega_{local} < \omega_{global} + \mathrm{TOL} \tag{55}$$

is checked for each of the three velocity components. If it is violated in one or more of the components the second point is written to the output, and the whole procedure is repeated for the two subintervals $[\mathrm{start}, p_2]$ and $[p_2, \mathrm{end}]$.

5.4.3 Tolerance measured in angles

The procedure will on a given interval do the following:
Calculate angular velocity ω_{global} beetween start and end point as the velocity alternative. Use a linear approximation[10] $\tilde{\Phi}_i = \Phi_{start} + \omega t$ and calculate the difference beetween sampled value and approximated value for each compoment. Given as vectors this becomes

$$\Delta\Phi = \Phi_i - (\Phi_{start} + \omega t) \tag{56}$$

Here Φ_i and Φ_{start} is the Euler Angles at datapoint i and the startpoint respectively. Calculating this differance for all points on the interval. Select the point with largest difference in norm i.e.

$$n_{max} = \max_n \left[|\Delta\Phi|_n = \sqrt{\sum_{i=1}^{3} \delta\phi_i^2} \right] \tag{57}$$

[10] A *linear* approximation will not yield any angular acceleration

If the maxnorm is larger than a given tolerance, divide the input interval in two at n_{max} and call the routine recursively for the two subintervals $[start, n_{max}]$ and $[n_{max}, end]$. Output the datapoint n_{max}. The C++ code for both methods can be found in appendix F.2

5.5 Joining the 3D- and orientation algorithms

The final task is to melt the positional reduction algorithm (3D-algorithm) and the best orientation reduction algorithm into a complete reduction algorithm which considers both position and orientation of the points in the set. Based on the results in sections 6.2 and 6.3 the best orientaion reduction algorithm is EulerAngle. Hence this will be the one that is implemented together with the 3D procedure in the joining algorithm.

5.5.1 Summary of the Joining Algorithm

Since the algorithm consist of mathematics and procedures that have been presented throughout this text, only the main parts in the flow of the algorithm is presented here.

1. When a dataset is read, the 3D-algorithm run through the dataset, defining a polynomial for each coordinate on an interval representing indeces in the set.

2. For each of the intervals EulerAngle is run to check whether the orientation satisfies its tolerance criteria or not. If it is

   accepted continue to next interval.

   rejected divide the interval at the point that exceeds the given tolerance. The two (or more) subintervals still have the same polynomial defining the positions as a function of indices.

3. The output of the algorithm, will be a set of points. Each one with three polynomials $(x, y, z)(t)$ defined on an interval, giving the tool center point's position, a quaternion representing the direction of the tool at the interval's first point, the interval of indeces $[T_0, T]$, and the time when the interval's first point was sampled.

The algorithm can be expressed mathematically as the function F with the mapping

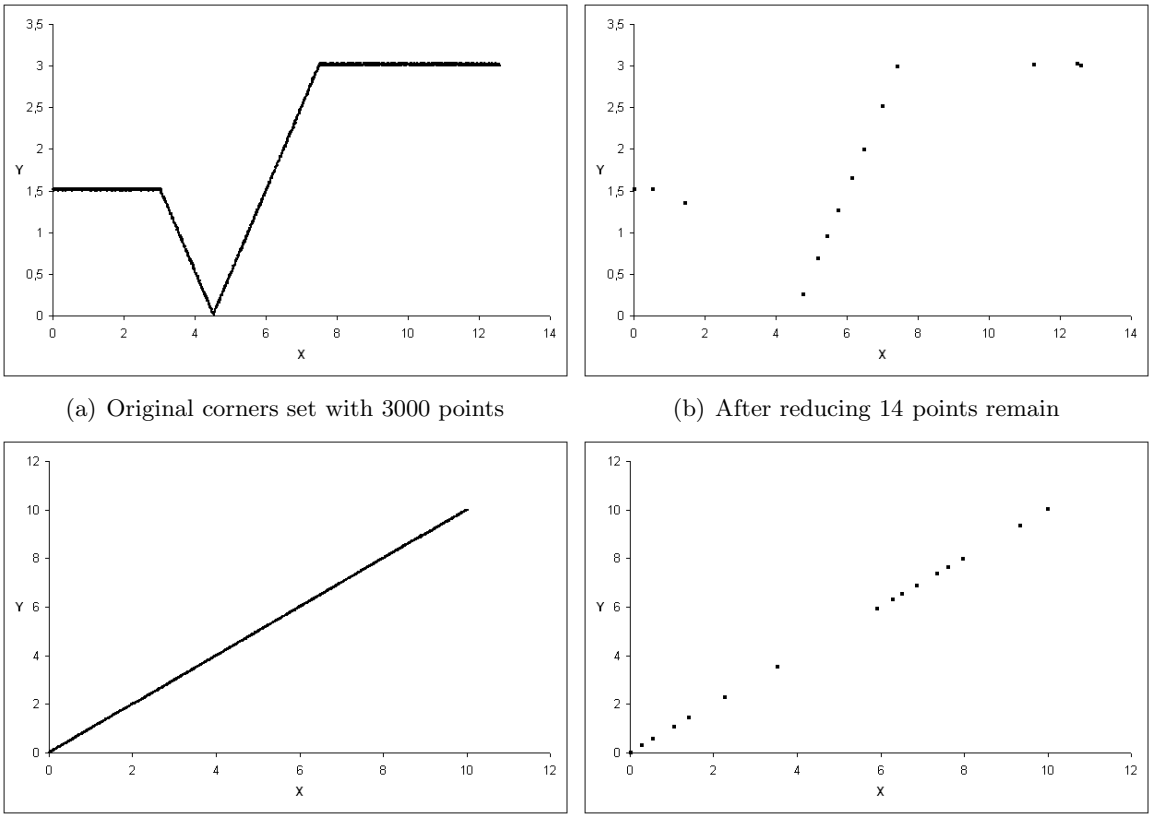
$$F : \{(x, y, z, q[4], t)_i | i = 1 \dots N\} \longmapsto \{((x, y, z)(t), q[4], [T_0, T], t)_i | i = 1 \dots n\} \quad \text{where } n \ll N \quad (58)$$

6 Results

In this section results from running the filtration algorithms in section 5 are presented.

6.1 3D-Algorithm

The 3D-Algorithm were run and tested on the sets presented in section 4.3.1 with different tolerances. Figure 19 show the x-y-plane of the sets after running the algorithm with tolerance 1. For comparison the original is shown. The figures corresponding to the results of the other tolerances are given in appendix E in figure 28 and 29.



(a) Original corners set with 3000 points

(b) After reducing 14 points remain

(c) Original line set with 3000 points

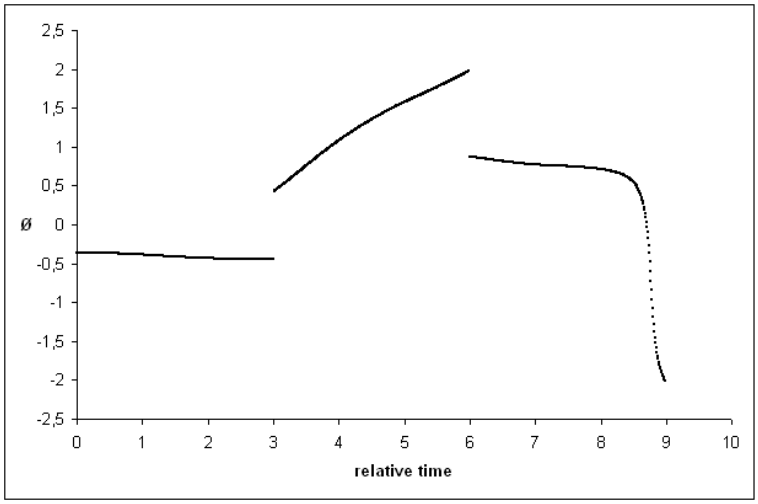
(d) After reducing 16 points remain

Figure 19: Plots of the results from testing the 3D algorithm with tolerance 1.

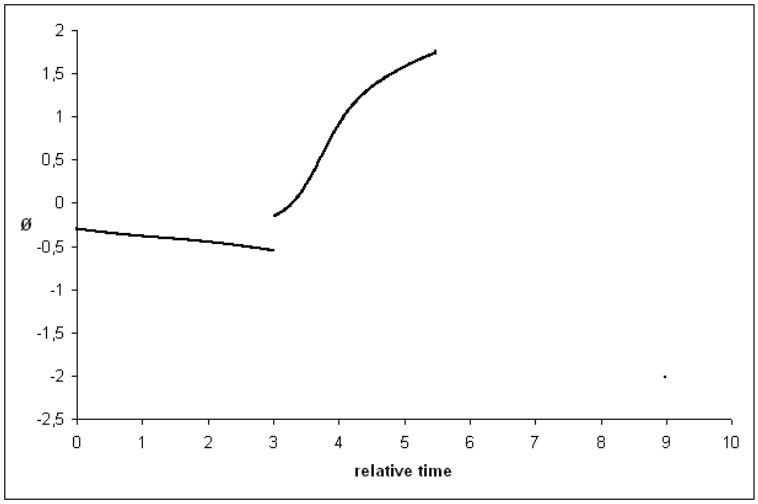
There is one thing about these results that needs to be discussed. The fact that for high tolerances the algorithm seems to break down. If it was flawless the results would be that if the tolerance was set high enough, then the whole set would be reduced to the start- and end-point. The numerical results showed that this was not so for the sets tested and it can also be seen on the figures in the above mentioned appendix. After reducing the set at tolerance 5 the corners set have more points than for tolerance 1 and lower. It was suspected that this error was caused by the numerics in the core of Newton's method. See section 8 for remarks on this matter.

6.2 Orientation

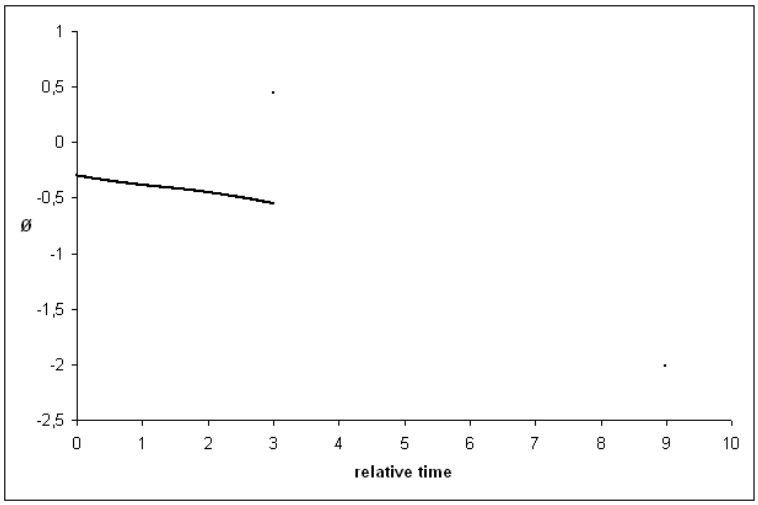
The orintation-algorithm or quaternion-reduction-algorithm were run on the set presented in section 4.3.2. The plots of the results are shown in figure 20.



(a) Original



(b) Tolerance 90



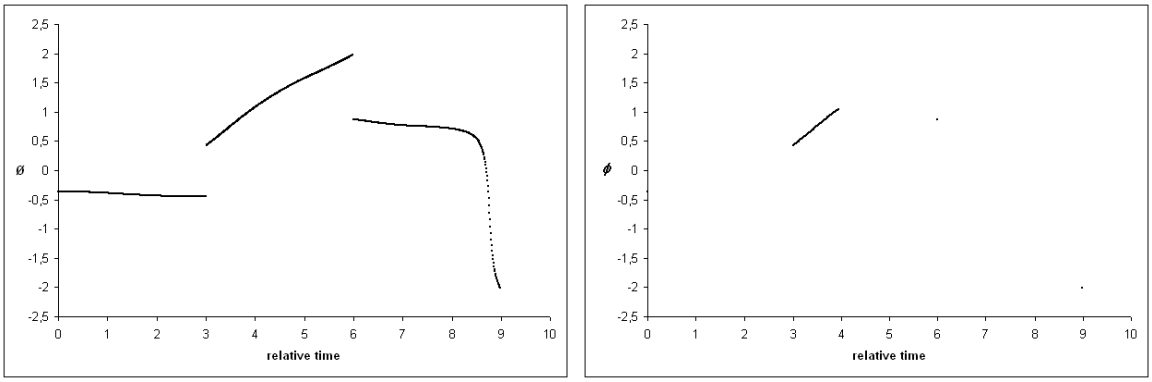
(c) Tolerance 105

Figure 20: The $\phi(t)$-plots after reducing the angles set with the quaternion reduction algorithm.

From these results it is clear that the algorithm does not work the way it was intended. For the angles set one would hope that the algorithm reduced the set to the start and end point of each of the three curved lines in the ϕ-plot, and perhaps a middlepoint on the last curve. This way the curvature of the set is preserved. The reason for the algorithms failure is that distance beetween quaternions could not be thought of as a distance in the usual sense of the word, and is meaningless to use during calculations that should be considering the Euler angles instead. Therefore another approach has to be considered. The Euler reduction in section 5.4 does that.

6.3 Euler Reduction

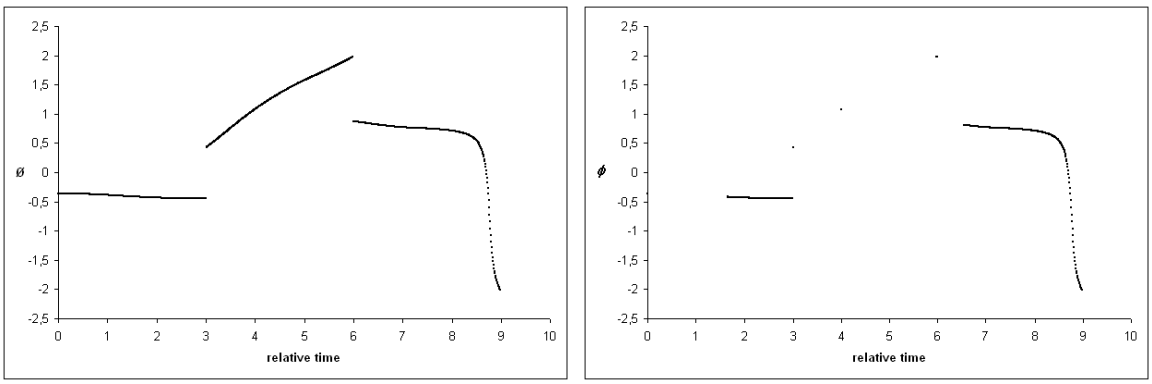
Like the quaternion reduction algorithm, both algorithms presented in section 5.4 were run on the angles dataset.
For the EulerVelocity algorithm the plot of the angles set after reduction with tolerance 0.5 is shown in figure 21. The plots for the other tolerances could be found in appendix E in figure 30.



(a) Original (b) Tolerance 0.5

Figure 21: The $\phi(t)$-plot of the angles set after reducing with the EulerVelocity Algorithm.

A similar plot to the previous algorithm show the angles set after reduction with with EulerAngle at tolerance 0.5 in figure 22. The plots for the other tolerances is given in figure 31 in appendix E.



(a) Original (b) Tolerance 0.5

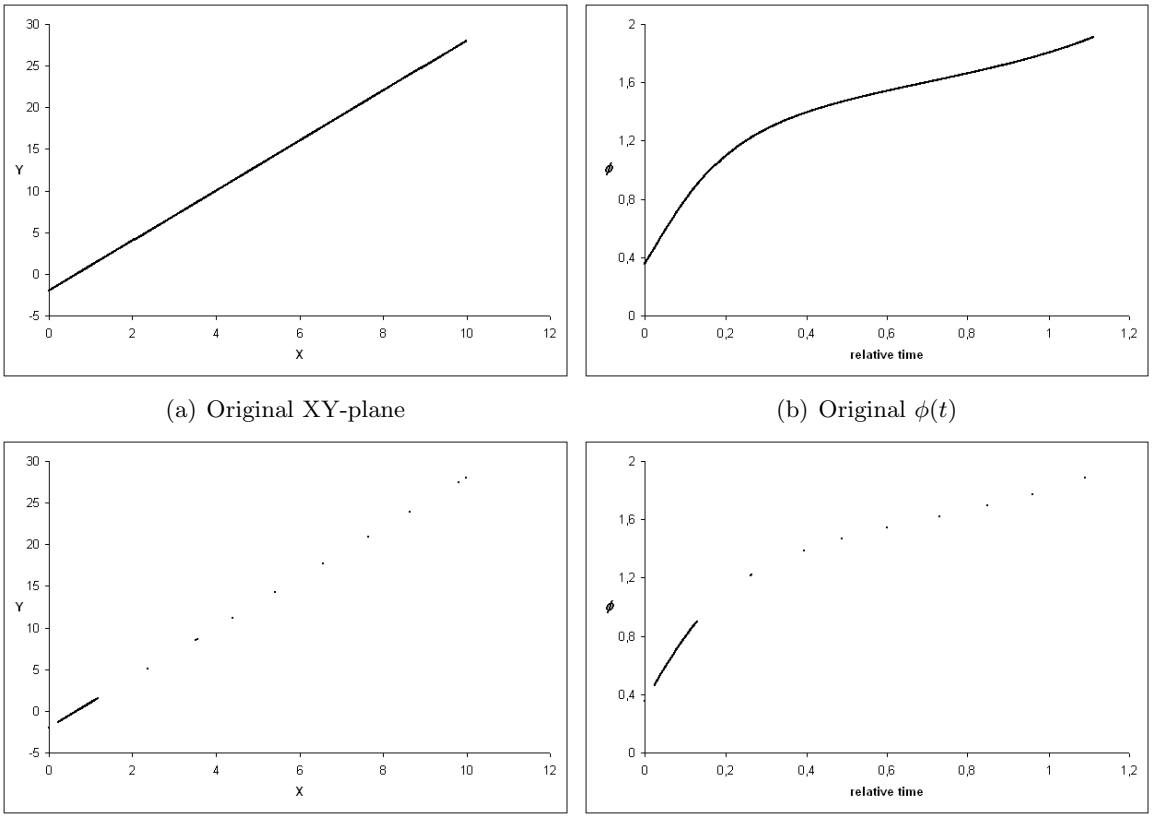
Figure 22: The $\phi(t)$-plot of the angles set after reducing with the EulerAngle Algorithm.

6.3.1 Summary of the Euler Reduction results

The numerical results for the two alternatives show an equal pattern, even though they measure the tolerance in very different ways. The differences beetween the two are best shown in the plots at various tolerances. By comparing the plots it is clear that the EulerAngle is the best algorithm. This is because it preserves the structure of the set. The EulerVelocity has similarities with the quaternion reduction, which was dismissed due to its problem with the preservation of the structure.
Based on these results the EulerAngle is chosen for orientational part of the final algorithm, the one that join position and orientation.

6.4 The joining algorithm

Here the results after running the algorithm presented in section 5.5 are shown. The results for the two datasets are presented in one figure for each of the two. The plots from results that are not shown here, can be found in appendix E, in figures 32 through 39. Since this is the final algorithm and sums up most of this work, the discussion of the results is left to section 8.



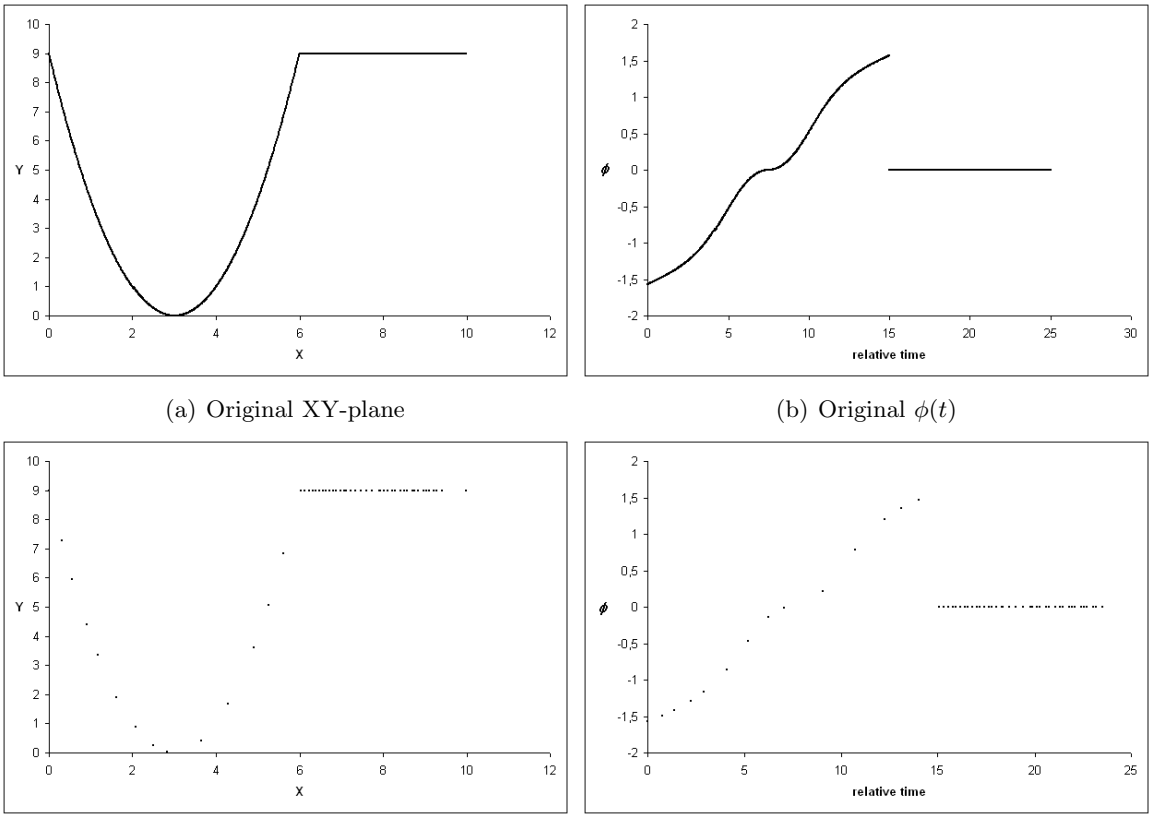
(a) Original XY-plane

(b) Original $\phi(t)$

(c) Filtered XY-plane

(d) Filtered $\phi(t)$

Figure 23: The figure show the XY-plane and the $\phi(t)$ for the *simple* dataset before and after filtering with tolerance 0.1 for both position and orientation.



(a) Original XY-plane

(b) Original $\phi(t)$

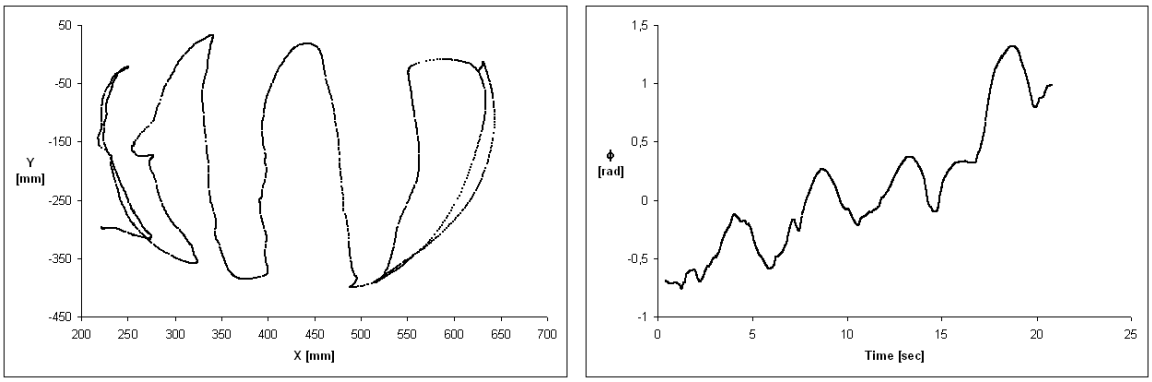
(c) Filtered XY-plane

(d) Filtered $\phi(t)$

Figure 24: The figure show the XY-plane and the $\phi(t)$ for the *complex* dataset before and after filtering with position tolerance 0.01 and orientation tolerance 0.1.

7 Test on a real set

The datasets used during the development of the various algorithms was generated as dimensionless data. This gave some indications on what worked and what did not, but for instance the numbers for different tolerances gave little meaning. To test the final product in the real world a physical test was needed.



(a) XY-plane (b) $\phi(t)$

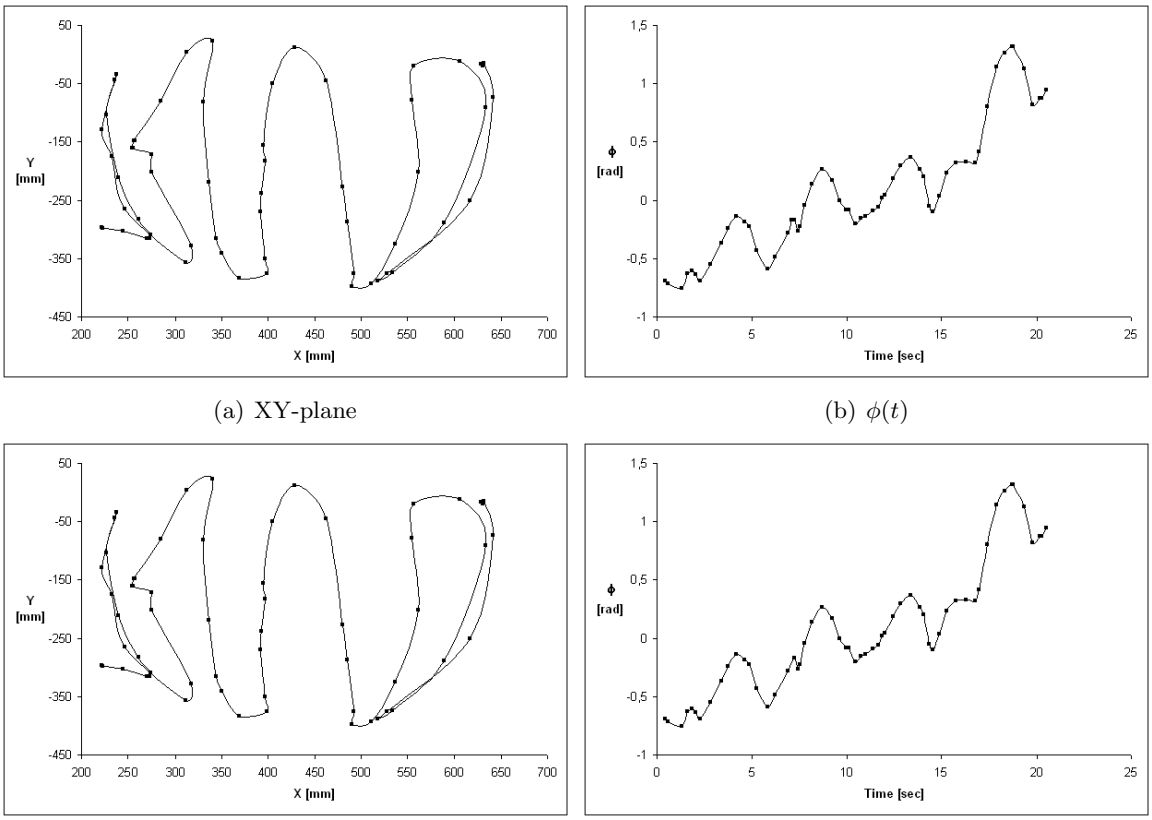
Figure 25: The original set showing the XY-plane and $\phi(t)$ of a bike helmet.

Figure 25 show the physical set used for the testing. It is a tracking of a motion that follow the "valleys" in a bike helmet. Originally the set contained 2100 points that was tracked at 103.15 Hz.

For the reduction 4 combinations of typical tolerances were used. The positional tolerance was given as 5 and 1 millimiters, and the orientational tolerance 5 and 10 degrees[11]. Figures 26 and 27 show the results of the reduction.

The results of the physical test show that the algorithm works well. The set was reduced by more than 97% at both tolerances. At 1 millimeter the set is slightly less reduced than at 5 millimeters, but that has the consequence that the original curve is preserved a little better.

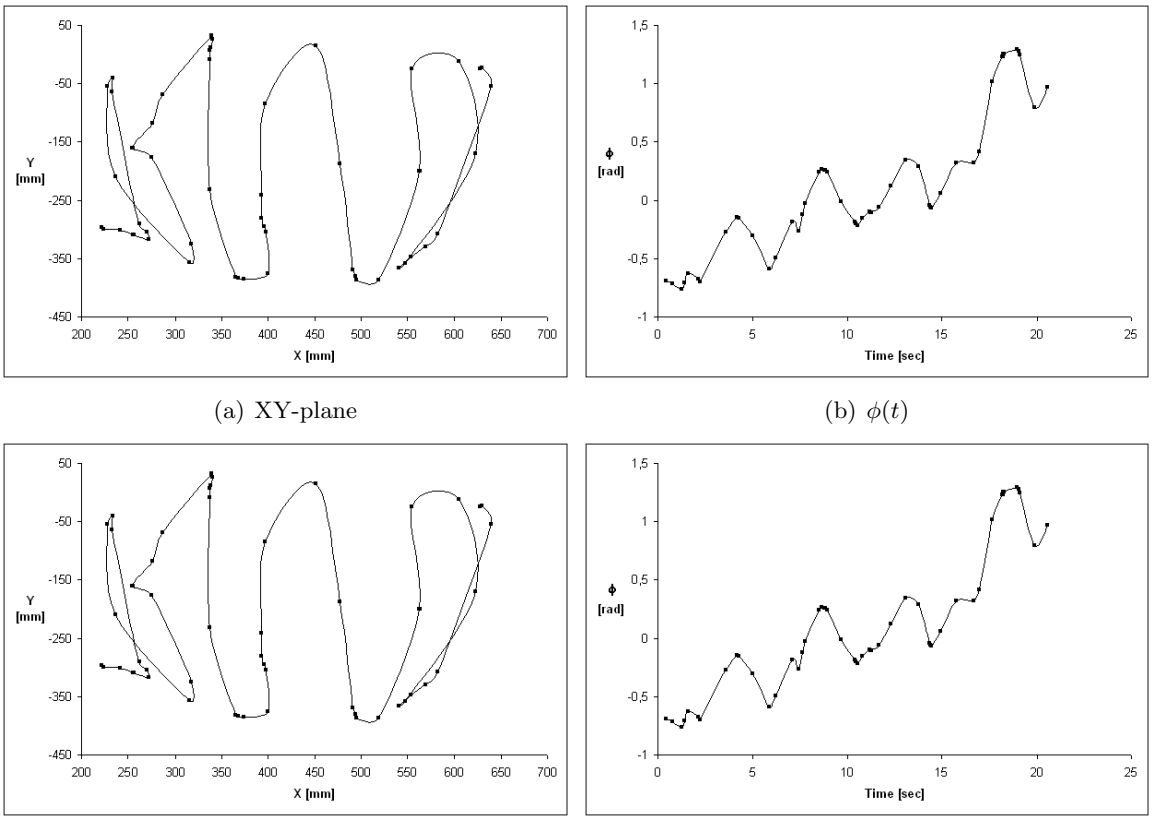
[11]The algorithm operates in radians so tolerances must be converted into radians for internal calculations.



(a) XY-plane (b) $\phi(t)$

(c) XY-plane (d) $\phi(t)$

Figure 26: The results after reducing with positional tolerance 1 millimeter. The upper half show orientational tolerance at 5 degrees and the bottom at 10 degrees. The lines beetween point are added to indicate which points follows another and to make it easier to recognize the structure.



(a) XY-plane

(b) $\phi(t)$

(c) XY-plane

(d) $\phi(t)$

Figure 27: The results after reducing with positional tolerance 5 millimeter. The upper half show orientational tolerance at 5 degrees and the bottom at 10 degrees. Again, the lines beetween point are added to indicate which points follows another and to make it easier to recognize the structure.

8 Discussion

8.1 Remarks on the 3D-Algorithm

In an attempt to deal with the error in this algorithm, several improvements was tried. One of these was Horner's method[12] which improves the speed of the algorithm by removing the necessity to compute powers of the argument. Another improvement was to check if the root that was found lie in the interval where the polynomial was defined. If not, the root would be divided out and the process could be repeated for the new algebraic equation with smaller degree. The operation of dividing out the root is known as *deflation* or *syntetic division* (see e.g.[6]). For better stability, Newton's method should be replaced by Laguerre's iteration method (see appendix B). This is not implemented or tested here though. A completely different approach to the root problem would be to use the theory of Sturm presented in section 3.6 and avoid the calculations of each root.

The results of the improvements of Newton's method was not promising. None of the roots seem to lie in the interval and when the root check was inserted into the original implementation of Newton's method, the results became the same for that one too. Doing the whole procedure by hand for a very small set though, works fine so there is nothing wrong with the mathematical assumptions this procedures is based upon. The programming on the other hand needs further investigation before it is implemented in a robust software solution. In this software solution it would probably be favourable to also have the opportunity to chose the degree of the aproximating polynomial. This would change the dimension of the matrix used to calculate the approximating polynomial but not the procedure itself. A single degree polynomial (2 by 2 matrix) would be better for aproximating a straight line rather than the third degree polynomial used in this text.

8.2 The orientation and joining algorithms

The purpose of the orientation algorithms was to decrease the number of points in the data, without damaging the structure of the curve showing the Euler angle ϕ as a function of time. The results of the testing of these algorithms showed that the EulerAngle algorithm was the best choice. Only one of the three Euler angle giving a complete orientation in space, are shown in the results. What happened to the other angles was not investigated, but they are being considererd in the algorithm. It is therefore no reason to suspect otherwise than if another Euler angle was selected for the plots, the results would be equal. In the tolerance routine for EulerAngle a linear aproximation was used and as it was mentioned above this is best for aproximating straight lines. One interesting possibility also for the orientation is to have polynomials expressing each Euler angle as a function of time, and estimate this using the same procedure as for the position. Clearly the problem of choosing the proper degrees of the aproximating polynomials has to be dealt with again.

Despite the problems with the 3D algorithm the results from the joining algorithm are promising. For the positional tolerance equal 0.1, which is about 1% of the maximum value in the y-coordinate, and the orientational tolerance equal 0.1, the *simple* and *complex* dataset were filtered with aproximately 90 and 98 percents respectively. Again we see that the algorithm reduces a curved set, slightly better than the straight line, where the degree of the polynomial is the main reason.

[12] $p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 = ((a_n z + a_{n-1})z + \cdots)z + a_0$

9 Conclusion

The main purpose of this work was to survey the mathematics in connection with the point reduction in IRP. Since most of the position reduction was treated in [8], this text has focused on the orientation and the joining of the two parts. The data from a typical sensor represents orientation in quaternions, so a method using these was first investigated. As discussed earlier in the text the results did show that the problem instead had to be handled with Euler angles. This had a lot to do with how the tolerance criteria were set. This was also the theme when the two approaches to the Euler reduction were treated. The algorithm that showed the best results from the testing was *EulerAngle*, and hence it was selected for the orientation part of the final algorithm.

The testing on a physical set showed good results. Realistic tolerances on realistic data seemed to remove the problems discussed in the previous section. For physical data it is highly satisfying when it is possible to reduce a set with 97% without loss of vital information.

References

[1] Gaudia Andor. Simulations of industrial robots. Diploma thesis, Budapest University of Technology and Economics, 2004.

[2] Reskó Barna. Optical motion tracking for industrial robot programming. Diploma thesis, Budapest University of Technology and Economics, 2004.

[3] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Brooks/Cole, seventh edition, 2001.

[4] Camelot. *Introduction to Robot Simulation*. `http://www.camelot.dk/english/introsimu.htm`, 2004.

[5] Thomas H. Cormen ...[et al]. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[6] Germund Dahlquist and Åke Björck. *Numerical Methods*. Prentice-Hall, 1974.

[7] John B. Fraleigh. *A First Course In Abstract Algebra*. Addison Wesley, seventh edition, 2003.

[8] Tore Martin Madsø Hauan. Filtering and point reduction in intuitive robot programming. *Unpublished*, 2005.

[9] A.S. Householder. *The Numerical Threatment of a Single Nonlinear Equation*. McGraw-Hill, 1970.

[10] Tore Håvie. *Numeriske beregninger*. Institute of mathematical sciences NTNU, 1996.

[11] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Springer-Verlag, 2005.

[12] Gilbert Strang. *Linear Algebra and its Applications*. International Thomson Publishing, third edition, 1986.

[13] Trygve Thomessen. Robot control system for safe and rapid programming of grinding applications. *Industrial Robot*, 27 and 28, 2001.

[14] J. N. Zemel W. Göpel, J. Hesse. *Sensors : a comprehensive survey*, volume 7. VCH Verlagsgesellschaft, 1989.

[15] J. N. Zemel W. Göpel, J. Hesse. *Sensors : a comprehensive survey*, volume 5. VCH Verlagsgesellschaft, 1989.

[16] Eric W. Weisstein. *Quaternion*. From *Mathworld* - A Wolfram Web Resource `http://mathworld.wolfram.com/Quaternion.html`, 2006.

[17] Are Willersrud. Restoration of curves disturbed by outliers and noise. Diploma thesis, Norwegian Institute of Technology, 1993.

A Quicksort Algorithm

Quicksort is a sorting algorithm whose worst-case running time is $\Theta(n^2)$ on an input array of n numbers. Despite this it is the best practical choice for sorting due to it's average running time [5]. In this context it is a clear advantage that qsort is an implemented method in the C/C++ standard library.
Quicksort sorts the subarray $A[p \dots r]$ by the following divide-and-conquer procedure:

Divide Rearrange the array $A[p \dots r]$ into two subarrays $A[p \dots q-1]$ and $A[q+1 \dots r]$ such that $\{a \leq A[q]$ — for all $a \in A[p \dots q-1]\}$ and $\{b \geq A[q]$ — for all $b \in A[q+1 \dots r]\}$.

Conquer Sort the two subarrays $A[p \dots q-1]$ and $A[q+1 \dots r]$ by recursive calls to quicksort.

Since the subarrays are sorted in place the entire array $A[p \dots r]$ is sorted.

B Laguerre's iteration method

This method follow the iteration scheme

$$z_{k+1} = z_k - \frac{np(z_k)}{p'(z_k) \pm \sqrt{H(z_k)}} \tag{59}$$

where

$$H(z) = (n-1)[(n-1)(p'(z))^2 - np(z)p''(z)], \tag{60}$$

and n is degree of $p(z)$. The sign in the denominator is chosen such that $|z_{k+1} - z_k|$ is as small as possible. It's main advantages over Newton's method is the cubic convergence for simple roots, and that the initial estimate does not have to be complex when it is to find a complex root. If the algebraic equation only has real roots the method is convergent for any choice of initial estimation.

C Look-Ahead Filter

The intention of this filter is to remove outliers from the datasets. As the name implies this is done by looking ahead in the dataset. For each point in the set, one looks at a selected number of following points, and determines the distance to each one of these. Based on this distances, one obtains information on one or more local minima. If the local minima is in the second half of the fixed interval, the point at hand are accepted and the filter algorithm proceeds to the next. If, on the other hand, the minima is in the first half, one determine the point where the lowest minima is located. For the points following the selected one, one accept them if their distance is smaller than the one to the minimum. Otherwise they are considered as outliers, and therefore discarded. The minimum point will be the next starting point. For more details and the C++ code, see [17].

D Denavit Hartenberg Convention

The Objective

The objective of the method is to calculate $\mathbf{T}_n^o = \begin{bmatrix} \mathbf{R}_n^o & \mathbf{p}_n^o \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}$. Thus, the goal is to establish the end effector position and orientation as a function of the link variables and parameters.

Assumptions

Assume an open kinematic chained mechanism with $n+1$ links numbered from 0 (the base) to n. The connecting joints are numbered from 1 to n.

The convention

1. Assign a joint number to each joint from 1 to n. Joint 1 connects link 0 with link 1. The variable of joint i is q_i, and is either revolute or prismatic.
2. Find and number the joint axis consecutively from 0 to $n-1$. Joint 0 connects the base with link 1.
3. Choose axis z_i along the axis of joint $i+1$, i.e. the z_0 axis points along joint 1. If q_i is revolute, then q_i rotate around the axis z_{i-1}, and if q_i is prismatic, then q_i translate along the axis z_{i-1}.
4. Locate the origin o_i where the common normal to z_i and z_{i-1} intersects z_i.
   (a) If z_i intersects z_{i-1} locate o_i at the intersection, x_i is the normal to the z_{i-1}, z_i plane.
   (b) Else they do not intersect, i.e. they are parallel or coplanar. Locate o_i at joint i. The x_i axis is along the common normal between z_{i-1} and z_i and in the direction from frame $i-1$ to frame i.
5. Establish the end-effector frame; frame n.
   (a) If joint n is revolute, then align z_n with z_{n-1}. Locate o_n in the appropriate center of the end-effector. The x_i axis is along the common normal between z_{n-1} and z_n and in the direction from frame $n-1$ to frame n.
   (b) Else, joint n is prismatic. Then z_n may be chosen arbitrarily, but it is convenient to assign z_n parallel to z_{n-1}. The x_n axis may then be chosen parallel to x_{n-1}
6. Create a table of the Denavit-Hartenberg link parameters $a_i, d_i, \alpha_i, \theta_i$ as follows:

   a_i The distance along the x_i axis from o_i to the intersection between the x_i and the z_{i-1} axis.

   d_i The distance along z_{i-1} from o_{i-1} to the intersection of the x_i and the z_{i-1} axis. d_i is the link variable q_i when joint i is prismatic.

   α_i The angle between z_{i-1} and z_i measured about x_i, see figure D.

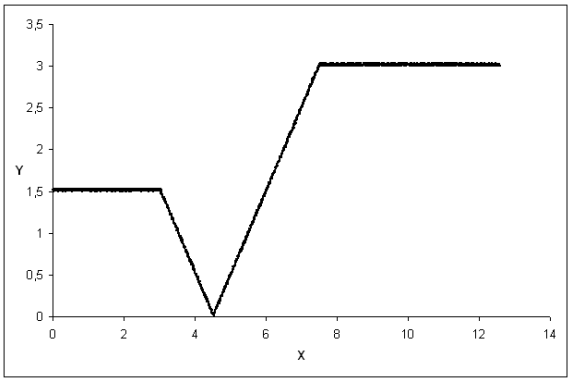
   θ The angle between x_{i-1} and x_i measured about z_{i-1}, see figure D. θ is the link variable q_i if joint i is revolute.
7. Form the homogenous transformation matrix $\mathbf{A}_i^{i-1}$ by substituting the D-H parameters into the following matrix:

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\alpha \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{61}$$

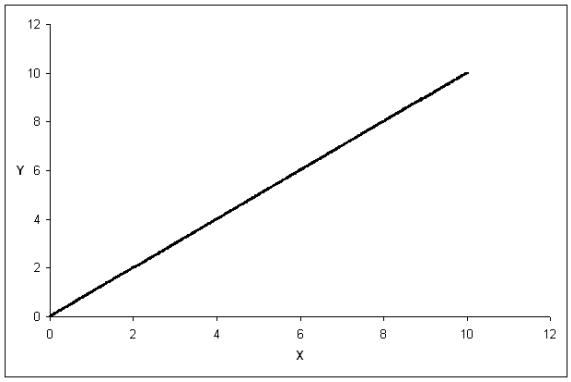
8. Form $\mathbf{T}_n^0 = \mathbf{A}_1^0\mathbf{A}_2^1\cdots\mathbf{A}_n^{n-1}$

E Plots

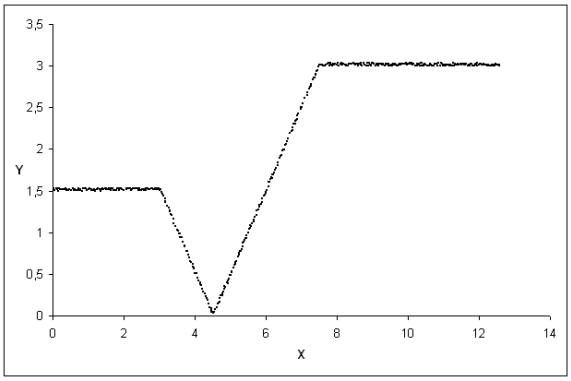
In this section all plots of the different datasets after filtering are shown. They follow the same structure as in the text.



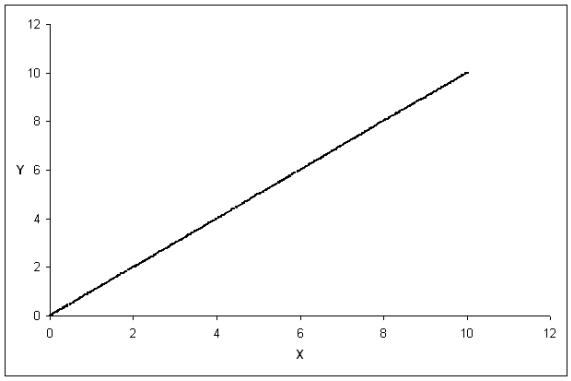
(a) Original corners set with 3000 points



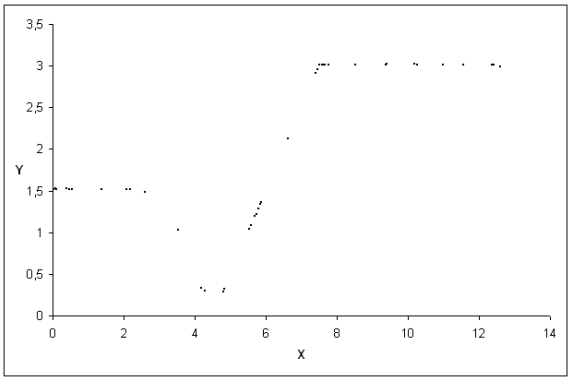
(b) Original line set with 3000 points



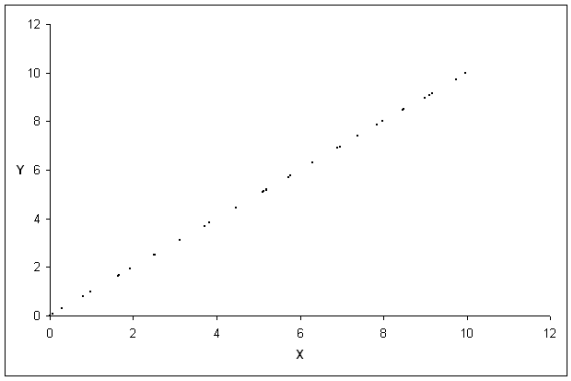
(c) Tolerance at 0.01 leave 500 points



(d) Tolerance at 0.01 leave 587 points

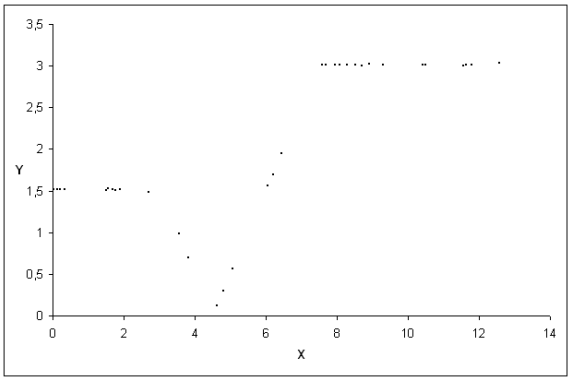


(e) Tolerance at 0.05 leave 41 points

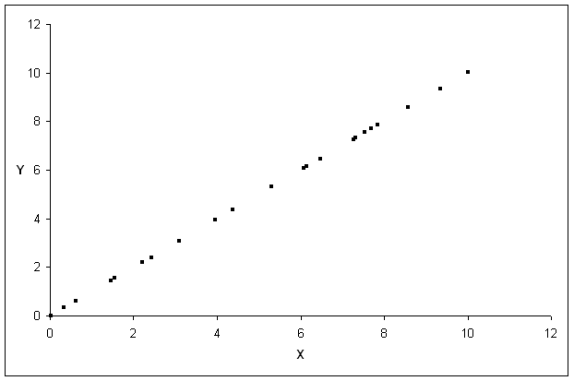


(f) Tolerance at 0.05 leave 32 points

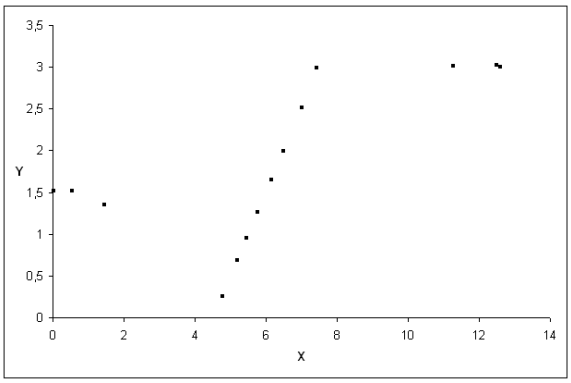
Figure 28: Plots of the original and results from testing the 3D algorithm with low tolerances



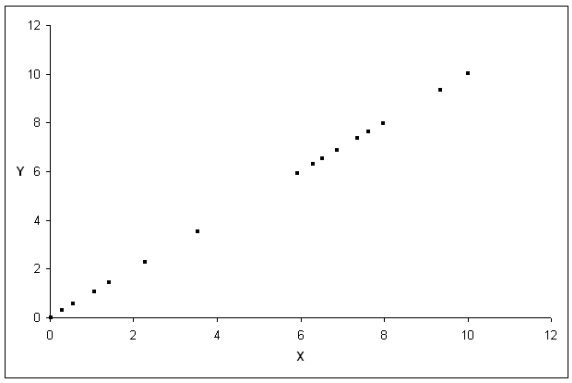
(a) Tolerance at 0.1 leave 37 points



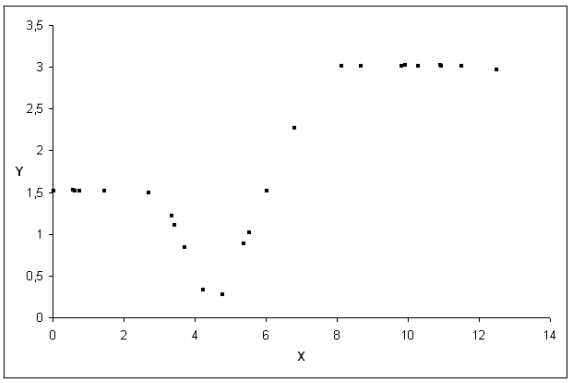
(b) Tolerance at 0.1 leave 21 points



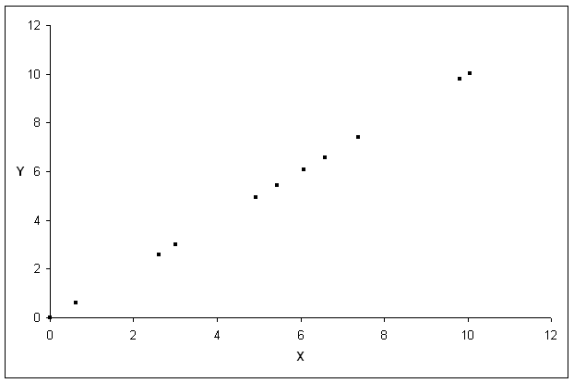
(c) Tolerance at 1 leave 14 points



(d) Tolerance at 1 leave 16 points

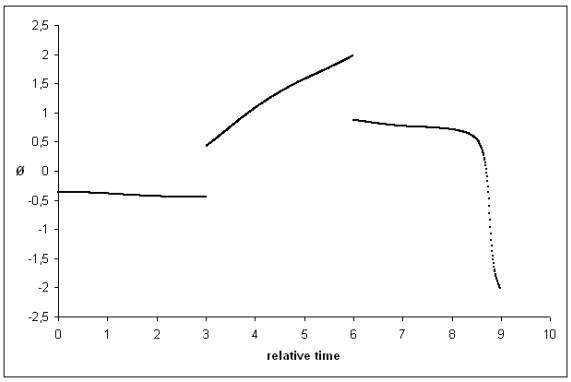


(e) Tolerance at 5 leave 24 points

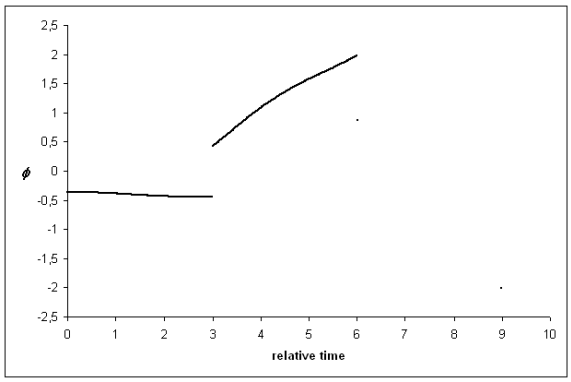


(f) Tolerance at 5 leave 11 points

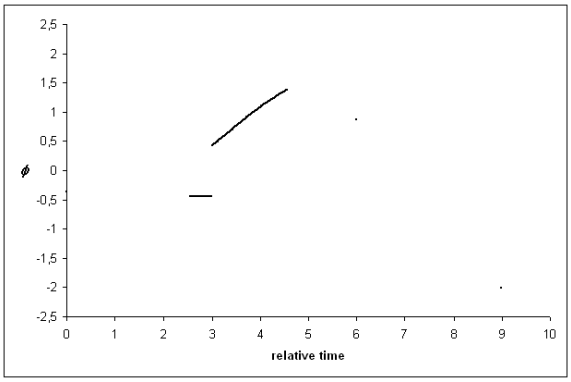
Figure 29: Plots of the results from testing the 3D algorithm with higher tolerances



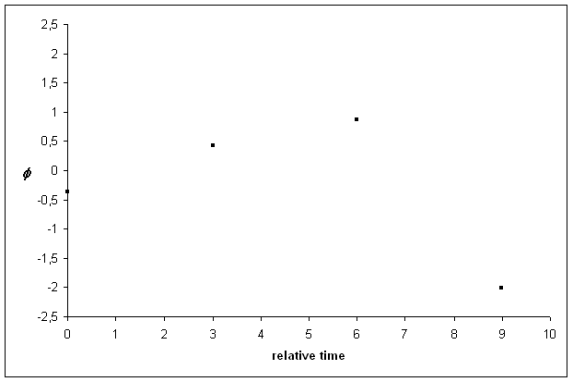
(a) Original with 1000 points



(b) Tolerance at 0.1 leave 336 points

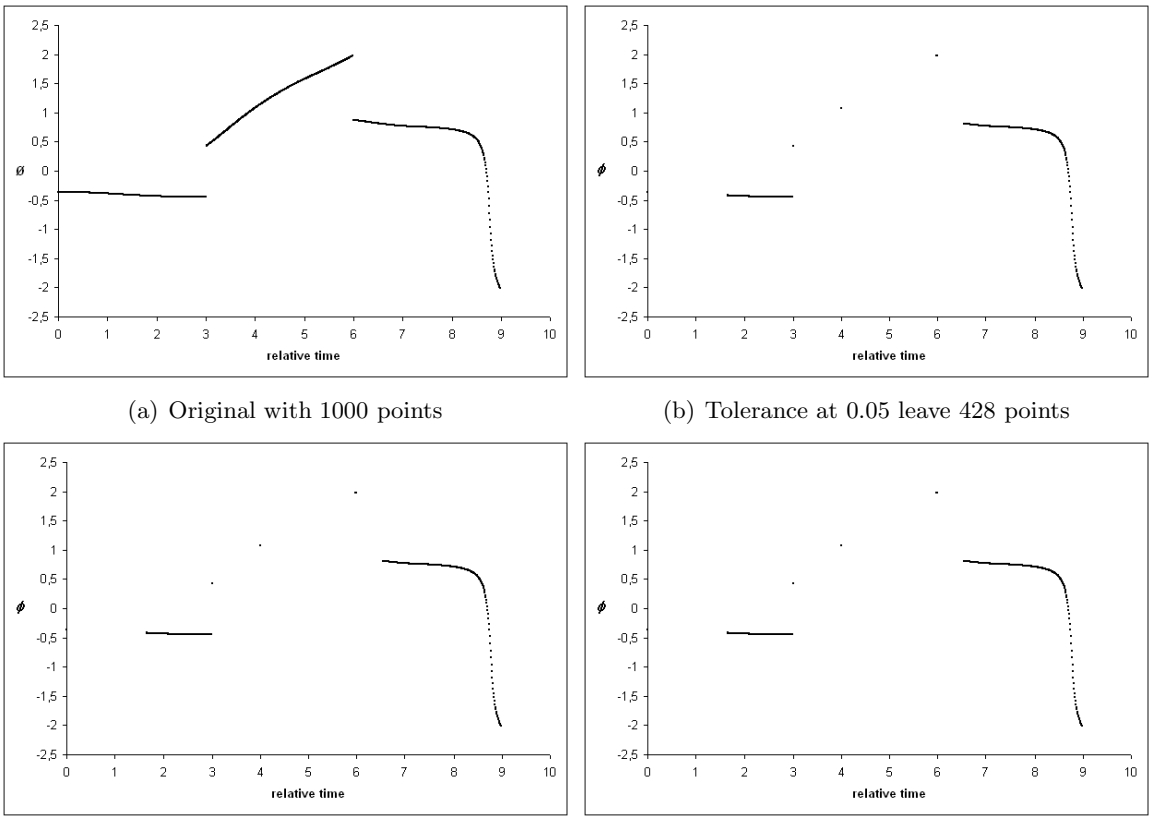


(c) Tolerance at 0.25 leave 115 points



(d) Tolerance at 0.75 leave 4 points

Figure 30: Plots of the angles set after reduction with the EulerVelocity algorithm.



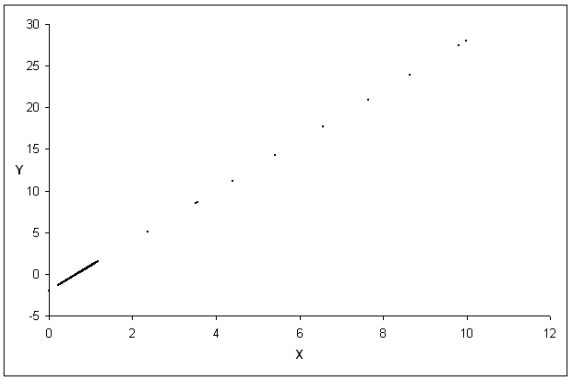
(a) Original with 1000 points

(b) Tolerance at 0.05 leave 428 points

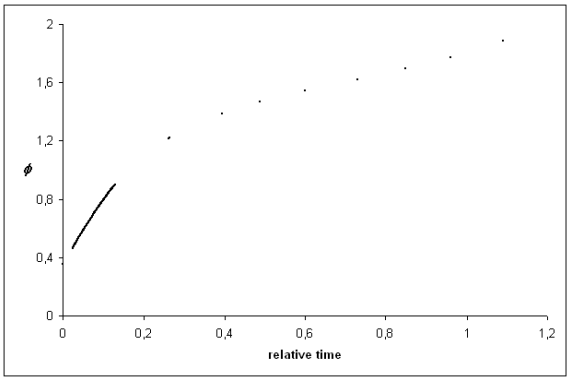
(c) Tolerance at 0.75 leave 88 points

(d) Tolerance at 0.8 leave 38 points

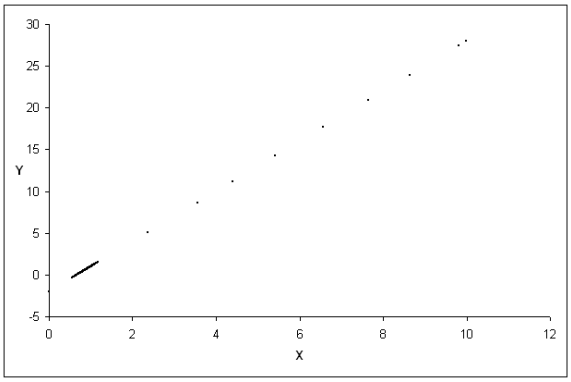
Figure 31: Plots of the angles set after reduction with the EulerAngle algorithm.



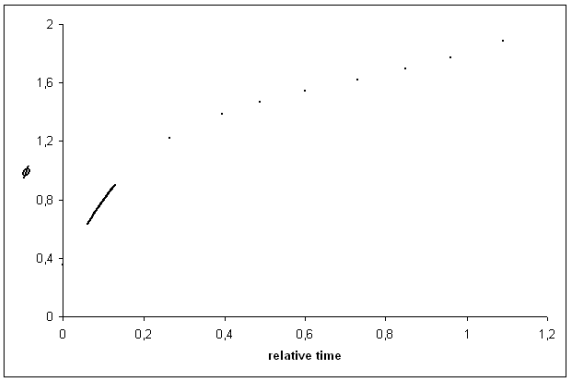
(a) XY-plane



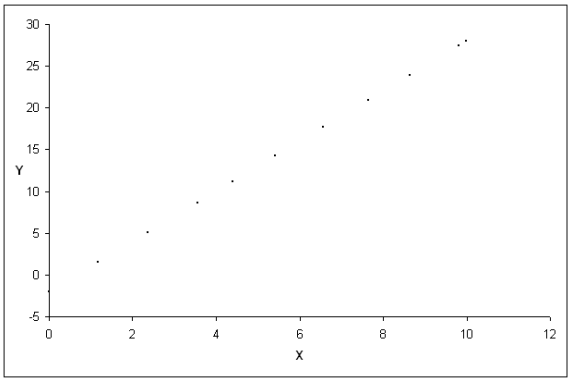
(b) $\phi(t)$ orientation tolerance 0.1



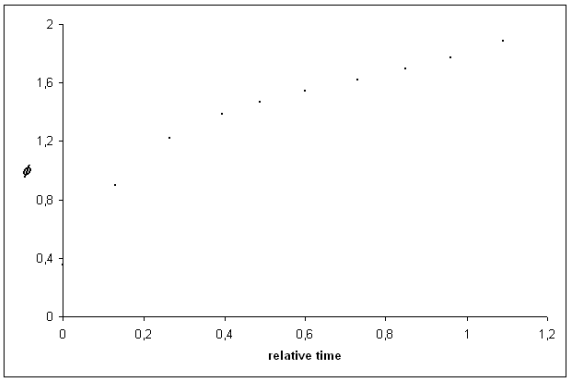
(c) XY-plane



(d) $\phi(t)$ orientation tolerance 0.25

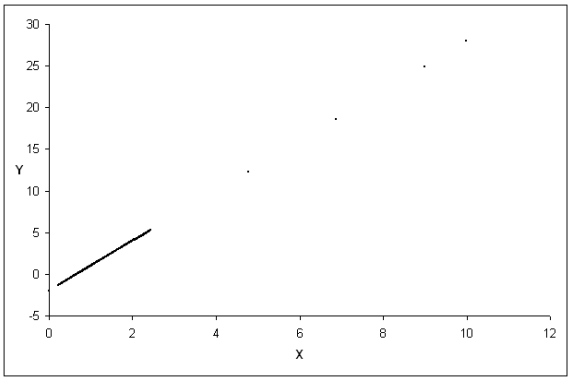


(e) XY-plane

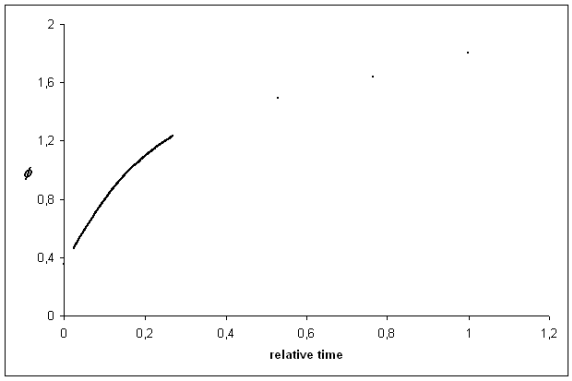


(f) $\phi(t)$ orientation tolerance 0.5

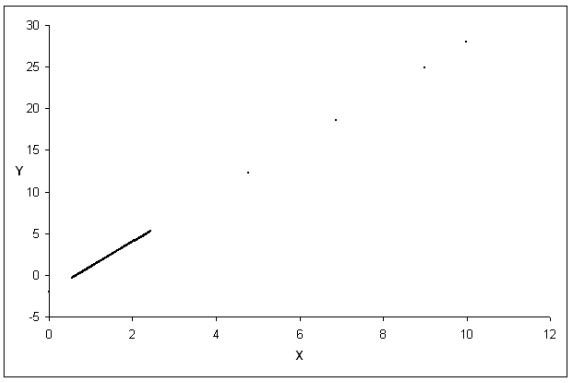
Figure 32: Plots after filtering the *simple* dataset with the joining algorithm. Position tolerance 0.1.



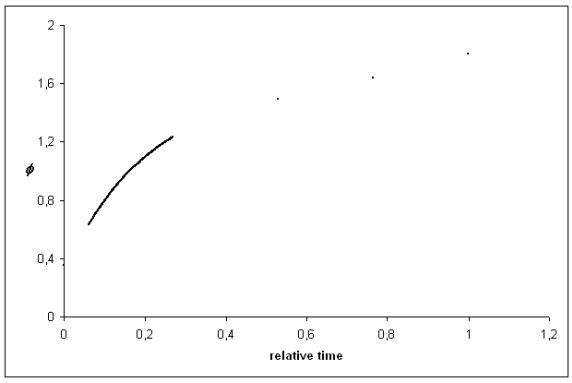
(a) XY-plane



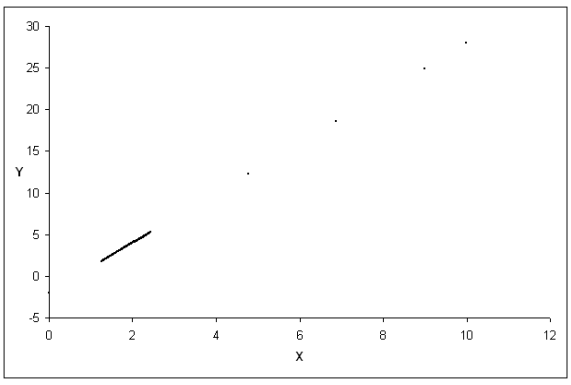
(b) $\phi(t)$ orientation tolerance 0.1



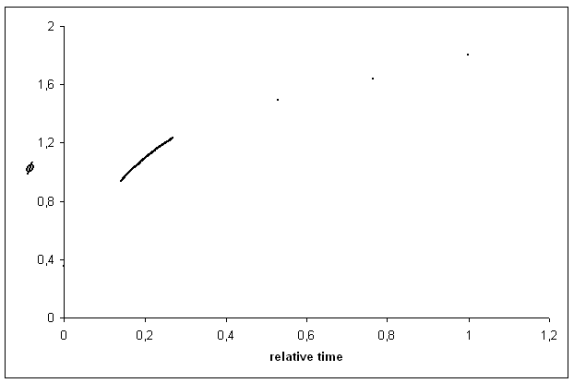
(c) XY-plane



(d) $\phi(t)$ orientation tolerance 0.25

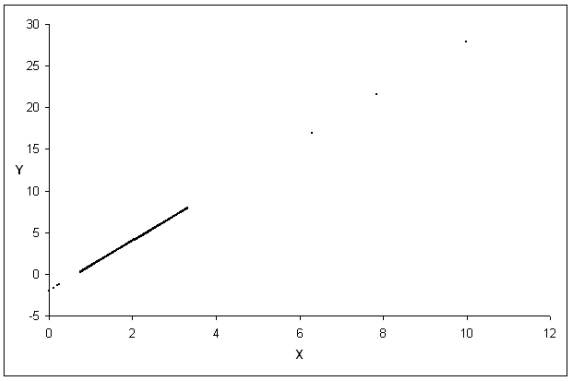


(e) XY-plane

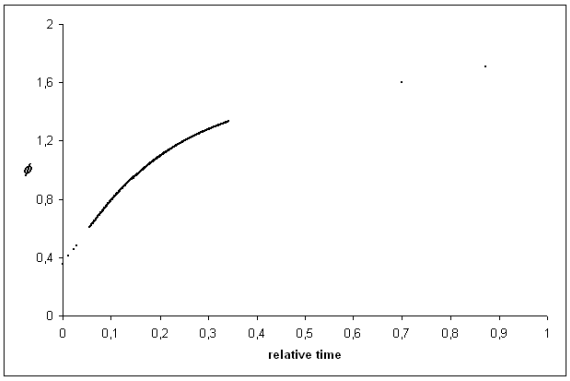


(f) $\phi(t)$ orientation tolerance 0.5

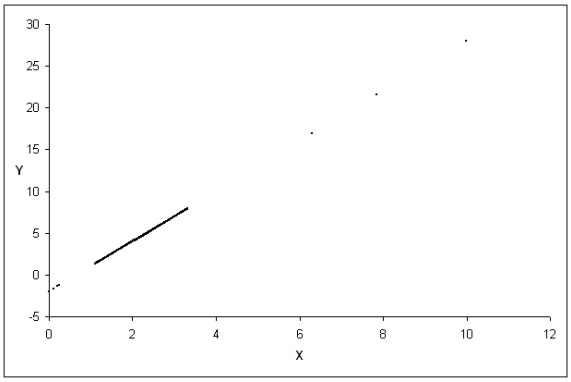
Figure 33: Plots after filtering the *simple* dataset with the joining algoorithm. Position tolerance 0.5.



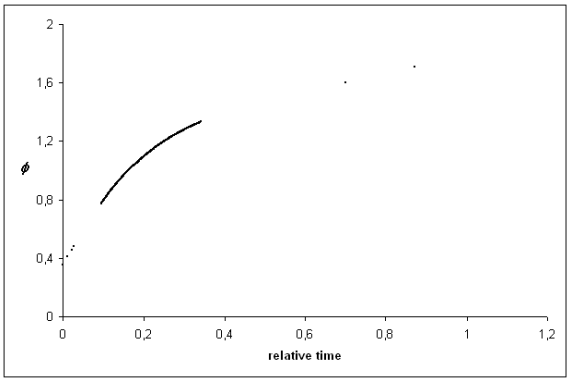
(a) XY-plane



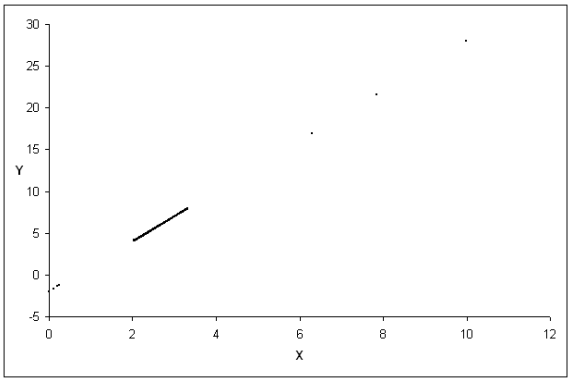
(b) $\phi(t)$ orientation tolerance 0.1



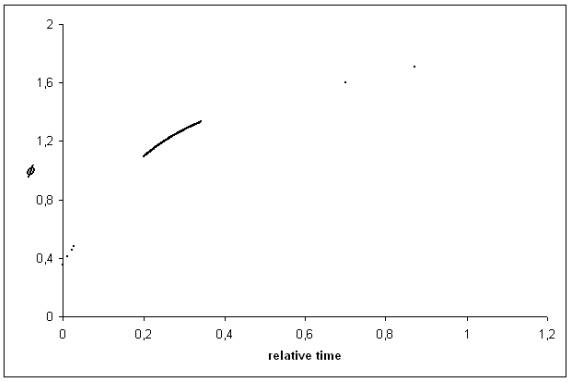
(c) XY-plane



(d) $\phi(t)$ orientation tolerance 0.25

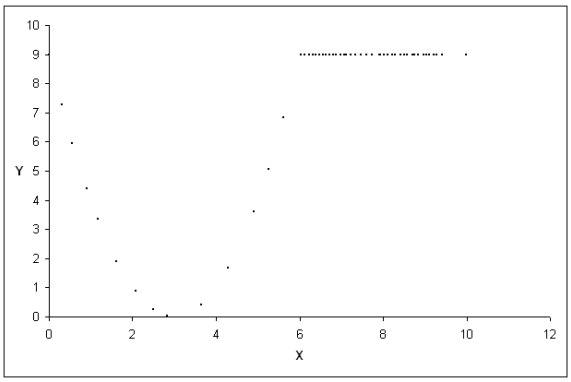


(e) XY-plane

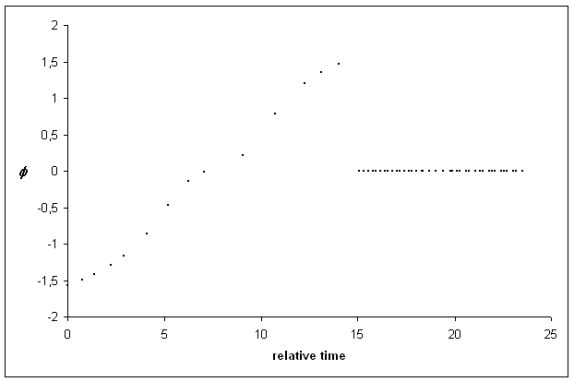


(f) $\phi(t)$ orientation tolerance 0.5

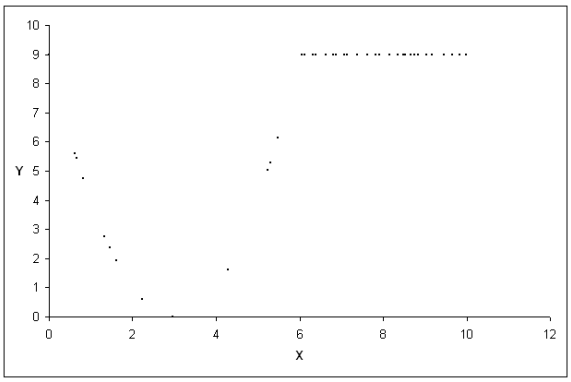
Figure 34: Plots after filtering the *simple* dataset with the joining algorithm. Position tolerance 1.



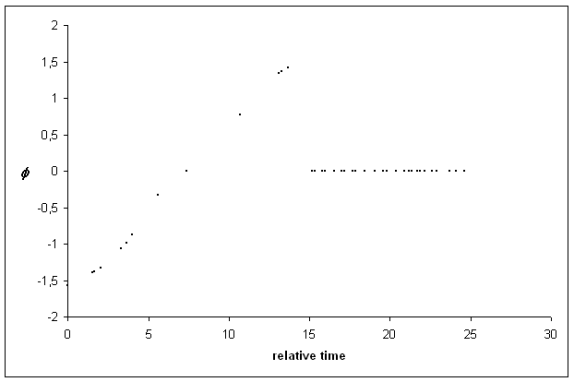
(a) XY-plane position tolerance 0.01



(b) $\phi(t)$

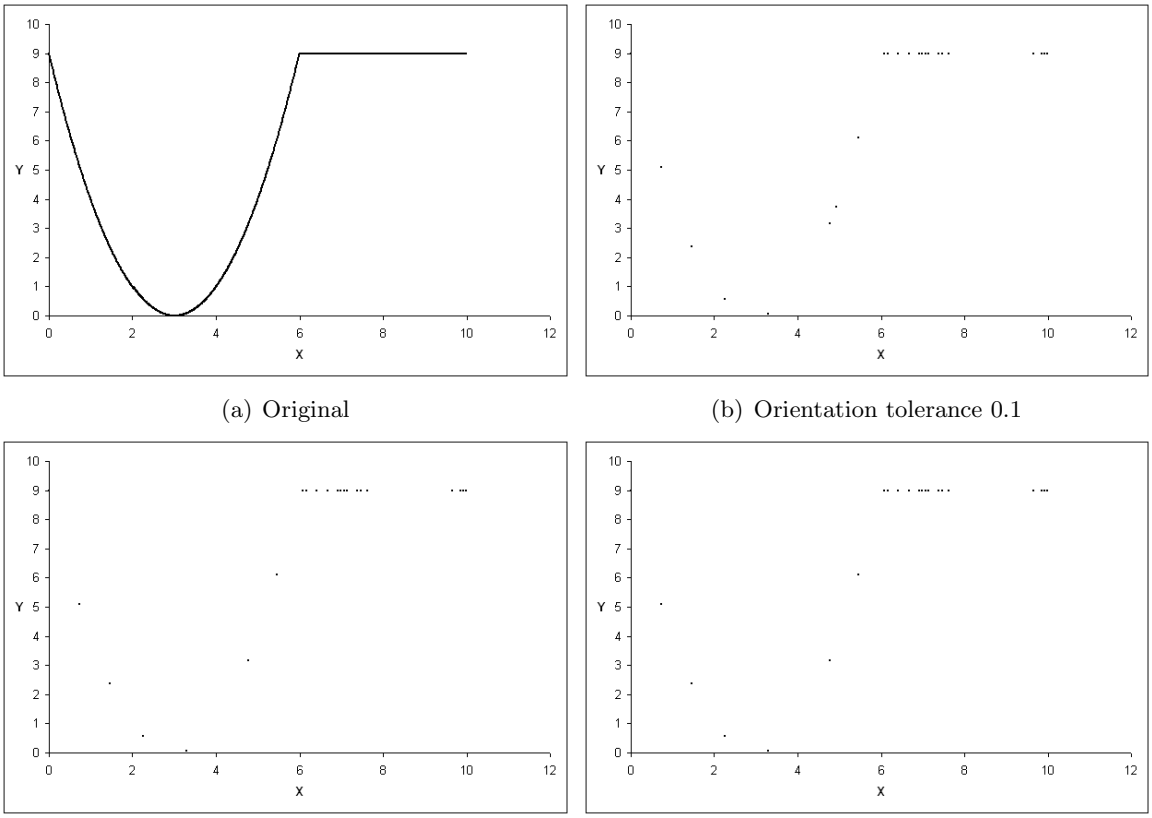


(c) XY-plane position tolerance 0.05



(d) $\phi(t)$

Figure 35: The plot show the *complex* dataset after reducing it with the joining algorithm, at the two smallest positional tolerances. In these cases the orientation tolerance did not have any influence on the result. Hence only one figure for each positional tolerance.



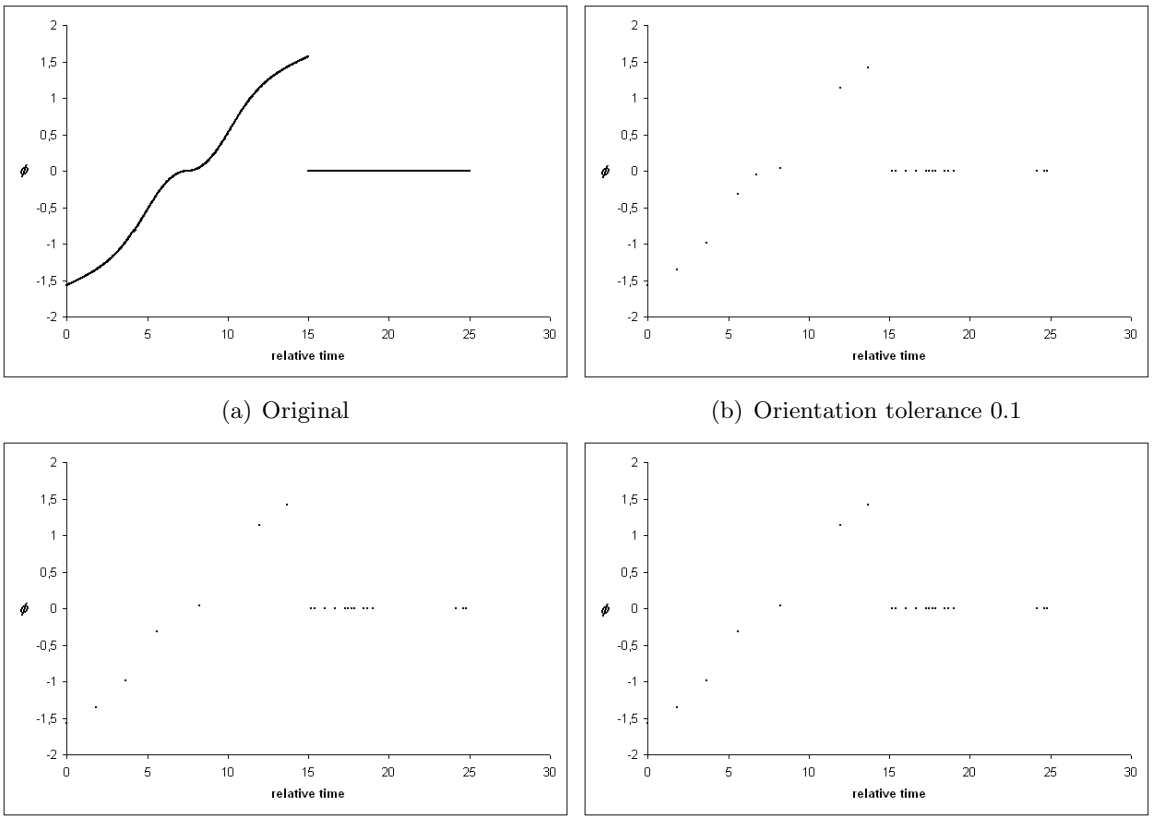
(a) Original

(b) Orientation tolerance 0.1

(c) Orientation tolerance 0.25

(d) Orientation tolerance 0.5

Figure 36: The XY-plane of the *complex* dataset filtered with the joining algorithm at positional tolerance 0.1.



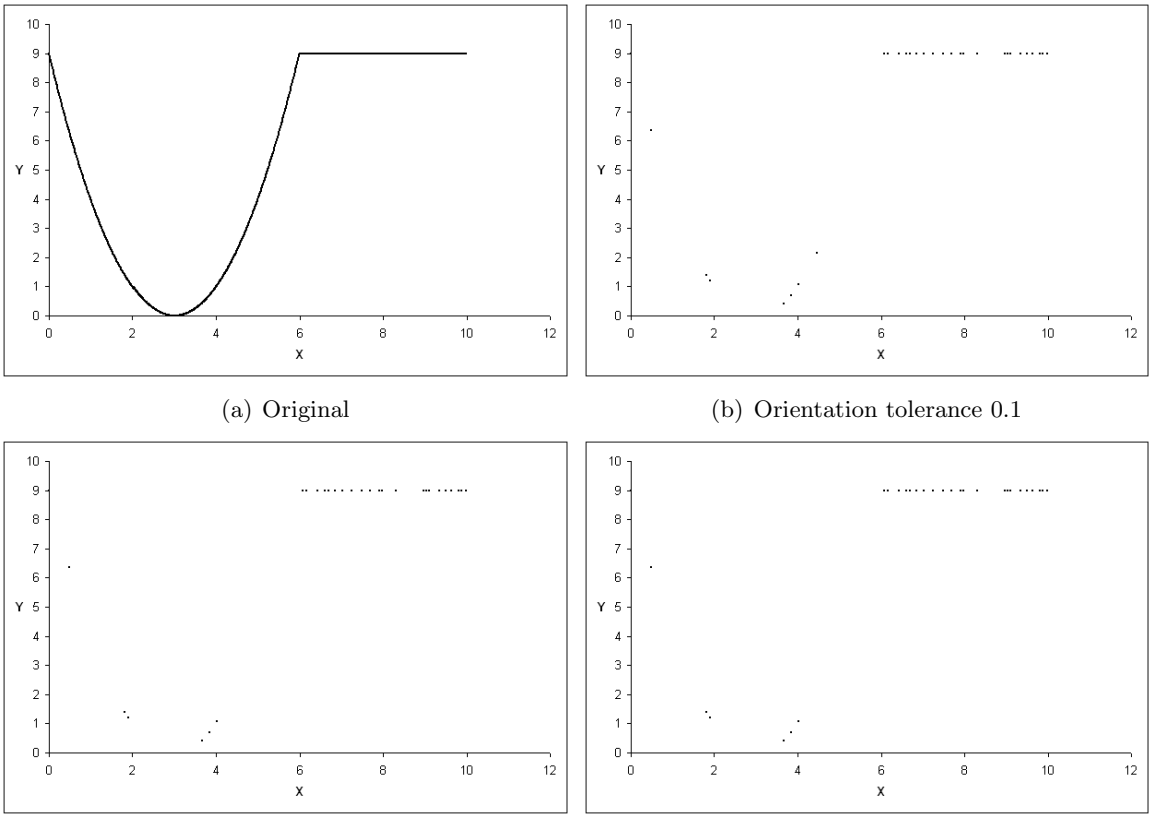
(a) Original

(b) Orientation tolerance 0.1

(c) Orientation tolerance 0.25

(d) Orientation tolerance 0.5

Figure 37: The $\phi(t)$ of the *complex* dataset filtered with the joining algorithm at positional tolerance 0.1.



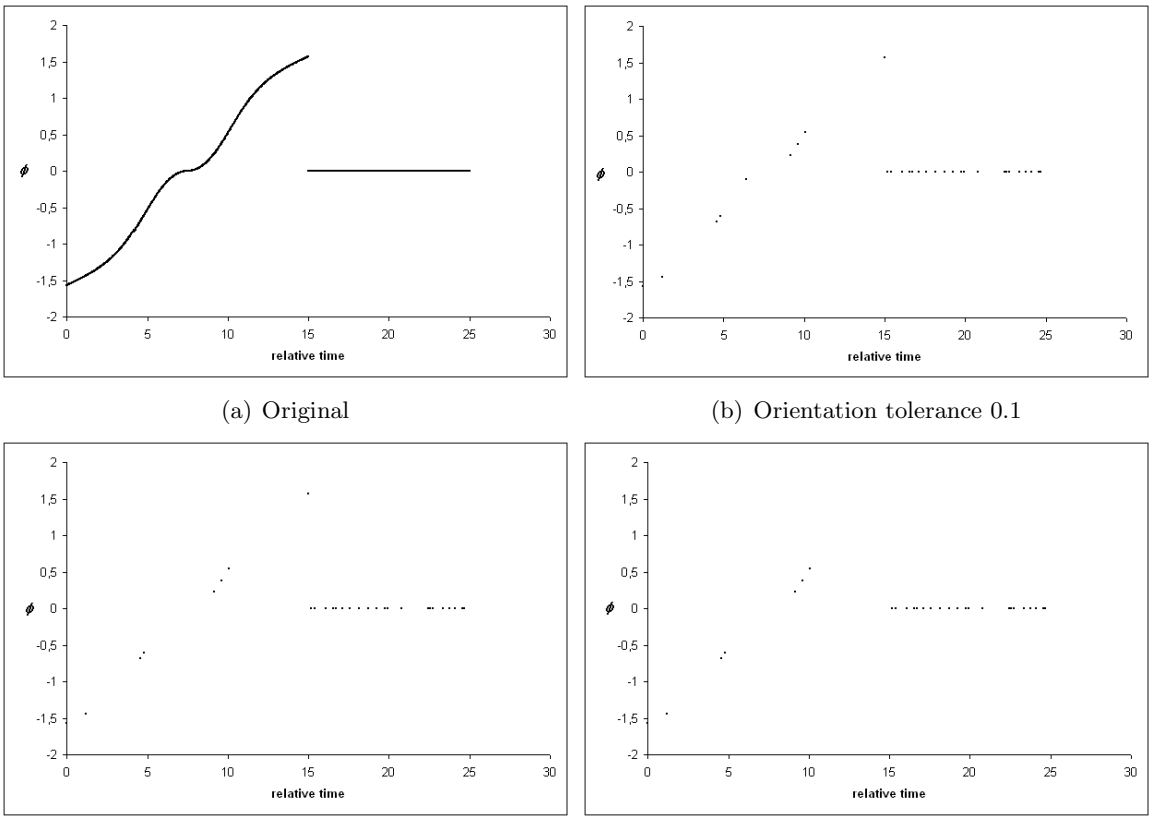
(a) Original

(b) Orientation tolerance 0.1

(c) Orientation tolerance 0.25

(d) Orientation tolerance 0.5

Figure 38: The XY-plane of the *complex* dataset filtered with the joining algorithm at positional tolerance 0.5.



(a) Original

(b) Orientation tolerance 0.1

(c) Orientation tolerance 0.25

(d) Orientation tolerance 0.5

Figure 39: The $\phi(t)$ of the *complex* dataset filtered with the joining algorithm at positional tolerance 0.5.

F C++ Code

This Appendix contain the code from selected methods.

F.1 Finding the third column of U in the SVD

```
/*
Input matrix A, i.e. a list of 3D points defining a plane.
Return normal vector to the plane, i.e. the third column
of U in the Singularvalue decomposition of A
*/


void normal(int n, float A[3][1000], float Z[3])
{
    // allowed numeric inaccuracy
    float tol = 0.0001;

    // Calculate AAT
    float T[3][3];
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<3; j++)
        {
            T[i][j]=0;
            for(int k=0; k<n; k++)
            {
                T[i][j] += A[i][k]*A[j][k];
            }
        }
    }


    // last element in gauss-elimination
    float t = T[2][2] - pow(T[0][2],2)/T[0][0]
     - pow((T[1][2]*T[0][0]
     - T[0][1]*T[0][2]),2)/(pow(T[0][0],2)*T[1][1]
     - T[0][0]*pow(T[0][1],2));

    if((-tol < t) && (t < tol)) Z[2] = 1;
    else Z[2] = 0;

    Z[1] =  -(T[0][0]*T[1][2]
     - T[0][1]*T[0][2])*Z[2]/(T[0][0]*T[1][1]
     - pow(T[0][1],2));


    Z[0] = -(T[0][1]*Z[1] + T[0][2]*Z[2])/T[0][0];
}
```

F.2 The two Euler methods

```
void EulerVelocity(int start, int end, coord tab[], float tol)
{
    float r[4], EA[3], V[3];

    Qtransform(tab[start].q, tab[end].q, r);
    Eulerangles(r, EA);

    for(int i=0; i<3; i++)
    {
        V[i] = fabs(EA[i] / (tab[end].time - tab[start].time));
    }

    for(int s=start+1; s<end-2; s++)
    {
        float rloc[4];
        float EAloc[3];
        float vl[3];

        Qtransform(tab[s].q, tab[s+1].q, rloc);
        Eulerangles(rloc, EAloc);

        for(int j=0; j<3; j++)
        {
            vl[j] = fabs(EAloc[j] / (tab[s+1].time - tab[s].time));
        }

        if(!((fabs(vl[0]-V[0])<tol)&&(fabs(vl[1]-V[1])<tol)&&(fabs(vl[2]-V[2])<tol)))
        {
            tab[s+1].b = true;
            if(start+2 < s) // are there any points beetween start and s?
            {
                EulerVelocity(start, s+1, tab, tol);
            }
            EulerVelocity(s+1, end, tab, tol);
            break;
        }
    }
}



void EulerAngle(int start, int end, coord tab[], float tol)
{
    float r[4], EA[3], eastart[3], eatmp[3], V[3], dv[3], dvtmp, dvmax;
    int imax;

    Qtransform(tab[start].q, tab[end].q, r);
    Eulerangles(r, EA);
```

```
    for(int i=0; i<3; i++)
    {
        V[i] = EA[i] / (float)(tab[end].time - tab[start].time);
    }

    Eulerangles(tab[start].q, eastart);
    dvmax = -10; imax = -1;

    for(int i=1; i < (end-start); i++)
    {
        Eulerangles(tab[start+i].q, eatmp);
        for(int t=0; t<3; t++) // each angle
        {
            dv[t] = eatmp[t]-(eastart[t]+V[t]*(tab[i+start].time-tab[start].time));
        }
        dvtmp = sqrt(pow(dv[0],2) + pow(dv[1],2) + pow(dv[2],2));

        if(dvtmp > dvmax)
        {
            dvmax = dvtmp;
            imax = i;
        }
    }
    // check if the maximum diviation is inside tolerance criteria
    Eulerangles(tab[imax+start].q,eatmp);
    for(int s=0; s<3; s++)
    {
        dv[s] = eatmp[s]-(eastart[s]+V[s]*(tab[imax+start].time-tab[start].time));
    }
    if((dv[0] > tol) || (dv[1] > tol) || (dv[2] > tol))
    {
        tab[imax+start].b = true;
        if(start < imax) EulerAngle(start, imax+start, tab, tol);
        if(imax < end) EulerAngle(imax+start, end, tab, tol);
    }
}
```