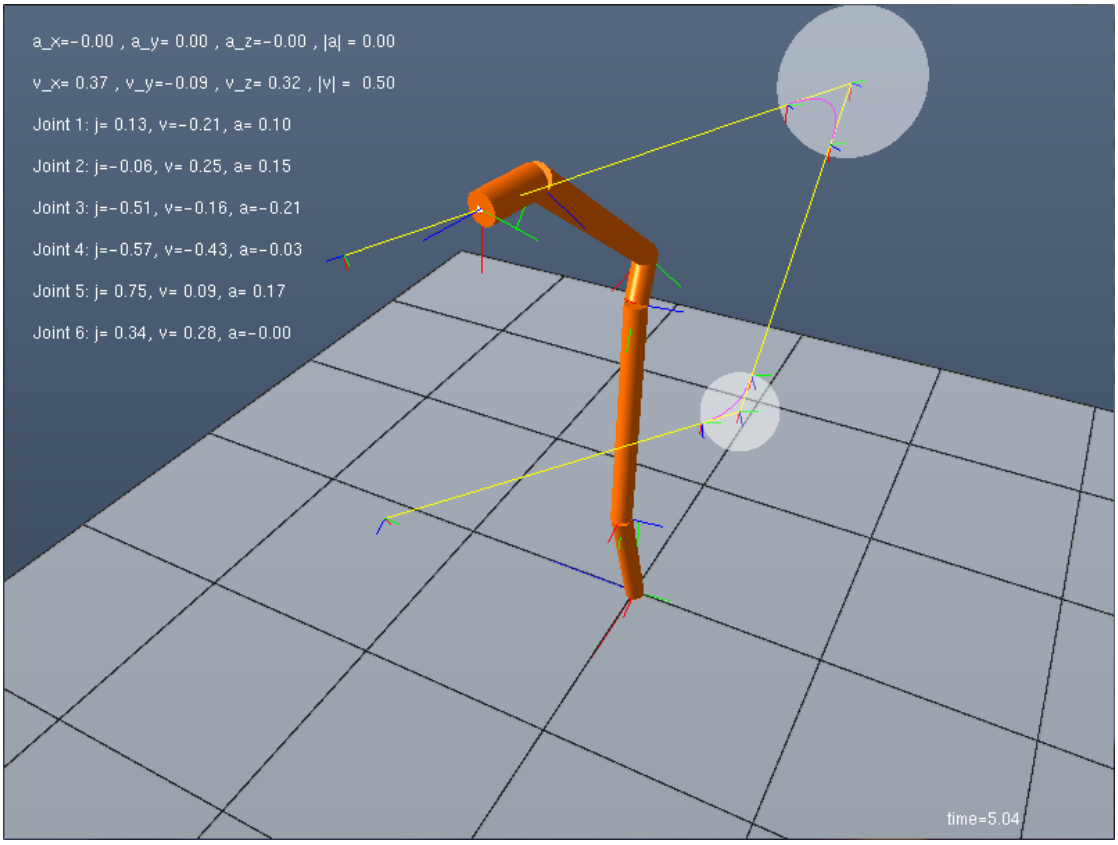
CHALMERS



Implementation of a generic Virtual Robot Controller

Master's Thesis in Systems, Control and Mechatronics

TIMUR USTYAN
VIKTOR JÖNSSON

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology
Göteborg, Sweden 2011
Master's Thesis EX034/2011

Master's Thesis EX034/2011

Implementation of a generic Virtual Robot Controller

Master's Thesis in Systems, Control and Mechatronics

TIMUR USTYAN
VIKTOR JÖNSSON

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology
Göteborg, Sweden 2011

Implementation of a generic Virtual Robot Controller
TIMUR USTYAN
VIKTOR JÖNSSON

Master's Thesis EX034/2011
Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
Screenshot of the graphical demonstrator of the path and trajectory planning performed by the Virtual Robot Controller.

Göteborg, Sweden 2011

Implementation of a generic Virtual Robot Controller
Master's Thesis in Systems, Control and Mechatronics
TIMUR USTYAN
VIKTOR JÖNSSON
Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
Chalmers University of Technology

Abstract

In order to simulate behavior and test non existing systems, a robot model approach can be applied to numerous mechanisms such as industrial robots welding a part onto a car body, or complex models of manikins simulating humans. Simulation of robot controllers is essential for path prediction in order to do reliable collision checking and accurate cycle time estimation.

This master thesis describes an algorithm of trajectory generation, simulating trajectories of real industrial robots. High calculation speed is necessary when automatically generating robot programs. To achieve this a generic *Virtual Robot Controller* (VRC), applicable to robots from different manufacturers, manikins, coordinate measurement machines, etc., has been implemented using a simplified model with only the most vital factors considered. In addition, a graphical demonstrator has been developed in order to visualize the simulations.

The VRC is validated against an existing robot controller in commercial software. The results prove the VRC to be quite accurate. Even with uncertainties in the dynamics properties of the robot model, the cycle time and general behavior on different trajectories are close to the existing robot controller.

Keywords: VRC, Fraunhofer Chalmers Centre, path, trajectory, kinematics, time optimal

Acknowledgements

We would like to thank everyone at FCC, and specifically the Geometry and Motion Planning group, for the opportunity to conduct our master thesis and a stimulating spring where our experience has grown significantly. Extra gratitude goes towards our main supervisor at FCC, Daniel Segerdahl, for guidance and practical help. We would also like to thank Johan Carlson, Robert Bohlin, Domenico Spensieri, and Staffan Björkenstam at FCC for many rewarding meetings with discussions and ideas of improvement.

Extra credit to our examiner Martin Fabian for feedback and administrative guidance.

Timur Ustyan
Viktor Jönsson
Göteborg, June 2011

List of notation

Symbols

$\mathbf{v}$	Vector v.
$\hat{\mathbf{v}}$	Normalized vector v.
$\mathbf{R}_i(\phi)$	Rotation matrix around axis i with the angle ϕ.
$\mathbf{R}_i^j$	Rotation matrix from i-th frame to j-th frame.
$\mathbf{T}_{(j-1),j}$	Transformation matrix from frame of joint $j-1$ to j.
q	Quaternion.
$\hat{q}$	Unit quaternion.
s	Normalized index between [0,1] on a track part.
V_{max}	Maximum allowed Cartesian space (or TCP) velocity.
Jv_{max}^i	Maximum allowed velocity for joint i.
Ja_{max}^i	Maximum allowed acceleration for joint i.

Abbreviations

VRC	Virtual Robot Controller.
FCC	Fraunhofer Chalmers Centre.
IPS	Industrial Path Solution.
TCP	Tool Center Point.
IK	Inverse Kinematics.
FK	Forward Kinematics.

Contents

1

Introduction

As the usage of industrial robots has grown, effective trajectory planning has become increasingly important. Optimizing the cycle times for each task the robot performs leads to an increase in productivity and a decrease in expenses. The general problem to solve is moving the robot from point A to point B, in a close to time optimal way, under certain velocity, acceleration and dynamic constraints. To determine the optimal trajectory for a robot off-line, an accurate model of the robot needs to be derived. In addition, kinematic laws and physical constraints need to be taken into consideration when performing trajectory planning.

Computerized simulations of systems or processes has become a vital part of verification and testing. Increasing computer performance allows for complex simulations of non existing systems and unwanted situations. In general, simulations are an important tool for companies to base decisions on in order to increase efficiency and reduce productions costs [1].

Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC) has been founded by *Chalmers* and the *Fraunhofer-Gesellschaft* to undertake and promote mathematical research to the benefits of industry, commerce, and public institutions [2]. FCC develops, among other projects, a simulation and optimization tool named *Industrial Path Solution* (IPS).

The first section in this chapter describes the purpose and aim of this thesis, followed by a set of limitations and a brief review of the previous work in this area. The last section gives an overview of the upcoming chapters in this thesis.

1.1 Main outcome of the thesis

This thesis describes the theory behind the implementation of a generic *Virtual Robot Controller* (VRC), simulating time optimal trajectory planning for a generic robot model. Implementation has been done in C++, however programming language specific details is excluded from the thesis. The purpose is to mimic the behavior of a real robot controller, and it is therefore important to verify the VRC against existing robot controllers in commercial software. The final aim is to integrate the VRC into IPS in order to improve the motion and time estimations in IPS. However, this integration is a later issue and is not discussed in this thesis.

1.2 Limitations

In order to complete this thesis within the specified time frame, the introduction of certain limitations were necessary. It is assumed that only one robot model is simulated at a time, eliminating the possibility for collisions with other robots. Furthermore, the trajectory planning assumes non existence of any obstacles in the surrounding environment. Obstacles and collisions will in fact be taken care of later in IPS.

Another limitation is the absence of external forces in the dynamics model of the robot. External forces that in theory would affect a real world robot include gravity and friction. This in turn would add to the internal inertia of the robot model, leading to lower acceleration limits throughout the path.

1.3 Previous work

A lot of previous work in this area has mainly focused on the path planning only, without consideration to time, velocity or dynamic constraints [3] [4] [5] [6] [7]. The denominator of these papers are the usage of spline functions. In [3], [4], and [5], the focus lies on transforming the Cartesian path into joint space by approximating the joint value profiles with splines. This thesis presents a more accurate approach using robot kinematics to calculate the joint values denser along the path. With effective routines for performing the robot kinematics, and modern computing power, this approach is generally fast enough for the main purpose of the VRC.

1.4 Structure of the report

The report consists of the following chapters:

2. **Underlying theory** - Both mathematical and robot specific theory applied later in the thesis is described more in detail in this chapter.

3. **Cartesian space path** - The position and orientation of a point in Cartesian (or TCP, work, task) space is described in absolute Cartesian x, y, z-coordinates. This chapter introduces the basic representation of a geometrical path.

4. **Joint space path** - A point in joint (or configuration) space is determined by a set of joint values. This chapter describes how to interpolate the path between the joint values.

5. **Trajectory** - This chapter adds the consideration of kinematic laws to the description of a path. Algorithms and kinematic equations are presented in order to move from point A to point B in a time optimal way, without violating possible velocity, acceleration, and dynamic constraints posed on the robot.

6. **Results** - This chapter presents the different tests made in order to validate the behavior of the implemented VRC.

7. **Summary** - A summary and conclusion of the results, together with some hints on possible future work are given at the end of the thesis.

2

Underlying theory

This chapter will introduce theory that will be applied later in the thesis. Both mathematical and robot specific concepts are presented here for the reader to acquaint with.

2.1 Orientation

Orientation and rotations of orientation can be represented in many ways. This section will briefly describe two possibilities: rotation matrices [8] and quaternions [9]. How to convert from quaternions to rotation matrices, and vice versa, is described in [10].

2.1.1 Rotation matrices

In three dimensions, it is convenient to represent the orientation as a 3×3 rotation matrix. A rotation matrix is an orthogonal matrix used for rotation of vectors and coordinate systems. This allows one vector to be expressed in terms of another vectors coordinate system. The rotation matrices around the x, y, z-axes are, for a right-handed coordinate system, given by:

$$\mathbf{R}_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$

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

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Using matrix multiplication, the rotation matrices can be combined to rotate from one coordinate system to another. For example, consider three coordinate systems O_0, O_1, and O_2. The rotation of O_i, relative O_j, is denoted $\mathbf{R}_i^j$ for $i, j = 0,1,2$ as $\mathbf{R}_2^0 = \mathbf{R}_1^0\mathbf{R}_2^1$. Note that $\mathbf{R}_1^0\mathbf{R}_2^1 \neq \mathbf{R}_2^1\mathbf{R}_1^0$, i.e the rotations are not commutative.

2.1.2 Unit quaternion

Quaternions extend the complex number system into four dimensions. Each quaternion, q, can be represented with one real part and three linearly independent imaginary parts such as

$$q = w + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z$$

and

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

where the real part is a scalar, and the imaginary part can be considered as a vector. Quaternions and rotation matrices share the same property of non-commutative multiplication, i.e $q_1 \cdot q_2 \neq q_2 \cdot q_1$.

A unit quaternion is commonly used for representation of orientation since it performs the same rotation regardless of the scalar part. In four-dimensional space, the unit quaternion lies on a sphere with radius one. According to [10], the unit quaternion can be shown to hold a rotation, θ, in the scalar part equivalent to $\cos\frac{\theta}{2}$, while the imaginary part points along the rotation axis, $\hat{\mathbf{r}}$, with magnitude $\sin\frac{\theta}{2}$. Consider the notation for the unit quaternion, $\hat{q} = \langle w,\mathbf{v}\rangle$, as

$$\hat{q}(\theta,\hat{\mathbf{r}}) = \left\langle \cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{r}} \right\rangle$$

and the multiplication of two quaternions, $\hat{q_1}\,\langle w_1,\mathbf{v}_1\rangle$ and $\hat{q_2}\,\langle w_2,\mathbf{v}_2\rangle$, is then defined as [10]

$$\hat{q_1} \cdot \hat{q_2} = \langle w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2\rangle$$

The inverse of a unit quaternion is defined as $\hat{q}^{-1} = \langle w, -\mathbf{v}\rangle$, leading to

$$\hat{q} \cdot \hat{q}^{-1} = \langle 1,0\rangle$$

Multiplying two unit quaternions corresponds to the multiplication of two rotation matrices. To rotate the vector $\mathbf{v}$ around $\mathbf{r}$ with the angle θ, using quaternions generates the new vector $\mathbf{v}'$ as

$$\mathbf{v}' = \hat{q} * \mathbf{v} * \hat{q}^{-1}$$

where

$$\hat{q} = \left\langle \cos\frac{\theta}{2}, \frac{\mathbf{r}}{||\mathbf{r}||}\sin\frac{\theta}{2} \right\rangle$$

In general, both rotation matrices and quaternions could be used for the same purpose. Although, since quaternions provide a more compact representation, less calculations are needed for performing a rotation [11].

2.2 Homogeneous transformation matrix

A homogeneous transformation matrix provides a convenient way of representing the transformation of position and orientation from one coordinate system to another. This section presents a brief summary from the material in [3].

Consider the two coordinate systems O_i and O_j, each holding a point described by a vector like

$$\mathbf{p}^i = p_x^i \hat{\mathbf{x}}_i + p_y^i \hat{\mathbf{y}}_i + p_z^i \hat{\mathbf{z}}_i$$
$$\mathbf{p}^j = p_x^j \hat{\mathbf{x}}_j + p_y^j \hat{\mathbf{y}}_j + p_z^j \hat{\mathbf{z}}_j$$

Let the vector $\mathbf{d}_j^i$ denote the relative distance between O_i and O_j, and the rotation matrix $\mathbf{R}_j^i$ describe the relative orientation. The point $\mathbf{p}^i$ can now be described as

$$\mathbf{p}^i = \mathbf{d}_j^i + \mathbf{R}_j^i \mathbf{p}^j \tag{2.1}$$

and

$$\mathbf{p}^j = \mathbf{R}_i^j \mathbf{p}^i - \mathbf{R}_i^j \mathbf{d}_j^i$$

in each coordinate system. To perform the transformations in a more compact way, the homogeneous transformation matrix is introduced as

$$\mathbf{T}_j^i = \begin{pmatrix} \mathbf{R}_j^i & \mathbf{d}_j^i \\ 0^T & 1 \end{pmatrix}$$

and $\mathbf{p}^i$ and $\mathbf{p}^j$ now becomes

$$\bar{\mathbf{p}}^i = \begin{pmatrix} \mathbf{p}^i \\ 1 \end{pmatrix} \qquad \bar{\mathbf{p}}^j = \begin{pmatrix} \mathbf{p}^j \\ 1 \end{pmatrix}$$

which gives

$$\bar{\mathbf{p}}^i = \mathbf{T}_j^i \bar{\mathbf{p}}^j$$

The result is the same as in (2.1), however less arithmetic operations are needed.

As [12] has shown, the computational difference between using quaternions or homogeneous transformation matrices is minor. The implemented VRC therefore supports both approaches.

2.3 Robot theory

A generic robot model consists of a number of links interconnected with joints. The links represent the physical structure of the robot, while the joint values determine the position and orientation for the links relative to each other, i.e. the pose of the robot. Each additional set of joint/link will increase the complexity of the model, since more degrees of freedom lead to a higher number of possible combinations of joint values for the same Cartesian position/orientation. A joint can be prismatic or rotating (or a combination of both), i.e performing either a translational or rotational motion. If two or more joints are aligned collinear, a singularity may occur in the configuration of the robot. This could lead to unpredictable behavior with infinite velocities and accelerations.

The Tool Center Point (TCP) is commonly placed at the end of a chain of links, and represents the position and orientation in absolute Cartesian coordinates. A robot model can hold several TCP's, depending on the physical structure, although this thesis is limited to one TCP per robot model. In order to transform from joint values to Cartesian coordinates and vice versa, robot kinematics must be used. The following two sections will describe this more in detail.

2.3.1 Forward kinematics

Forward, or direct kinematics is used to find the position and orientation of the TCP, as a function of the joint values. Robot links are described as a local coordinate transformations between joints. Given the joint value, the transformation matrix rotates or translates the coordinate system around either the x, y, or z-axis. The Cartesian position and orientation for each joint j, relative to previous joint is then described by the transformation matrix $\mathbf{T}_{(j-1),j}$. To find the transformation from the base coordinate frame to the j-th joint coordinate frame, the following formula is used:

$$\mathbf{T}_{0,j} = \mathbf{T}_{0,1}\mathbf{T}_{1,2}\cdots\mathbf{T}_{(j-1),j}$$

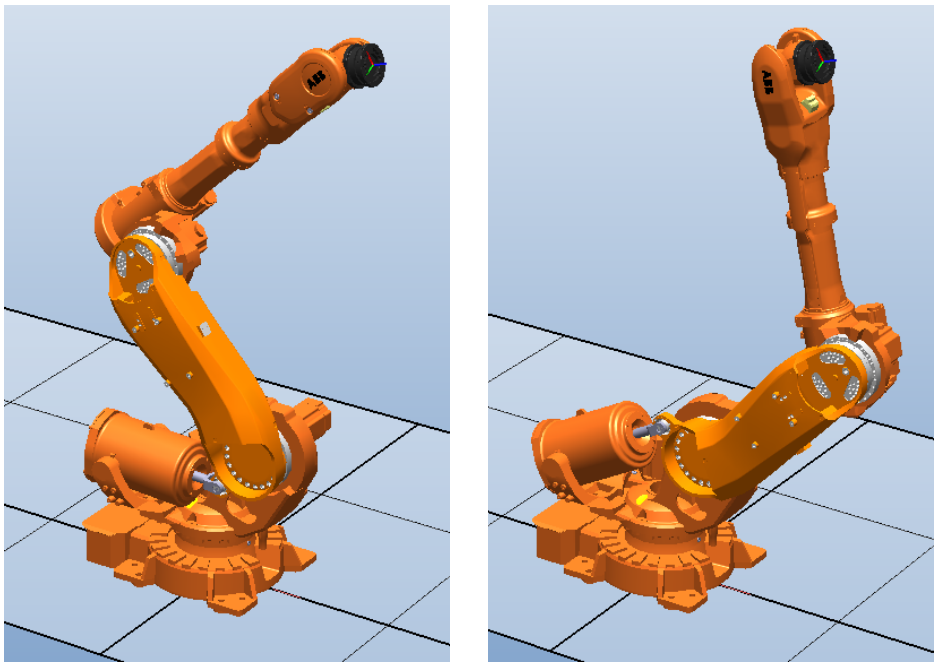
Transformation from the base coordinate frame to the TCP coordinates is then simply a matter of matrix multiplications like

$$\mathbf{T}_{0,TCP} = \mathbf{T}_{0,1}\mathbf{T}_{1,2}\cdots\mathbf{T}_{(n-1),n}\mathbf{T}_{n,TCP}$$

where n is number of joints, $\mathbf{T}_{n,TCP}$ is the local TCP transformation (from the face plate frame of the robot to the TCP). This is only valid if the links are structured like an open kinematic chain, i.e. there are no loops between the links. In forward kinematics, there is always a unique solution for a set of joint values.

2.3.2 Inverse kinematics

Given the desired TCP position, inverse kinematics is used to find the corresponding joint values. It is of vital importance to find the motion assigned to the TCP in Cartesian space also in joint space, since this is where the actual robot is controlled. Generally, this is a difficult problem since multiple solutions may exist, or even no solution at all. Figure 2.1 shows two different solutions to the same TCP position for an industrial robot model (IRB6640 from ABB's software *RobotStudio*) with six joints. When performing path planning for a robot in general, an increase in the number of joints leads to an exponential increase in the number of joint value combinations for any given TCP position



(a) Pose 1 (b) Pose 2

Figure 2.1: Different inverse kinematics solutions (screenshots are from ABB *RobotStudio*)

[13]. However, this complexity increase also depends on how the links are interconnected via the joints, i.e the degrees of freedom in the system.

The solution to the inverse kinematic problem can be obtained using different methods. The Jacobian transpose, Levenberg-Marquardt damped least squares, and the pseudoinverse are three relatively simple and effective methods. A good description of these three is provided in [14].

3

Cartesian space path

This chapter will introduce general semantics for the representation of a path. Most of its content is based on the work presented in [3].

A path is a geometrical track outlined in Cartesian space and without any consideration taken to velocity or acceleration. The path is constructed by a set of way points interconnected with segments. Each way point must be specified by the user and the shape of the segments depends on the desired movement between way points. In order to have as smooth path as possible, a small deviation from the way points is allowed. This deviation is also specified by the user, and is called a zone radius. Figure 3.1 shows one simple track. The solid line is the intended path and the dotted lines represents the segments inside the zone

In this thesis, only line and arc segments are considered in Cartesian space. Inside the zones the path is interpolated between the segments, together with a blending polynomial. The position along the path is represented as a 3-dimensional vector and unit quaternions are used for the orientation. In order to determine the position and orientation at any time along the path, a normalized index, s, is used. For each segment, $s = 0$ corresponds to the start of the segment, and $s = 1$ the end of the segment.

3.1 Line segment

The line segment represents a linear motion of the TCP between a start and end point in Cartesian space. Each start and end point of the segment is described by a position vector, $\mathbf{v}_{start}$ and $\mathbf{v}_{end}$, and a unit quaternion, $\hat{q}_{start}$ and $\hat{q}_{end}$. This allows the position and orientation of the TCP to be interpolated given an index s, at any time throughout the segment.

Position

Linear interpolation is used to calculate the position, $\mathbf{r}(s)$, along a line segment. The start and end point vectors of the segment together with the normalized index, s, is used like

$$\mathbf{r}(s) = (1 - s)\mathbf{v}_{start} + s \cdot \mathbf{v}_{end}$$

where $\mathbf{r}$ is a vector in absolute Cartesian x, y, z-coordinates.

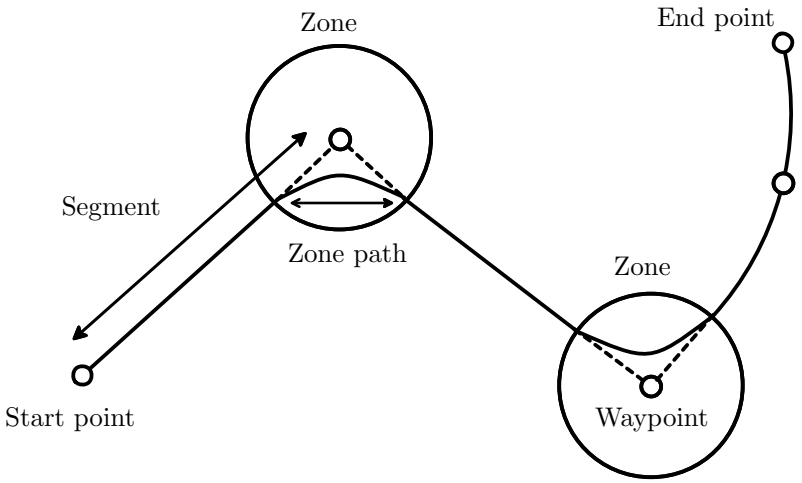


Figure 3.1: One simple path in Cartesian space, consisting of three segments

Orientation

The orientation at the start and end point of the segment is described by the unit quaternions $\hat{q}_{start}$ and $\hat{q}_{end}$ (see Section 2.1.2 for a description of quaternions). This allows the orientation to be interpolated along the segment as

$$\hat{q}(s) = \sin\left((1-s)\phi\right)\hat{q}_{start} + \sin\left(s\phi\right)\hat{q}_{end}$$

where $\phi = \arccos\left(\hat{q}_{start} \cdot \hat{q}_{end}\right)$. Orientation interpolation is illustrated in Figure 3.2.

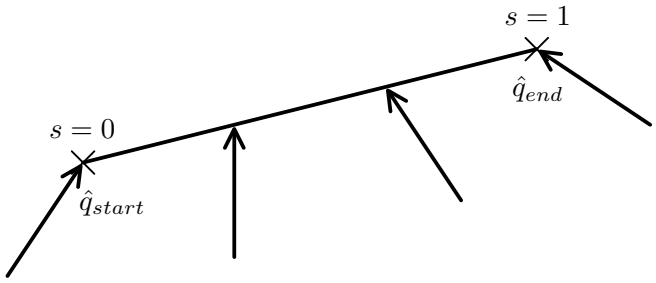


Figure 3.2: Orientation interpolation along a line segment

3.2 Arc segment

Each arc segment contains three points: start, intermediate, and end point. This represents a circular motion in Cartesian space, where the intermediate point determines the direction of the arc.

Position

This section will present one way to determine the position using interpolation between two unit vectors. In order to calculate the position correctly, the angle between the start

and end vectors of the segment, α, must be considered and three cases will arise:

1. $\alpha < \pi$. The unit interpolation can simply be performed between the start and end vector
2. $\alpha = \pi$. The start and end vector are opposite of each other, see Figure 3.3
3. $\alpha > \pi$. See Figure 3.4

Case 2 can occur whenever the Cartesian coordinates for the start and end vector forms an angle of π radians between them. Figure 3.3 gives one example of this. When this occurs, the interpolation should be performed either between start and intermediate point or intermediate and end point, depending on the value of s. Furthermore, the index s has to be adjusted accordingly. If s lies between start and intermediate vector

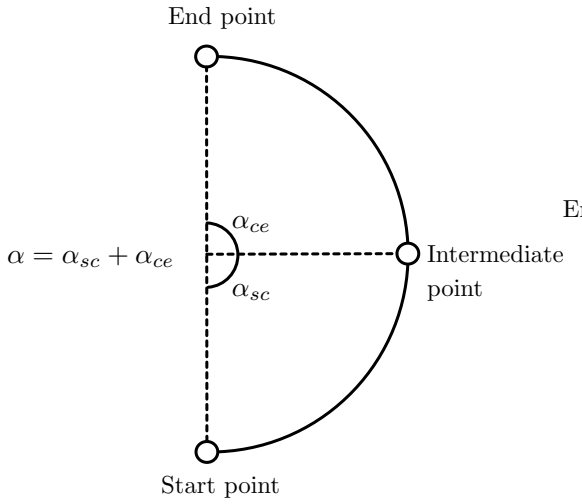


Figure 3.3: Case 2

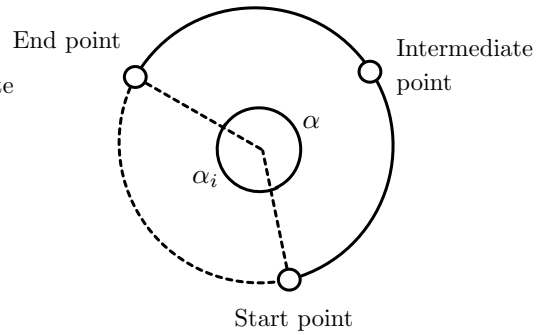


Figure 3.4: Case 3

($s \leq \frac{\alpha_{sc}}{\alpha}$) then

$$s = s \cdot \frac{\alpha}{\alpha_{sc}},$$

otherwise s should be adjusted between intermediate and end vector like

$$s = \frac{s - \frac{\alpha_{sc}}{\alpha}}{1 - \frac{\alpha_{sc}}{\alpha}}$$

The unit interpolation is performed between two unit vectors, $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$, chosen between the start, intermediate or end point. The resulting vector of the interpolation is the normalized position, $\hat{\mathbf{v}}(s)$

$$\hat{\mathbf{v}}(s) = \frac{\sin\left((1-s)\alpha\right)}{\sin\alpha}\hat{\mathbf{v}}_1 + \frac{\sin\left(s \cdot \alpha\right)}{\sin\alpha}\hat{\mathbf{v}}_2$$

In case 2 the index s is adjusted, as described earlier, before the interpolation. In case 3, the angle α is determined by the location of the intermediate point. The normalized position vector is then transformed to the real position vector using the center point, $\mathbf{v}_c$, and the radius, r, of the arc

$$\mathbf{r}(s) = \hat{\mathbf{v}}(s) \cdot r + \mathbf{v}_c$$

Orientation

There are two possible directions of orientation interpolation between start and end point. The intermediate point orientation is only used to determine in what direction the actual interpolation will be performed. It means that in the intermediate point the actual orientation will be different from the specified one.

When interpolating the orientation along an arc, the arc can be considered as a line bent around the face of a cylinder. This allows the orientation to be interpolated in the same way as for a line segment. Summarized in three steps [3]:

1. In order to compensate for the curvature of the cylinder, rotate $\hat{q}_{end}$ with the angle α around the z-axis of the cylinder and call the new orientation $\hat{q}'_{end}$.
2. Perform interpolation between $\hat{q}_{start}$ and $\hat{q}'_{end}$ in the same way as for a line segment, using the parametric index s.
3. Transform the result by rotating with $s \cdot \alpha$ in the opposite direction as in 1.

Using this method, the orientation will be correctly interpolated even for arcs with $\alpha \geq \pi$.

3.3 Zone path

The zones are introduced to allow the movement to be as smooth as possible when passing the way points. With a greater zone radius, the velocity along the zone path can be higher while satisfying acceleration constraints. This zone path must be constructed from the incoming and outgoing segments of the zone, and to guarantee a smooth trajectory, it must be continuous at the zone borders. In order to interpolate the zone path, the intersection between the zone and the segments must be known. A more elaborate description of the theory in this chapter can be found in [3].

Position

For two line segments the intersection between the incoming segment and the zone, s_{1z}, can be determined by

$$s_{1z} = \frac{R_z}{L_{s_1}}$$

where R_z is the zone radius and L_{s_1} is the total length of the incoming line segment. s_{1z} is an index along the segment between 0 and 1. For the outgoing segment, the intersection can thus be found as

$$s_{2z} = 1 - \frac{R_z}{L_{s_2}}$$

Note that this implies $\frac{R_z}{L_{s_2}} \leq 1$ which means that $R_z \leq L_{s_2}$, although in practice it is assumed that $R_z \leq \frac{1}{2} L_{s_2}$. In order words, to avoid overlapping zones the zone radius may not exceed half the segment length. In the same way the intersection between a zone and an incoming arc segment is determined by

$$s_{1z} = \frac{1}{\alpha} \arccos \left[1 - \frac{1}{2} \left(\frac{R_z}{R_0} \right)^2 \right]$$

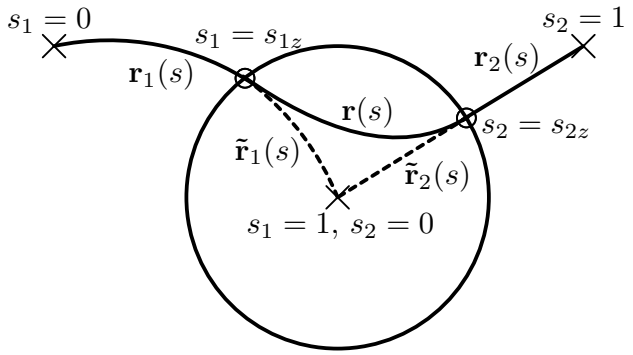


Figure 3.5: Two segments interconnected with one zone (redrawn from [3])

And for the outgoing

$$s_{2z} = 1 - \frac{1}{\alpha} \arccos \left[1 - \frac{1}{2} \left(\frac{R_z}{R_0} \right)^2 \right]$$

Using this information, a parable shaped zone path can be created between s_{1z} and s_{2z} inside the zone.

Consider the simple example in Figure 3.5 of two segments and one zone, where $\mathbf{r}_1(s)$ describes the position for the segment entering the zone and $\mathbf{r}_2(s)$ describes the position for the segment exiting. Denote the position for the part of the segments parts that lies inside the zone $\tilde{\mathbf{r}}_1(s)$ and $\tilde{\mathbf{r}}_2(s)$ respectively, using normalized index s. This gives $\tilde{\mathbf{r}}_1(0) = \mathbf{r}_1(s_{1z})$, $\tilde{\mathbf{r}}_1(1) = \mathbf{r}_1(1)$, $\tilde{\mathbf{r}}_2(0) = \mathbf{r}_2(0)$ and $\tilde{\mathbf{r}}_2(1) = \mathbf{r}_2(s_{2z})$, and the expressions for $\tilde{\mathbf{r}}_1(s)$ and $\tilde{\mathbf{r}}_2(s)$ are the following:

$$\tilde{\mathbf{r}}_1(s) = \mathbf{r}_1(s_{1z} + s \cdot (1 - s_{1z}))$$
$$\tilde{\mathbf{r}}_2(s) = \mathbf{r}_2(s \cdot s_{2z})$$

The position for the path inside the zone is then interpolated as

$$\mathbf{r}(s) = (1 - s)\tilde{\mathbf{r}}_1(s) + s \cdot \tilde{\mathbf{r}}_2(s)$$

In [3] it is shown that using linear interpolation for the zone path is not sufficient. This function fulfills the requirement of continuity in the zone borders, $\mathbf{r}(0) = \tilde{\mathbf{r}}_1(0)$ and $\mathbf{r}(1) = \tilde{\mathbf{r}}_2(1)$, but its first derivative does not. This could lead to jumps in velocity, and possible violations of the acceleration constraints in the joints. To solve this issue, a non-linear polynomial, $p(s)$, is introduced [3] (*Note*: $p(s)$ is called $\alpha(s)$ in reference)

$$\mathbf{r}(s) = \tilde{\mathbf{r}}_1(s) + p(s)(\tilde{\mathbf{r}}_2(s) - \tilde{\mathbf{r}}_1(s))$$

And the derivative of $\mathbf{r}(s)$ becomes

$$\mathbf{r}'(s) = \tilde{\mathbf{r}}_1'(s) + p(s)(\tilde{\mathbf{r}}_2'(s) - \tilde{\mathbf{r}}_1'(s)) + p'(s)(\tilde{\mathbf{r}}_2(s) - \tilde{\mathbf{r}}_1(s))$$

Which gives the following requirements on $p(s)$, in order to obtain continuity at the zone borders

$$p(0) = 0$$
$$p'(0) = 0$$
$$p(1) = 1$$
$$p'(1) = 0$$

A polynomial of degree $2N + 1$, where N is the number of derivatives, gives continuity at the zone border up to the N-th derivative. In the implemented VRC it is possible to interpolate the zone path with N=1, N=2, or N=3 which means that the polynomial becomes either

$$\begin{aligned} p_3(s) &= 3s^2 - 2s^3 \\ p_5(s) &= 10s^3 - 15s^4 + 6s^5 \\ p_7(s) &= 35s^4 - 84s^5 + 70s^6 - 20s^7 \end{aligned}$$

Orientation

The orientation can be interpolated in a similar manner as the position inside the zone. Let $\hat{q}_1(s)$ and $\hat{q}_2(s)$ represent the orientation of two segments intersecting with one zone. The parts of the orientation inside the zone is given by $\hat{\tilde{q}}_1(s)$ and $\hat{\tilde{q}}_2(s)$, where $\hat{\tilde{q}}_1(0) = \hat{q}_1(s_{1z})$ and $\hat{\tilde{q}}_2(1) = \hat{q}_2(s_{2z})$. This allows the orientation to be interpolated inside the zone between $\hat{\tilde{q}}_1(s)$ and $\hat{\tilde{q}}_2(s)$ as

$$\hat{q}(s) = \sin\left((1 - p(s))\phi\right)\hat{\tilde{q}}_1(s) + \sin\left(p(s)\phi\right)\hat{\tilde{q}}_2(s)$$

where $\phi = \arccos\left(\hat{\tilde{q}}_1(s) \cdot \hat{\tilde{q}}_2(s)\right)$, and $p(s)$ is the same polynomial used in the position interpolation.

4

Joint space path

In joint space, the TCP position and orientation is determined by a set of joint values. This means that every joint is dependent on the position and orientation of other joints, and this leads to many possible combinations of joint values for each position/orientation. Furthermore, the path consists of joint segments, interpolated from the joint values and the normalized index s.

In this chapter, two different methods for approximation of the joint space path of the TCP are discussed. The first method transforms the Cartesian path into joint space by approximating the joint values with a spline function. This transformation is based on the algorithm presented by [5], with some modifications. In the second method, the Cartesian path is divided into a number of points and the joint values in between these points are interpolated. The last section in this chapter summarizes the methods used in the implemented VRC.

Moving the TCP using joint values does not typically generate a linear and predictable motion in the trace of the TCP. The intersection point of the segments and the zones is therefore more difficult to find than in the Cartesian case. The chapter starts with a description of one way to find these intersection points, and also a brief introduction to spline functions.

4.1 Intersection between segment and zone

In order to interpolate the zone path, the intersection point between the joint segments and the zones must be known. One way to find these intersection points is to search iteratively along the joint segment and check with the Cartesian path when the position of index s is outside or inside the zone. Pseudo code for an algorithm that finds the intersection between a segment and a zone at the end of the segment is given in Algorithm 1. This algorithm starts at the center of the zone, and steps with a certain step size towards the start of the segment. The step size is then adjusted to reach some desired accuracy, ϵ. Finding the intersection between a segment and a zone at the start of the segment can be done in a similar way.

One limitation with this iterative search is that if there in fact are several intersections on incoming or outgoing joint segment, the algorithm will give only one intersection per segment, and the intersections in between will simply be ignored in the zone path interpolation.

Algorithm 1 Find intersection between segment and a zone with radius, R_z at the end of the segment

Require: $R_z > 0$
  $\Delta L \leftarrow 0$
  $stepSize \leftarrow 1.0$
  $s \leftarrow 1$
  repeat
    if $\Delta L \geq R_z$ **then**
      $s \leftarrow s + stepSize/2$
    else
      $s \leftarrow s - stepSize/2$
    end if
    $stepSize \leftarrow stepSize/2$
    $\Delta L \leftarrow ||positionAt(s) - positionAt(1)||$ {returns position vector, given index s}
  until $|\Delta L - R_z| > \epsilon$
  return s

4.2 Splines

A spline function is piecewise determined by polynomials. Figure 4.1 shows one simple example for two cubic polynomials, $c_1(x)$ and $c_2(x)$, passing through three control points, forming one spline function $C(x)$ approximating the function $f(x)$. The polynomials are then defined between their control points like [15]

$$c_i(x) = a_3(x - x_i)^3 + a_2(x - x_i)^2 + a_1(x - x_i) + a_0 \quad , i = 1,2,..,n-1$$

In order to have continuity at the control points and a smooth curve with continuous second derivative, the following requirements must be fulfilled

$$\begin{aligned} c_i(x_i) &= c_{i-1}(x_i) \quad , i = 2,3..,n \\ c_i'(x_i) &= c_{i-1}'(x_i) \quad , i = 2,3..,n-1 \\ c_i''(x_i) &= c_{i+1}''(x_i) \quad , i = 1,2..,n-1 \end{aligned}$$

where n is the total number of polynomials. The spline function is then calculated using the joint values and the derivative of the joint values at the two control points, J_p, J_p'. At the point p, the coefficients for the point $p + k$ of the spline, where k is the step size, is calculated as

$$\begin{aligned} a_0 &= J_p \\ a_1 &= J_p' \\ a_2 &= -\frac{3(J_p - J_{p+k}) + k(2 \cdot J_p' + J_{p+k}')}{k^2} \\ a_3 &= \frac{2(J_p - J_{p+k}) + k(J_p' + J_{p+k}')}{k^3} \end{aligned}$$

In [5] an algorithm for adjusting the step size is presented to reach a desired accuracy.

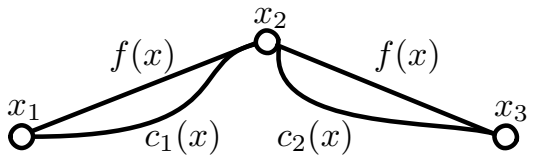


Figure 4.1: Spline function with two cubic polynomials

4.3 Path division

The path of a track is divided into a set of parts, see Figure 4.2. Each part consists of a number of key points, depending on the track division method used. Between the key points, either interpolation or spline functions can be used to approximate the joint values.

4.3.1 Divide path by ΔL_p

The Cartesian length of one part, L_p, is approximated with lines like

$$L_p = \sum_{i=0}^{N} \left|\left| \mathbf{r}\left(\frac{i+1}{N}\right) - \mathbf{r}\left(\frac{i}{N}\right) \right|\right|$$

where N is the total number of key points, and $\mathbf{r}$ represents the position in Cartesian x-y-z coordinates. Furthermore, the total length, L, of the path is the sum of all L_p. Using this, the path is sampled with a fixed length, ΔL_p, depending on the length of the current part

$$\Delta L_p = \frac{L_p}{round\left[\frac{L_p}{L}(N_p \cdot N_s)\right]}$$

where N_p is the number of parts on the path, and N_s is the desired number of samples per part. Since the length of the current part, L_p is considered when calculating ΔL_p, this will guarantee that there is always a key point at the zone borders, see Figure 4.2. Using ΔL_p, the index s can be determined in each key point and this enables the position and orientation to be calculated throughout the path.

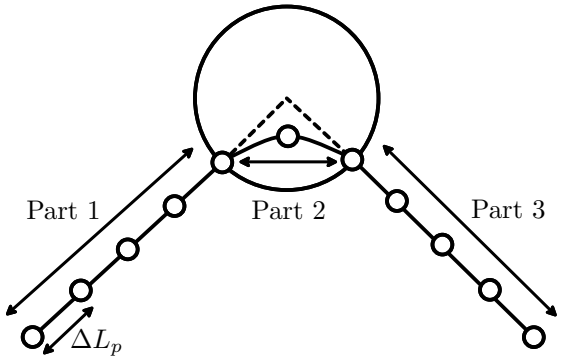


Figure 4.2: Simple track with the path divided and sampled with a, for each part, fixed ΔL_p

4.3.2 Divide path by Δs

One disadvantage of sampling with ΔL_p is that if there exists such a part that, $L_p < \Delta L_p$, then this part will simply be ignored when dividing the path into key points. Instead, sampling with a constant Δs as

$$\Delta s = \frac{1}{N_s}$$

ensures that no part of the path is missed. However, this means that all parts, regardless of length, are divided into an equal amount of key points.

4.3.3 Approximation of joint values between points

Each joint segment holds information about start and end joint values, and given the index s, values in between can either be linearly interpolated or approximated with spline functions. In fact, a linear interpolation corresponds to a spline function of degree one. A lot of the previous work in this area is mainly focused on using cubic splines [4] [5] [6] [7] [16] or even polynomials of degree five [17] [18].

In theory, using splines should render a more computational effective approximation of the joint values, due to less robot kinematic calculations. The trade off is accuracy, since the spline may deviate from the real joint space path in some cases. With modern computing power, the computational time complexity difference between spline functions and interpolation (with robot kinematics included) was not large enough to motivate the usage of spline functions in the implemented VRC.

4.4 Zone path interpolation

Inside the zone, the path can be divided by either Δs or ΔL_p in order to obtain the key points. The values in between the key points needs to be interpolated, and the interpolation can either take place in joint or Cartesian space. Consider the following three cases:

1. One incoming segment specified in Cartesian space, connected with a zone and an outgoing segment specified in Cartesian space.
2. One incoming segment specified in Cartesian space, connected with a zone and an outgoing segment specified in joint space.
3. One incoming segment specified in joint space, connected with a zone and an outgoing segment specified in joint space.
4. One incoming segment specified in joint space, connected with a zone and an outgoing segment specified in Cartesian space.

Let the position and orientation along the Cartesian segments be represented by $\mathbf{T}_1(s)$ and $\mathbf{T}_2(s)$. Consider also the joint values for the segments specified in joint space as $\mathbf{J}_1(s)$ and $\mathbf{J}_2(s)$.

In case 1, the most intuitive way to perform the interpolation is

$$\mathbf{T}(s) = (1 - s) \cdot \mathbf{T}_1(s) + s \cdot \mathbf{T}_2(s)$$

and using inverse kinematics (IK), the joint values are obtained as $\mathbf{J}(s) = IK(\mathbf{T}(s))$. This is interpolation in Cartesian space, although there is also a possibility to perform the interpolation in joint space for this case

$$\mathbf{J}(s) = (1 - s) \cdot IK(\mathbf{T}_1(s)) + s \cdot IK(\mathbf{T}_2(s))$$

Using forward kinematics (FK), the Cartesian position and orientation is calculated as $\mathbf{T}(s) = FK(\mathbf{J}(s))$. The zone path will be slightly different depending on if it is interpolated in joint or Cartesian space. However, solving inverse kinematics is generally a time consuming operation and therefore interpolation in Cartesian space is preferable in this case.

For case 2, interpolation in Cartesian space is performed as

$$\begin{aligned}\mathbf{T}(s) &= (1 - s) \cdot \mathbf{T}_1(s) + s \cdot FK(\mathbf{J}_2(s)) \\ \mathbf{J}(s) &= IK(\mathbf{T}(s))\end{aligned}$$

and in joint space as

$$\begin{aligned}\mathbf{J}(s) &= (1 - s) \cdot IK(\mathbf{T}_1(s)) + s \cdot \mathbf{J}_2(s) \\ \mathbf{T}(s) &= FK(\mathbf{J}(s))\end{aligned}$$

With the same amount of kinematics operations, it is not possible to decide which one is better in this case.

In case 3, the interpolation is more convenient in joint space due to less calculations needed. The interpolation is performed as

$$\begin{aligned}\mathbf{J}(s) &= (1 - s) \cdot \mathbf{J}_1(s) + s \cdot \mathbf{J}_2(s) \\ \mathbf{T}(s) &= FK(\mathbf{J}(s))\end{aligned}$$

in joint space, and as

$$\begin{aligned}\mathbf{T}(s) &= (1 - s) \cdot FK(\mathbf{J}_1(s)) + s \cdot FK(\mathbf{J}_2(s)) \\ \mathbf{J}(s) &= IK(\mathbf{J}(s))\end{aligned}$$

in Cartesian space.

Case 4 is the same situation as Case 2, and is handled the same way. The interpolation space matters for the trajectory calculations since the length of the path will differ.

4.5 Implemented approach

This chapter has presented different possibilities for constructing a joint space path of the TCP. The following list summarizes what has actually been implemented, with the previously provided motivations:

- **Path division** - The path is divided into key points by Δs to ensure that no part is missed, see Section 4.3.2.
- **Approximation of joint values** - To achieve better accuracy, the joint values are calculated by inverse kinematics along the path. This approach is in general not as computational effective as when the joint values are approximated with spline functions. See Section 4.3.3.
- **Zone path interpolation space** - Objective is to minimize the number of robot kinematic calculations. The following cases are from Section 4.4 with the final decision on which space the interpolation inside the zone should be performed in:
  Case 1: Interpolation is performed in Cartesian space.
  Case 2: Interpolation is performed in joint space.
  Case 3: Interpolation is performed in joint space.
  Case 4: Interpolation is performed in joint space.

5

Trajectory

All the trajectory calculation approaches could be divided into two different classes:

- **Cartesian space trajectories**, where calculations are performed in Cartesian space, and all the limits are converted into Cartesian velocity and accelerations limits.
- **Joint space trajectories**, where all calculations are done in joint space and limits are converted into joint velocities and acceleration limits.

Depending on the chosen method, it is necessary to use forward or inverse kinematics calculations for conversion from one space to another.

In practice, there are always physical constraints on the joint velocities and accelerations, determined by mechanism dynamics. In addition, Cartesian velocity constraints are often introduced for example due to process requirements such as sealing and gluing, which need a constant speed, or due to safety reasons.

Despite the space to use, there are always some general procedures and assumptions that are required for trajectory calculations. They are described in Section 5.1. In Section 5.2, two different methods for performing trajectory calculations, consisting of pure kinematics, are described. Both methods has been implemented in the VRC and their differences are discussed. The last section of this chapter adds the consideration of dynamics into the trajectory calculations.

5.1 Trajectory calculation basics

This section describes the basics of trajectory calculation. The one dimensional case could be considered without loss of generality. Assume that there is a known bound on acceleration without any bound on jerk, i.e. acceleration could be changed immediately from one level to another. This method is known as Line Segments Parabolic Blended (LSPB), and gives an optimal time under constraints on velocity and acceleration [17].

5.1.1 Time on the interval

To calculate the time between two points with known displacement, the following algorithm could be used. Given a distance d between points, start velocity v_0, desired end velocity v_e, velocity bound v_{max} and acceleration bound a_{max}. There are three possible scenarios:

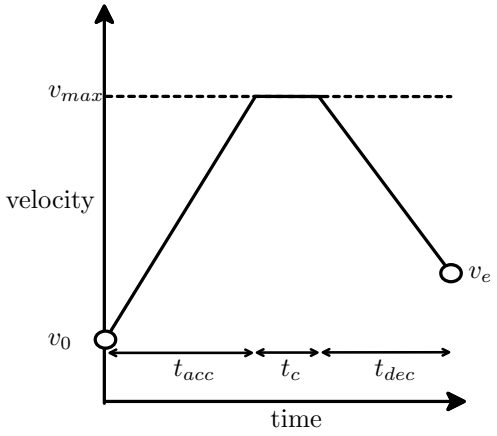


Figure 5.1: Scenario 1

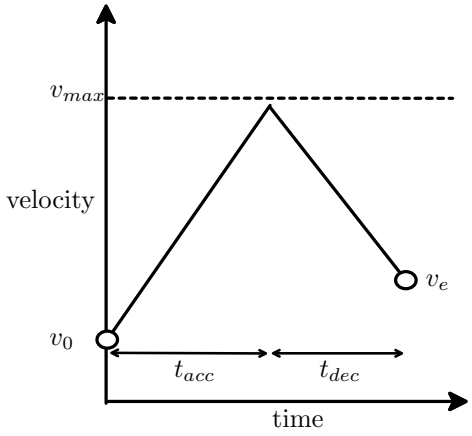


Figure 5.2: Scenario 2

1. Distance between points is large enough to reach maximum velocity.
   In this case, the trajectory is divided into three intervals: acceleration stage, constant speed and deceleration stage, see Figure 5.1. Corresponding times on the intervals are calculated as

$$\begin{aligned} t_{acc} &= \tfrac{(v_{max}-v_0)}{a_{max}}, \quad d_{acc} = v_0 t_{acc} + \tfrac{a_{max} t_{acc}^2}{2} \\ t_{dec} &= \tfrac{(v_{max}-v_e)}{a_{max}}, \quad d_{dec} = v_{max} t_{dec} - \tfrac{a_{max} t_{dec}^2}{2} \\ t_c &= \tfrac{d - d_{acc} - d_{dec}}{v_{max}} \end{aligned}$$

   Total time t between points is then

$$t = t_{acc} + t_c + t_{dec}$$

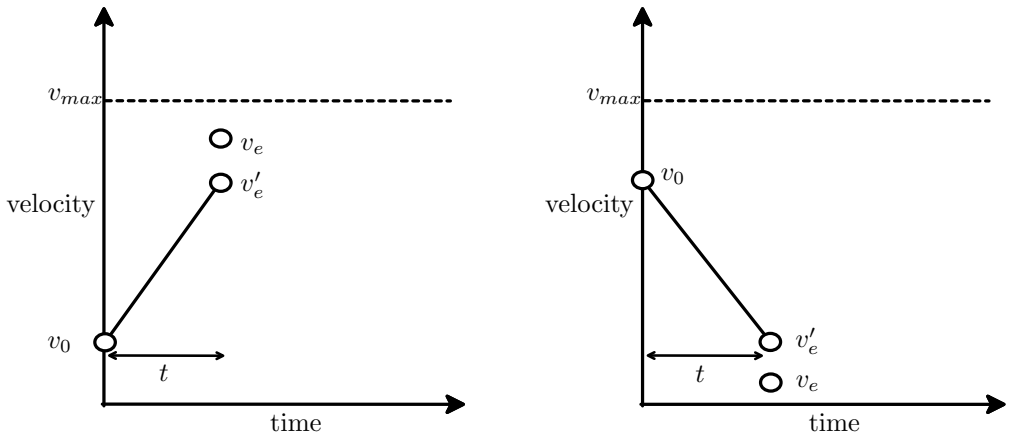
2. Distance between points is not large enough to reach maximum velocity, but large enough to reach desired end velocity.
   In this case trajectory consist of two intervals: acceleration stage and deceleration stage, see Figure 5.2. Corresponding time on the intervals:

$$\begin{aligned} t_{acc} &= \frac{-v_0 + \sqrt{\frac{v_0^2 + v_e^2}{2} + d \cdot a_{max}}}{a_{max}} \\ t_{dec} &= t_{acc} + \tfrac{v_0 - v_e}{a_{max}} \\ t &= t_{acc} + t_{dec} \end{aligned}$$

3. Distance between points is not large enough to reach desired end velocity.
   In this case trajectory has just one interval: acceleration (Figure 5.3(a)) or deceleration (Figure 5.3(b)) stage, depending on the velocities. Time on the interval is then

$$t = \begin{cases} \frac{v_0 - \sqrt{v_0^2 - 2d \cdot a_{max}}}{a_{max}}, & \text{if } v_0 \geq v_e \\ \frac{-v_0 + \sqrt{v_0^2 + 2d \cdot a_{max}}}{a_{max}}, & \text{if } v_0 < v_e \end{cases}$$

Generally, scenario 3 might be tricky to handle. If $v_0 < v_e$, then the new final velocity, which is possible to reach with maximum acceleration/deceleration, is v_e' and this velocity is used as the start velocity on the next interval. On the other hand, if $v_0 > v_e$, it might be critical to reach v_e and therefore v_0 needs to be lowered. This could cause a chain reaction where the lowering of v_0 must be propagated backwards in order to satisfy earlier intervals.



(a) $v_0 < v_e$ (b) $v_0 \geq v_e$

Figure 5.3: Scenario 3

5.1.2 TCP velocity limit inside zones

The maximum bound on the TCP velocity, V_{max}, is specified on each segment. However, inside the zones there are different approaches to determine the velocity limit on the TCP. Consider the case of one zone with an incoming and outgoing segment. The maximum velocity allowed along the zone path can then be chosen from the segments in different ways:

1. $V^z_{max} = min\,[V^{s1}_{max}, V^{s2}_{max}]$. The maximum allowed velocity of the TCP inside the zone is determined by the lowest limit on the two segments.

2. $V^z_{max} = max\,[V^{s1}_{max}, V^{s2}_{max}]$. The maximum allowed velocity of the TCP inside the zone is determined by the highest limit on the two segments.

3. $V^z_{max} = \begin{cases} V^{s1}_{max}, & s \leq 0.5 \\ V^{s2}_{max}, & s > 0.5 \end{cases}$, for $s = [0,1]$.
   The maximum allowed velocity of the TCP inside the zone is equal to the incoming segment value for the first half, and to the outgoing segment value for the second half.

4. $V^z_{max} = (1-s)V^{s1}_{max} + s \cdot V^{s2}_{max}$, for $s = [0,1]$.
   The maximum allowed velocity of the TCP inside the zone is interpolated between the limits on the two segments.

The first approach may not be time optimal in some cases, whereas the second method may be unsafe for some applications. The third approach gets rid of the disadvantages of the first two methods. The fourth method gives a more smooth trajectory, but it is more complicated. The implemented VRC has a possibility to choose between the suggested methods. However, the third method is used by default, since it is simple and lacks the disadvantages of the first and second method.

5.2 Trajectory calculation methods

Trajectory calculation addresses the problem of assigning feasible velocities to each key (sample) point, under velocity and acceleration constraints. This section will present two implemented methods of calculating the trajectory throughout the track. The difference lies in the sampling of the track, where the first method samples with a constant Δs (see Section 4.3.2), and the second method works more like a conventional robot controller and samples with $\Delta time$.

5.2.1 Method 1, sampling with a fixed Δs

The first trajectory calculation method to be presented in this thesis can be summarized in the following steps:

1. Calculate the local velocity limit at each pre-calculated key point, based on the curvature of the path.
2. Calculate velocity limits backwards, assuming that the end velocity of the path is equal to zero.
3. Starting from some key point, make an optimistic estimate of the allowed velocity to the next key point. If above any of the previously calculated limits, use the limit instead. Knowing velocities between key points, calculate the time as in Section 5.1.

Local velocity limits

There are three constraints to take into consideration when calculating the time at each key point, k: maximum allowed Cartesian velocity, V_{max} (5.1), velocity and acceleration constraint for joint i, Jv^i_{max} (5.2) and Ja^i_{max} (5.5). The most limiting of these three constraints will decide the local velocity limit for each joint.

Figure 5.4 shows three key points, sampled with a fixed Δs. The minimum time for satisfying the Cartesian velocity limit, V_{max}, is calculated using the Cartesian distance between the key points as

$$t_c = \frac{L_{k+1} - L_k}{V_{max}} \tag{5.1}$$

In the same way, the minimum time needed to satisfy the joint velocity constraint is calculated as

$$\begin{aligned} t^i_{jv} &= \tfrac{J^i_{k+1} - J^i_k}{Jv^i_{max}} \quad , i = 1,2,3,..,n \\ t^{min}_{jv} &= max\left[t^1_{jv}, t^2_{jv}, t^3_{jv}, .., t^n_{jv}\right] \end{aligned} \tag{5.2}$$

where n is the total number of joints and t^{min}_{jv} represents the time for the most constraining joint (i.e. the "slowest" joint).

Under the assumption of equidistant sampling, the following relationship between the acceleration constraint and minimum time on the interval holds

$$Ja^i_{max} = \tfrac{J^i_{k+1} - 2J^i_k + J^i_{k-1}}{t^{i\,2}_a} \quad , i = 1,2,3,..,n \tag{5.3}$$

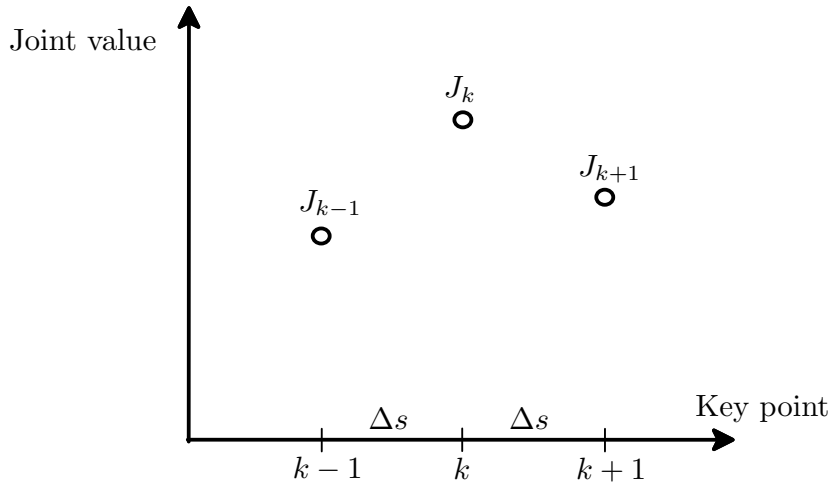


Figure 5.4: The joint values at each key point for one joint.

However, sampling with Δs may lead to large differences in distance between the key points at the borders between, for example, a short and a long part, which will lead to unnecessary increase of t_a^i. To improve the time estimation for this case, an interpolation coefficient, x, dependent on the Cartesian length between points is introduced. Then averaged joint displacement between the key points are introduced as

$$\Delta J^i = \left(J_{k+1}^i - J_k^i\right) + x\left(J_{k-1}^i - J_{k+1}^i\right) \tag{5.4}$$

where $x = \frac{L_{k+1}-L_k}{L_{k+1}-L_{k-1}}$. Substituting this displacement for the distance measurement in (5.3), and solving for t_a^i gives

$$\begin{aligned} t_a^i &= \sqrt{\frac{|2\Delta J^i|}{Ja_{max}^i}} \qquad\qquad , i = 1,2,3,..,n \\ t_a^{min} &= max\left[t_a^1,t_a^2,t_a^3,..,t_a^n\right] \end{aligned} \tag{5.5}$$

Similar to the case in (5.2), the time t_a^{min} represents the time needed for the most constraining joint under acceleration constraints.

Finally, to calculate the local velocity limit for joint i at key point k:

$$Jv_{lim_k}^i = \frac{J_{k+1}^i - J_k^i}{t_k} \tag{5.6}$$

where t_k is chosen from equations (5.1), (5.2), (5.5) as

$$t_k = max\left[t_c, t_{jv}^{min}, t_a^{min}\right] \tag{5.7}$$

Note: The local velocity limit, $Jv_{lim_k}^i$, calculated in (5.6), is in fact the average velocity limit on the interval between key points k and $(k+1)$.

Velocity limits backwards

The local velocity limits are calculated in the previous subsection. Now the backwards velocity limits need to be adjusted so that it is feasible to reach zero velocity at the end of the path, or slow down along the path before sharp turns and slow segments. To be able to stop at some key point, the velocity at the previous key point should not exceed a value greater than what the acceleration constraint allows for deceleration (Figure 5.3(b) shows an example of this case). If it does, then the velocity at the previous key point needs to be lowered, and the change needs to be propagated backwards.

Algorithm 2 presents one way to calculate the backwards velocity limits for the simplified case of starting at the end of the path, and converging when the start is reached. To find the exact velocity limit at the key point, interpolation is performed between average interval velocities throughout the path to achieve constant acceleration on both intervals between the key points. The interpolation coefficient depends on the previously calculated (in (5.7)) minimal time between points.

Figure 5.5, gives one simple example of starting at the last key point of the path where all joint velocities are zero, adjusting the joint velocity limits backwards to be able to reach this velocity at the end point. As in Algorithm 2, the local limit at the key point, Jv^i_{loc}, is adjusted if the backwards limit, Jv^i_{back}, is lower to form the final limit at the key point, Jv^i_{lim}.

Algorithm 2 Adjust local velocity limits $Jv^i_{lim_k}$ backwards if needed, to reach zero end velocity

Require: Key points, $k = 0,1,2,..,m$
  $k \leftarrow m-1$
  $Jv^i_{lim_{k+1}} \leftarrow 0$ {Set the velocity at the last key point to 0}
  while $k > 0$ **do**
    for all joints i **do**
      $Jv_i \leftarrow \sqrt{Jv^{i2}_{lim_{k+1}} + 2Ja^i_{max}\left(J^i_{k+1} - J^i_k\right)}$ {Estimated velocity}
      $V_{lim} \leftarrow Jv^i_{lim_{k-1}} + \left(\frac{t_{k-1}}{t_{k-1}+t_k}\right)\left(Jv^i_{lim_k} - Jv^i_{lim_{k-1}}\right)$ {Local velocity limit}
      if $Jv_i < |V_{lim}|$ **then**
        $Jv^i_{lim_k} \leftarrow Jv_i \cdot sign\left(V_{lim}\right)$
      else
        $Jv^i_{lim_k} \leftarrow V_{lim}$
      end if
    end for
    $k \leftarrow k-1$
  end while

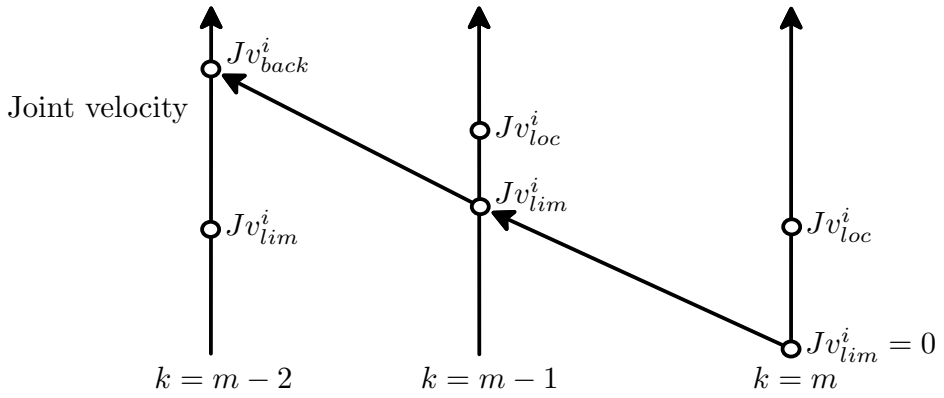


Figure 5.5: The velocity limit for joint i at key point k adjusted backwards

Forward calculations

With a local and backwards velocity limit specified at each key point, allowing to slow down without violating any constraints, the actual trajectory between the key points could be calculated. In general, the calculations can start at any key point, as long as the velocity for joint i at the previous key point, Jv^i_{k-1}, is given. This holds under the assumption that the velocity at the first key point, Jv^i_0, is equal to zero. The velocity at the next key point, $Jv^i_{est_k}$, can then be optimistically estimated, assuming maximum acceleration Ja^i_{max} on the given interval ΔJ between points as

$$Jv^i_{est_k} = \sqrt{{Jv^i_{k-1}}^2 + 2Ja^i_{max}\Delta J} \tag{5.8}$$

where $\Delta J = J^i_k - J^i_{k-1}$. If this estimate of the velocity is higher than the allowed limit, then update the estimate as $Jv^i_{est_k} = Jv^i_{lim_k}$.

The calculation of the time allows for two different approaches in this case. Either a constant acceleration between the key points is used, or one interval of max acceleration followed by constant velocity (see Figure 5.6 and Figure 5.7). Using a constant acceleration is not time optimal, since the desired velocity is not reached until the end of the time interval. Acceleration or deceleration with the maximum allowed value divides the time interval into two parts; acceleration/deceleration and constant velocity. Similar to Section 5.1, the time between the key points for each joint i is calculated as

$$\Delta t^i = \begin{cases} \frac{\Delta J}{Jv^i_{est_k}} + \frac{\left(Jv^i_{k-1} + Jv^i_{est_k}\right)^2}{2Ja^i_{max}Jv^i_{est_k}}, & \text{if } Jv^i_{k-1} \leq Jv^i_{est_k} \\ \frac{\Delta J}{Jv^i_{k-1}} + \frac{\left(Jv^i_{k-1} + Jv^i_{est_k}\right)^2}{2Ja^i_{max}Jv^i_{k-1}}, & \text{if } Jv^i_{k-1} > Jv^i_{est_k} \end{cases} \tag{5.9}$$

The time interval, Δt^i, is calculated between the same key points for all joints. This gives the final time interval for all the joints as

$$\Delta t = max\left[\Delta t^1, \Delta t^2, .., \Delta t^n\right] \tag{5.10}$$

With a known time interval, Δt, and a defined distance between the joint values, ΔJ, the joint velocity Jv^i_k at each key point k is given by

$$Jv^i_k = \frac{2\Delta J}{\Delta t} - Jv^i_{k-1} \tag{5.11}$$

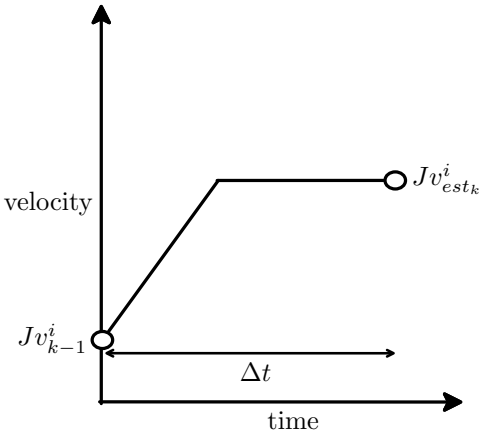


Figure 5.6: When $Jv^i_{k-1} \leq Jv^i_{est_k}$

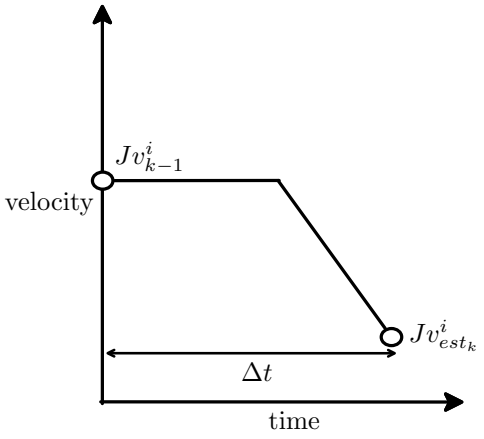


Figure 5.7: When $Jv^i_{k-1} > Jv^i_{est_k}$

and in the very same way, the velocity of the TCP in Cartesian space is calculated as

$$V_k = \frac{2\Delta L}{\Delta t} - V_{k-1} \tag{5.12}$$

where ΔL is the Cartesian length of the path between points.

5.2.2 Method 2, sampling with a fixed Δtime

A conventional robot controller has in general an internal sampling frequency, which allows the controller to calculate the trajectory between a fixed interval in time. In order to emulate this behavior without the concept of real time, a search for an appropriate sampling time is needed.

The trajectory calculations are basically the same as for Method 1, although no precalculated key points exists and the time is obtained slightly different throughout the path. The work-flow of the algorithm is summarized as:

1. Calculate velocity limits backwards with a fixed Δs (as in Method 1), assuming that the end velocity of the path is equal to zero.
2. Set current point to start of the track. Set Δs to some initial value.
3. Set next point at the end of Δs interval from current. Find local velocity limit at next point (described below).
4. As in Method 1, calculate the time Δt on the interval Δs, with consideration to local and backwards velocity limits of all joints.
5. Compare if Δt is approximately equal to some desired sampling frequency ΔT. If not, adjust Δs and jump to step 3.
6. Calculate velocities for all joints using the Δt.

Local velocity limits

Local velocity limit calculation is similar to the previous method, but instead of using pre-calculated key points to form the intervals, small position deviations around the current points are used. Given a point p, the local velocity limit needs to be calculated in this point. Using small deviations of the position along the path gives the neighboring points, $(p-1)$ and $(p+1)$. Denote the distance between p and $(p-1)$ in Cartesian space as ΔL_{prev} and between $(p+1)$ and p as ΔL_{next}. Together with the maximum allowed Cartesian velocity, V_{max}, the minimum time to satisfy the TCP velocity constraint on the interval is given by

$$t_c^{prev} = \frac{\Delta L_{prev}}{V_{max}} \tag{5.13}$$

$$t_c^{next} = \frac{\Delta L_{next}}{V_{max}} \tag{5.14}$$

In a similar way, the displacement between the joint values at the points are given by ΔJ_{prev} and ΔJ_{next}. This gives the minimum time to satisfy the joint velocity constraint Jv_{max}^i for joint i, as

$$t_j^{prev^i} = \frac{\Delta J_{prev}^i}{Jv_{max}^i} \tag{5.15}$$

$$t_j^{next^i} = \frac{\Delta J_{next}^i}{Jv_{max}^i} \tag{5.16}$$

To simplify, denote the maximum of $t_j^{prev^i}$ and $t_j^{next^i}$ in (5.15) and (5.16), for all joints, as t_j^{prev} and t_j^{next} respectively (as in (5.2)).

In order to satisfy the acceleration constraint at the point p, both neighboring intervals must be taken into consideration. The minimum time, t_a^i, needed to satisfy the acceleration limit is calculated as in Method 1 (5.5), with an added interpolation between the intervals. In summary, the distance ΔJ^i in (5.5) is replaced by

$$\Delta J^i = \Delta J_{next}^i + x\left(-\Delta J_{prev}^i - \Delta J_{next}^i\right) \tag{5.17}$$

where

$$x = \begin{cases} \frac{\Delta L_{next}}{\Delta L_{next}+\Delta L_{prev}}, & \text{if } \Delta L_{next} \neq 0 \text{ } or \text{ } \Delta L_{prev} \neq 0 \\ 0.5, & \text{else} \end{cases} \tag{5.18}$$

Again, denote the maximum of t_a^i for all joints as t_a. The maximum times on the intervals are chosen as $t^{prev} = max\left[t_c^{prev}, t_j^{prev}, t_a\right]$ and $t^{next} = max\left[t_c^{next}, t_j^{next}, t_a\right]$. The local velocity limit at the point, p, for each joint, i, is now calculated as an interpolation between the two average velocities on the intervals, where the interpolation coefficient depends on the interval times.

$$Jv_{lim_p}^i = \frac{\Delta J_{prev}^i}{t^{prev}} + x\left(\frac{\Delta J_{next}^i}{t^{next}} - \frac{\Delta J_{prev}^i}{t^{prev}}\right) \tag{5.19}$$

$$x = \frac{t^{prev}}{t^{prev} + t^{next}} \tag{5.20}$$

Finding a suitable Δtime

The procedure of calculating the velocity limits backwards differs from the rest of the calculations, and it is performed in the same way as for Method 1. This results in two velocity limits, one local and one backwards limit, that needs to be obeyed when calculating the actual trajectory along the path.

Denote the local velocity limit for joint i at point p as $Jv^i_{local_p}$, and in the same way, the backwards velocity limit as $Jv^i_{back_p}$. Estimate the velocity on the interval from $(p-1)$ to p, separated by some initial Δs, as in (5.8) to receive $Jv^i_{est_p}$. The maximum allowed velocity for joint i at point p is then given by

$$Jv^i_{lim_p} = min\left[Jv^i_{local_p}, Jv^i_{back_p}, Jv^i_{est_p}\right] \tag{5.21}$$

and the minimum time one the interval is calculated similar to (5.9) as

$$\Delta t^i = \frac{\Delta J}{Jv^i_{lim_p}} + \frac{\left(Jv^i_{p-1} + Jv^i_{lim_p}\right)^2}{2Ja^i_{max}Jv^i_{lim_p}} \tag{5.22}$$

The actual time on the interval, Δt, is then determined by the "slowest" joint as in (5.10). By adjusting the Δs between points, an iterative search to find a Δt matching a desired sampling frequency ΔT is performed. This search can in some cases be time consuming since all the kinematic equations needs to be re-calculated for each new Δs. On the other hand, this method is in general more robust than Method 1, and no pre-calculation of the path is needed. With a satisfying Δt, the joint and TCP velocity at point p is calculated as in (5.11) and (5.12).

5.3 Dynamics

The methods described in Section 5.2 deals with all the kinematic constraints such as velocity and acceleration limits, that is assumed to be constant through all the path and depend on the robot model. But in reality this is not really true, since the dynamics of the robot could have a large effect on the performance. To improve time estimations, other constraints should be considered as well. Torque limits on the robot's actuators is one of the important issues to be taken into account. The dynamics model of the robot could be expressed as an equation of the motion of all links [19]:

$$\tau + J^T G_{ext} = M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + N(\mathbf{q}, \dot{\mathbf{q}}) \tag{5.23}$$

where

τ - vector of input torques (force), applied to the joint actuators,

$J^T G_{ext}$ - mapping of the external wrench G_{ext} to the joint space,

$M(\mathbf{q})$ - the manipulator inertia matrix,

$C(\mathbf{q}, \dot{\mathbf{q}})$ - accounts for centrifugal and Coriolis effects,

$N(\mathbf{q}, \dot{\mathbf{q}})$ - any other external force and torque, including gravity, friction, etc.

How to find each term is described in [20]. This dynamics model is highly coupled, which makes it difficult to use directly. Another problem is that the manipulators data, such as masses and inertia of links, is not available. Therefore, the upcoming section describes the simplified dynamics model used, where only the inertia component is considered.

5.3.1 Robot load

The methods described in Section 5.2 does not consider the actual robot load when calculating the trajectory. The resulting trajectory is identical regardless of the weight of the load, while in reality when the robot is handling something large, it will go slower due to torque limits on the actuators. The idea is to convert these dynamic constraints into previously used kinematic constraints, so the same trajectory calculation algorithm as described in Section 5.2 could be used. Assume torque limits τ_{lim} for each joint is known as well as acceleration limits α_0 for the non loaded case. The basic dependency between torque and angular acceleration could be written as following:

$$\tau = I\alpha, \tag{5.24}$$

where $I = I_0 + I_l$ is the total moment of inertia of the robot and load around the axis of rotation, and α is the angular acceleration. Then for a given pose (joint values), the moment of inertia I_0 of the robot without load could be estimated from torque and acceleration limits, which are known, as

$$I_0 = \frac{\tau_{lim}}{\alpha_0} \tag{5.25}$$

For the loaded case, acceleration limit for each joint could be estimated as

$$\alpha = \frac{\tau_{lim}}{I_0 + I_l} \tag{5.26}$$

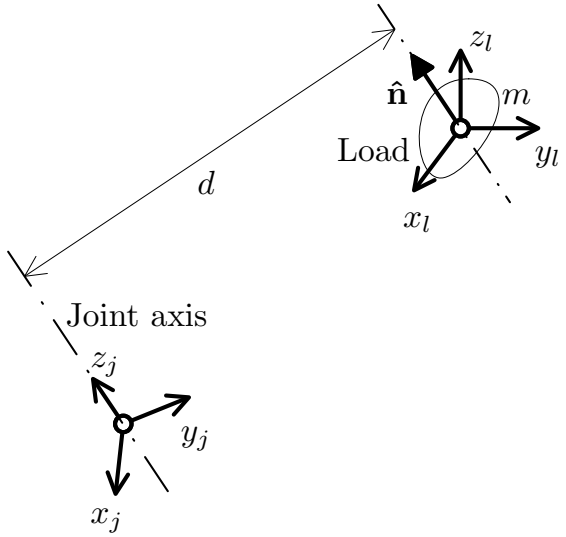


Figure 5.8: Load rotation around joint axis

where I_l is the moment of inertia of the load around the joint's rotation axis. Using the parallel axis theorem, I_l is defined as

$$I_l = I_{l0} + md^2 \tag{5.27}$$

where I_{l0} is moment of inertia of the load around the axis at its center of mass, m is the mass of the load and d is the distance between center of mass and the joint's axis of rotation (Figure 5.8).

The load's inertia is usually defined as a vector with components I_x, I_y and I_z, which form the diagonal tensor of inertia. In this case I_{l0} could be found as

$$I_{l0} = \hat{\mathbf{n}}^T \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \hat{\mathbf{n}} \tag{5.28}$$

where $\hat{\mathbf{n}}$ is a rotation axis vector in the center of mass coordinate system (see Figure 5.8).

The acceleration limits now depend on the pose of the robot (joint values), since the inertia of the load depends on the joint values. However, the inertia of the robot itself is assumed to be constant. The algorithms described in Section 5.2 could now be used with an addition of a subroutine to calculate the acceleration limits from joint values.

6

Results

Numerous tests has been performed in order to validate the behavior of the implemented VRC. Either *RobotStudio*, *Process Simulate*, or both, have been used to verify the correctness of the results. Tests evaluating the behavior of the VRC close to singularities can be seen in Appendix A.4. This chapter will start with a brief description of the robot model used for testing.

6.1 ABB's robot IRB 6640

IRB 6640 is a robot with six joints, mainly used for material handling, machine tending and spot welding [21]. Table 6.1 summarizes the physical attributes of the robot taken from the manufacturers specification. Note that the acceleration and torque limits are not stated in the specification, and have therefore been estimated from independent test cases in *RobotStudio* (see Appendix B.1 for more details).

Joint	Range	Velocity limit	Acceleration limit	Torque limit
#	deg	deg/s	deg/s^2	Nm
1	-170..170	100	(438)	(5400)
2	-65..85	90	(212)	(5000)
3	-180..70	90	(334)	(3800)
4	-300..300	170	(2405)	(1000)
5	-110..110	120	(1878)	(1200)
6	-360..360	190	(2536)	(1000)

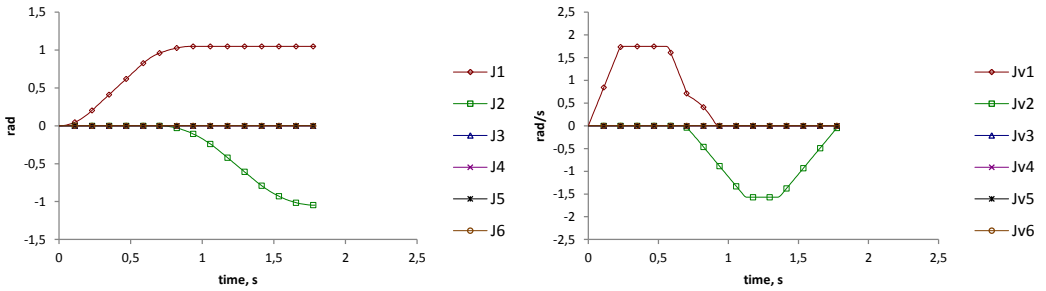
Table 6.1: Physical limits for ABB's robot IRB 6640

6.2 Validation tests in joint space

When the robot is moving in joint space, the targets to reach are specified in joint values, eliminating the need for inverse kinematics. The tests presented here has been performed on three different tracks, and both *RobotStudio* and *Process Simulate* has been used to evaluate the time optimality. Cycle times for each track are summarized in Table 6.2.

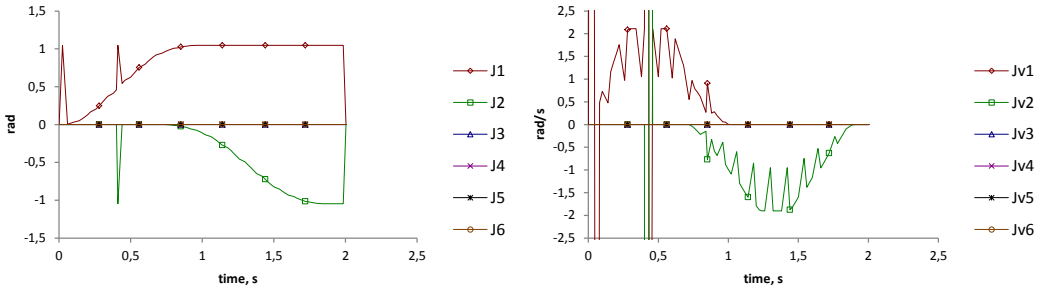
Track 1: Sharp turn

A sharp turn of the TCP can be achieved in joint space by rotating joint one, followed by a movement of joint two. This test has been performed with the zone radius, R_z, set to 200 mm and 0 mm. Figure 6.1 shows profiles of joint values and velocities as a function of time for the test with 200 mm zone radius (see Appendix A.1 for the case of 0 mm zone radius). In *RobotStudio* only the joint values over time are available to plot.



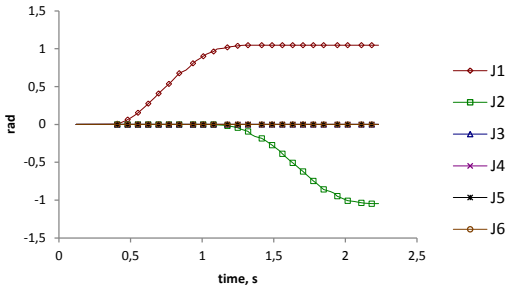
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

(d) *Process Simulate*, joint velocities

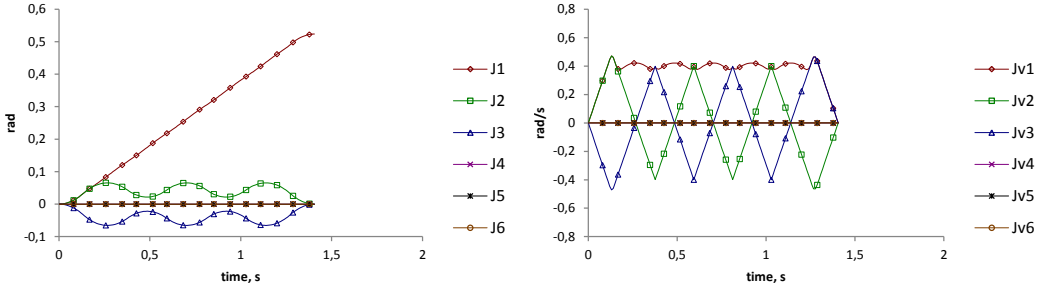


(e) *RobotStudio*, joint values

Figure 6.1: Joint values and velocities profiles for sharp turn test with $R_z = 200$ mm

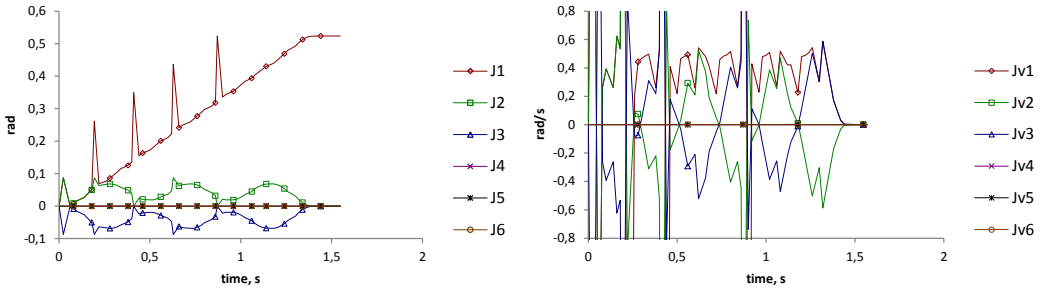
Track 2: Zigzag

If the zone radius is greater than half the segment length, the implemented VRC adjusts the zone radius to half the segment length. To verify how the commercial softwares handles this situation, a zigzag pattern with overlapping zones in the TCP trace was tested. Results are shown in Figure 6.2 (see Appendix A.1 for the case of non overlapping zones).



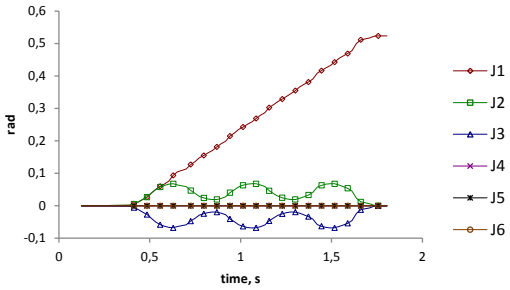
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

(d) *Process Simulate*, joint velocities

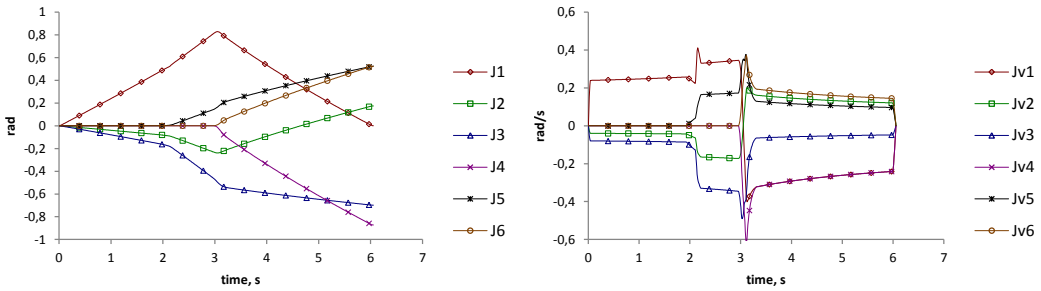


(e) *RobotStudio*, joint values

Figure 6.2: Joint values and velocities profiles for zigzag test with overlapping zones

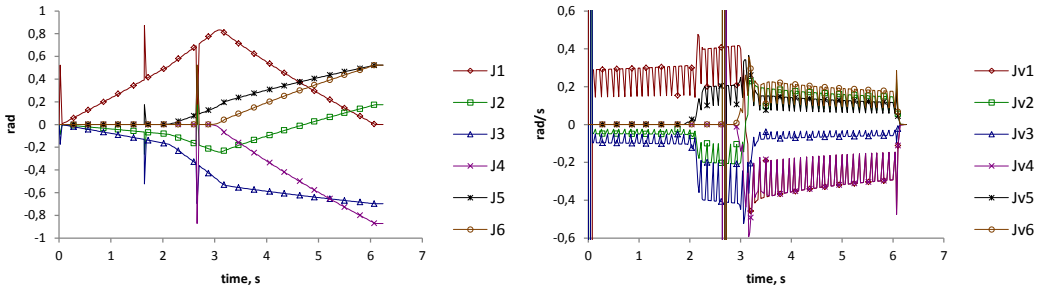
Track 3: A general track

With a strict limit on the maximum allowed TCP velocity, V_{max}, other uncertainties in the system are reduced (such as acceleration limits of the joints). Figure 6.3 demonstrates the velocity profile and joint values when the TCP velocity is limited (see Appendix A.1 for the case of unlimited V_{max}).



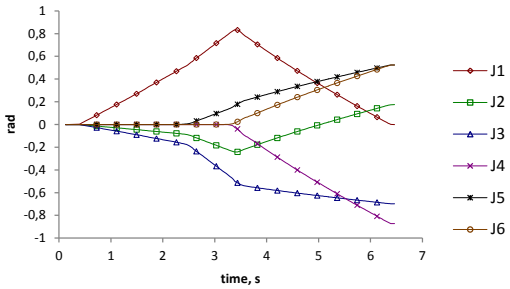
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

(d) *Process Simulate*, joint velocities



(e) *RobotStudio*, joint values

Figure 6.3: Joint values and velocities profiles for general track test with limited V_{max}

Summary

Table 6.2 summarizes the cycle times for each track type, using the three different simulation tools. Note that cycle time for *RobotStudio* was calculated from timestamped values, and internal simulation delay before actual start of the movement was removed (this delay can be seen in the graphs in Figure 6.1 - Figure 6.3).

Track type	*RobotStudio* *sec*	*Process Simulate* *sec*	VRC *sec*	Error value[1] %
Sharp turn, $R_z = 200\ mm$	1.85	1.98	1.79	3.2
Sharp turn, $R_z = 0\ mm$	1.94	2.25	1.92	1.0
Zigzag, overlapping zones	1.42	1.57	1.41	0.7
Zigzag, non overlapping zones	1.66	1.79	1.77	6.6
General track, limited V_{max}	6.07	6.26	6.06	0.2
General track, unlimited V_{max}	1.66	1.79	1.53	7.8

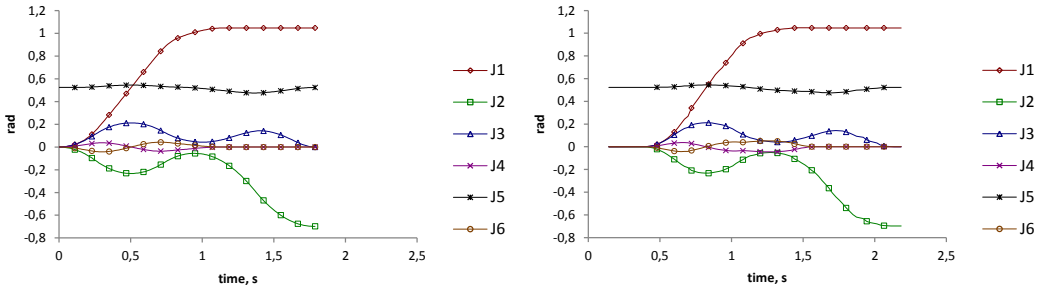
Table 6.2: Summary of cycle times for each track with target way points specified in joint space

6.3 Validation tests in Cartesian space

Similar test tracks as in the previous section are evaluated here, although the target way points are specified in Cartesian coordinates. The comparison has been made against *RobotStudio*, with added test cases on limiting the maximum TCP velocity (see Appendix A.2 for graphs on additional test cases).

Track 1: Sharp turn

Figure 6.4 presents the graphs of joint values for a sharp turn with way points specified in Cartesian space. The zone radius is 200 mm and without any limit on the maximum TCP velocity.



(a) VRC, joint values

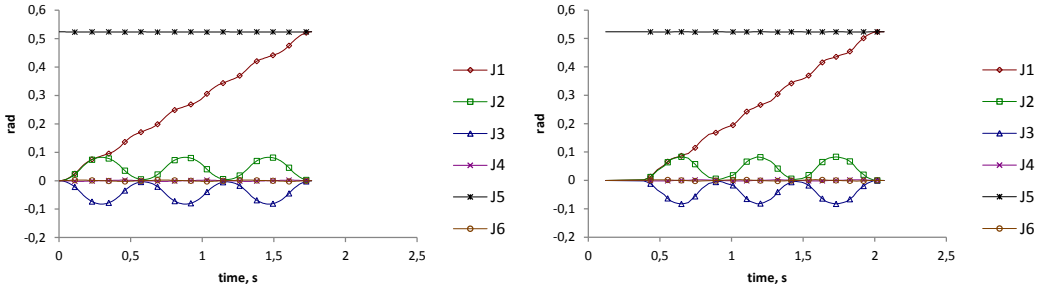
(b) *RobotStudio*, joint values

Figure 6.4: Joint values for a sharp turn with way points specified in Cartesian space, and $R_z = 200\ mm$

[1]The percentage difference between *RobotStudio* and the implemented VRC

Track 2: Zigzag

In Figure 6.5, the graphs for a zigzag track in Cartesian space is presented. The zones radius is 20 mm, and without any limit on the maximum TCP velocity.

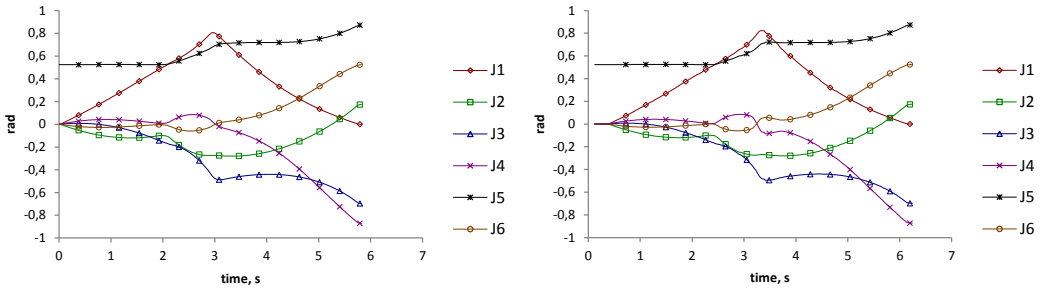


(a) VRC, joint values (b) *RobotStudio*, joint values

Figure 6.5: Joint values for a Zigzag track with way points specified in Cartesian space

Track 3: A general track

The last test case is a general track where each way point has a different zone radius. The graphs of the joint values over time is presented in Figure 6.6.



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure 6.6: Joint values for a general track with way points specified in Cartesian space, with a limited V_{max}

Summary

Table 6.3 shows the total cycle times for each track.

Track type, zone radius (mm), TCP velocity limit (mm/s)	*RobotStudio* sec	VRC sec	Error value %
Sharp turn, $R_z = 200$, $V = V_{max}$	1.78	1.79	0.6
Sharp turn, $R_z = 200$, $V = 100$	31.06	31.20	0.5
Sharp turn, $R_z = 0$, $V = V_{max}$	1.82	1.91	4.9
Sharp turn, $R_z = 0$, $V = 100$	32.09	32.63	1.7
Zigzag, $R_z = 200$, $V = V_{max}$	1.42	1.40	1.4
Zigzag, $R_z = 200$, $V = 100$	10.80	10.76	0.4
Zigzag, $R_z = 20$, $V = V_{max}$	1.68	1.76	4.7
Zigzag, $R_z = 20$, $V = 100$	11.38	11.41	0.3
Zigzag, $R_z = 0$, $V = V_{max}$	1.75	1.85	5.7
Zigzag, $R_z = 0$, $V = 100$	11.62	11.70	0.7
General track, limited V_{max}	5.88	5.82	1.0
General track, unlimited V_{max}	2.09	1.99	4.8

Table 6.3: Summary of cycle times for each track with target way points specified in Cartesian space

6.4 Zone path interpolation test

The path inside the zone could be derived in numerous ways. Different robot manufacturers might treat this differently, and exactly how they construct the corner path is unknown. This section will present the comparisons made against *RobotStudio*, and the results might not be applicable to other manufacturers and other robots.

To verify the zone path, simple test case in Cartesian space was constructed. The robot performs a 90 degree turn in Cartesian x-y plane, while the zone radius is varying. Figure 6.7 presents the TCP trace for such a test inside a 200 mm zone. As the graph shows, the maximum deviation is around 8 mm. Table 6.4 summarizes the approximate maximum deviations for different zones with different radius, when performing a 90 degree turn in Cartesian space.

Zone radius mm	Maximum error mm	Maximum error %
200	~ 8	4.0
100	~ 4	4.0
5	~ 0.2	4.0

Table 6.4: Maximum error in zone path interpolation compared against *RobotStudio*

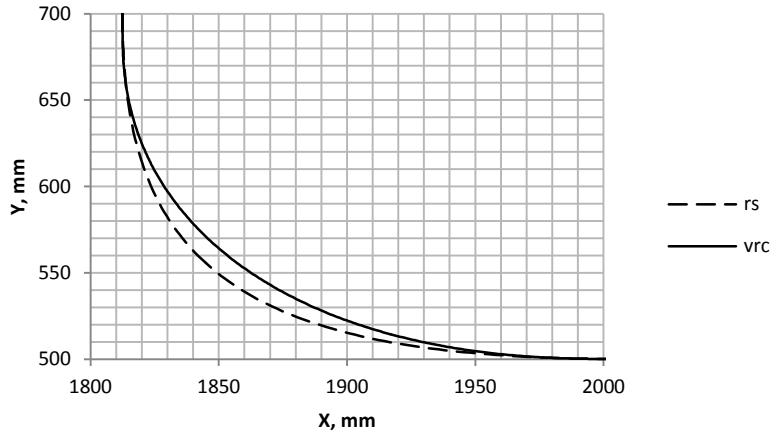


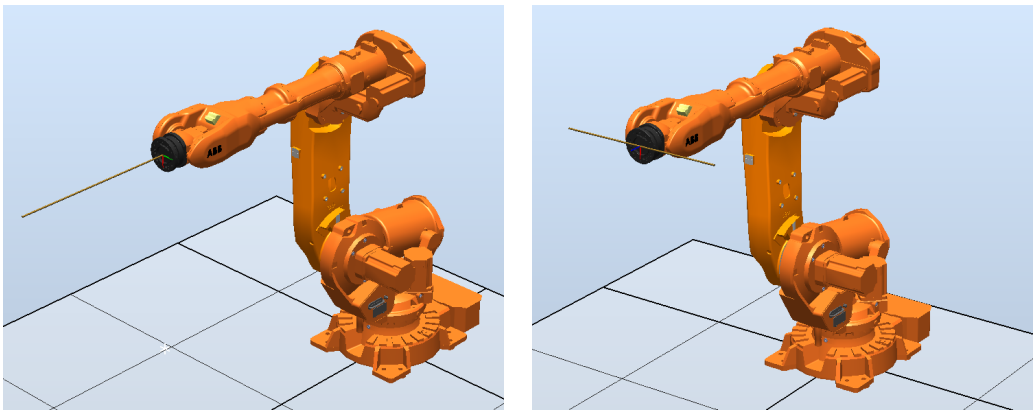
Figure 6.7: Comparison of the TCP trace inside a 200 mm zone against *RobotStudio* (rs)

6.5 Validation tests while carrying a tool

With a tool attached at the TCP of the robot, the effects of the dynamics in the system can be evaluated. In this section, a thin rod of length 1 m and a weight of 100 kg is attached in two different ways, while evaluating the time optimality for different tracks. The rod is either placed along or across the direction of the robot's arm, see Figure 6.8, changing the center of mass and inertia of the tool. Three different tests were performed with no load attached and two different load directions, as described above:

1. Joint 6 rotation.
2. General track in Cartesian space, same as in Section 6.3.
3. Joint 1 rotation in extreme pose, see Figure 6.9, where the inertia of the robot itself is large.

This gives totally 9 test cases, which cycle times are summarized in Table 6.5 and graphs comparing the joint values over time against *RobotStudio* can be seen in Appendix A.3. The results are quite accurate for tracks that puts the robot near the home position, while they are different in the extreme cases far away from the home position. This is due to the change in the robot's own inertia depending on the pose. Since the robot characteristics such as masses and inertia of links are unknown, it is difficult to implement the correct dynamic model. An approach to improve the model is described in Appendix B.2.



(a) (b)

Figure 6.8: Rod attached to robot along (a) and across (b) the direction of the robot's arm (screenshots are from ABB's *RobotStudio*)

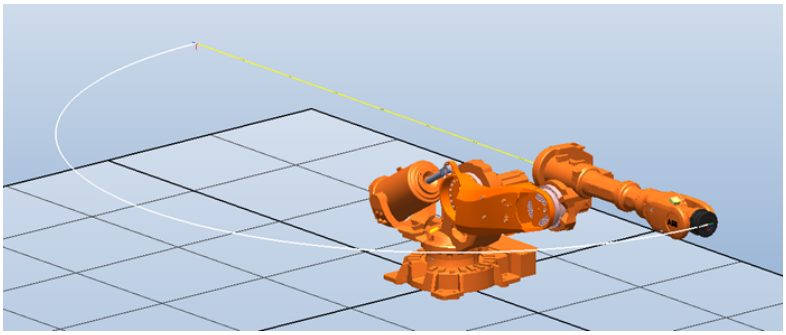


Figure 6.9: A pose with large inertia on the robot itself (screenshot is from ABB's *RobotStudio*)

Track type, load	*RobotStudio* *sec*	VRC *sec*	Error value %
Joint 6 rotation, no load	6.07	6.13	1.0
Joint 6 rotation, rod along	6.07	6.13	1.0
Joint 6 rotation, rod across	6.22	6.30	1.3
General track, no load	2.09	1.99	4.8
General track, rod along	2.35	2.43	3.4
General track, rod across	2.35	2.28	3.0
Joint 1 rotation, no load	15.14	12.52	17.3
Joint 1 rotation, rod along	18.05	14.79	18.1
Joint 1 rotation, rod across	17.28	14.19	17.9

Table 6.5: Summary of cycle times for each test case for different loads

7

Summary

This thesis has presented the theory behind the implementation of a software simulation tool, VRC, calculating the time optimal trajectory for a generic robot model. The VRC has been tested on a model of the industrial robot IRB 6640 from ABB, and the results has been compared against the robot controller in *RobotStudio* and *Process Simulate*. To demonstrate the trajectory planning a graphic interface has been developed, although implementation specific details regarding the interface has not been discussed in this thesis.

7.1 Conclusion

As the previous result chapter has shown, the implemented VRC performs well in most cases. However since several uncertainties exists in the robot model, the accuracy of the VRC varies in comparison to the robot controller in existing commercial software.

The tests of a general track in both joint and Cartesian space with a limit on the maximum allowed TCP velocity is very close to the robot controller (error value of 0.2 % in joint space and 1.0 % in Cartesian space, see Table 6.2 and Table 6.3) in *RobotStudio*. The probable reason for this is that a strict limit on the maximum allowed TCP velocity will be the dominating constraint in the trajectory planning. Other constraints on the joints, which are approximated and therefore uncertain, will never be reached.

One additional uncertainty is the zone path interpolation. As Section 6.4 has shown, the zone path in the implemented VRC deviates from the robot controller in *RobotStudio*. This deviation scales and is around 4 %, depending on the size of the zone radius. In theory, this means that the path generated from the VRC will be shorter than in reality, hence the time it takes to travel along this path will also be shorter if the velocity is constant (when moving on a velocity limit). This affects all test cases with a zone radius greater than 0 mm.

As for now, the dynamics of the robot are only approximated for one certain pose, the home pose. This implies that the current dynamic model is only valid whenever the robot carries a load close to this pose. The effects of this can be seen in Table 6.5. The cycle time estimation is quite accurate for the six first test cases where the robot is moving close to the home pose. However, the effect of the large inertia change of the robot itself is evident in the last 3 test cases.

The two trajectory calculation methods described in Section 5.2 has both been evaluated and shown to behave in the same way. The calculation time complexity is in general lower for Method 1, although Method 2 provides a more robust result with no need for pre-computation of key points.

The main goal of this thesis has been fulfilled. Extensive testing has proven the VRC to behave similar to a conventional robot controller. An augmented model of the dynamics would in general improve the cycle time estimations when the robot is carrying a load, among other future improvements listed in the upcoming section.

7.2 Future work

The implementation of a VRC is an extensive project. This section aims at listing a few additional features that could be added in future improvements.

- **Approximate the robot's own inertia** - Approximate the robot's inertia as a function of its pose. See Appendix B.2.
- **Jerk bounded trajectory calculations** - Additional constraint in the trajectory calculation with purpose of minimizing the jerk in each joint throughout the path. The jerk is affected by the changes in acceleration. This constraint would minimize the wear and tear in the mechanical parts of the robot model and increase the operational life time.
- **External forces** - A robot controller is in reality affected by numerous external forces such as gravity and friction that should be considered.

Bibliography

[1] A. Pfeiffer, B. Kádár, L. Monostori, Evaluating and improving production control systems by using emulation, Proc. of the 12 IASTED Int. Conf. on Applied Simulation and Modelling (2003) 261–267.

[2] Fraunhofer-Chalmers Research Centre for Industrial Mathematics, FCC, available 2011-04-26.
URL `http://www.fcc.chalmers.se`

[3] D. Forsman, Bangenerering för industrirobot med 6 frihetsgrader (in swedish), LiTH-ISY-EX-3519-2004, Master thesis, Tekniska Högskolan i Linköping (2004).

[4] M. Nyström, Bangenerering för industrirobot (in swedish), LiTH-ISY-EX-3357-2003, Master thesis, Tekniska Högskolan i Linköping (2003).

[5] M. Norrlöf, On path planning and optimization using splines, LiTH-ISY-R-2490, Technical report, Division of Automatic Control, Department of Electrical Engineering, Linköping University (2003).

[6] C. H. Lin, P. R. Chang, J. Y. S. Luh, Formulation and optimization of cubic polynomial joint trajectories for industrial robots, in: Transaction on Automatic Control, Vol. AC-28, No. 12, IEEE, 1983, pp. 1066–1074.

[7] M. Nyström, M. Norrlöf, Path generation for industrial robots, Technical report, Department of Electrical Engineering, Linköping University (2004).

[8] B. Siciliano, L. Sciavicco, L. Villani, Robotics: modelling, planning and control, Advanced textbooks in control and signal processing, Springer, 2009, Ch. 2, pp. 40–48.

[9] W. R. Sir Hamilton, On quaternions; or on a new system of imaginaries in algebra, Philosophical Magazine (1844), edited by D.R, Wilkins (2000). Available 2011-03-30.
URL `http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/OnQuat/OnQuat.pdf`

[10] K. Shoemake, Animating rotation with quaternion curves, in: Proceedings of ACM SIGGRAPH, The Singer Company, Link Flight Simulation Division, 1985, pp. 245–254.

[11] R. H. Taylor, Planning and execution of straight line manipulator trajectories, in: M. Brady, J. M. Hollerbach, T. L. Hohnson, T. Lozano-Perez, M. T. Mason (Eds.), Robot Motion: Planning and Control, The MIT Press, 1982.

[12] J. Funda, R. P. Paul, A comparison of transforms and quaternions in robotics, in: International Conference on Robotics and Automation, Vol. 2, IEEE, 1995, pp. 886–891.

[13] J. Gutmann, M. Fukuchi, M. Fujita, Real-time path planning for humanoid robot navigation, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2005, pp. 1232–1238.

[14] S. R. Buss, Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods, Technical report, Department of Mathematics, University of California, San Diego (2009).

[15] S. McKinley, M. Levine, Cubic spline interpolation (1998), available 2011-03-31. URL `http://online.redwoods.cc.ca.us/instruct/darnold/laproj/fall98/skymeg/proj.pdf`

[16] S. A. Bazaz, B. Tondu, On-line computing of a robotic manipulator joint trajectory with velocity and acceleration constraints, in: International Symposium on Assembly and Task Planning, IEEE, 1997.

[17] S. Macfarlane, E. A. Croft, Jerk-bounded manipulator trajectory planning: Design for real-time applications, in: Transaction on Robotics and Automation, Vol. 19, No. 1, IEEE, 2003, pp. 42–52.

[18] S. Macfarlane, On-line smooth trajectory planning for manipulators, Master thesis, The University of New Brunswick (1999).

[19] S. Y. Nof, Handbook of industrial robotics, 2nd Edition, Vol. 1, John Wiley and Sons, 1999, Ch. 6, pp. 94–98.

[20] V. D. Tourassis, C. P. Neuman, The inertial characteristics of dynamic robot models, in: Mechanism and Machine Theory, Vol. 20, No.1, 1985, pp. 41–52.

[21] ABB, IRB 6640/ IRB 6640ID Industrial Robot Specification (2010).

[22] ABB Robotics Products AB, S-72168, Västerås, Sweden, RAPID Reference Manual, Rev.1.

Appendix A

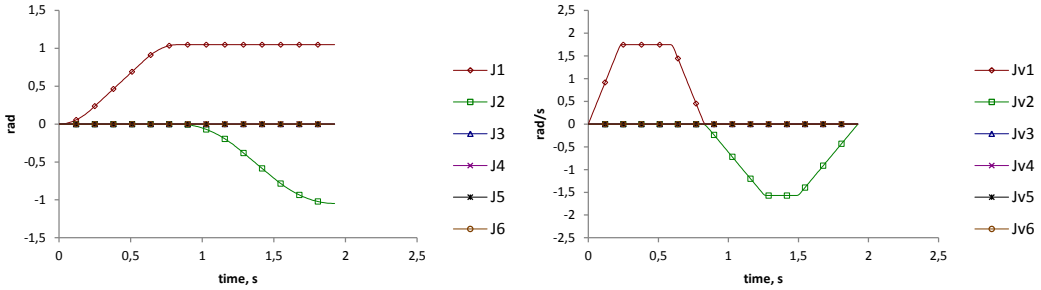
Additional tests

This appendix contains other test results that are not included in the main part of the report.

A.1 Graphs for additional tests in joint space

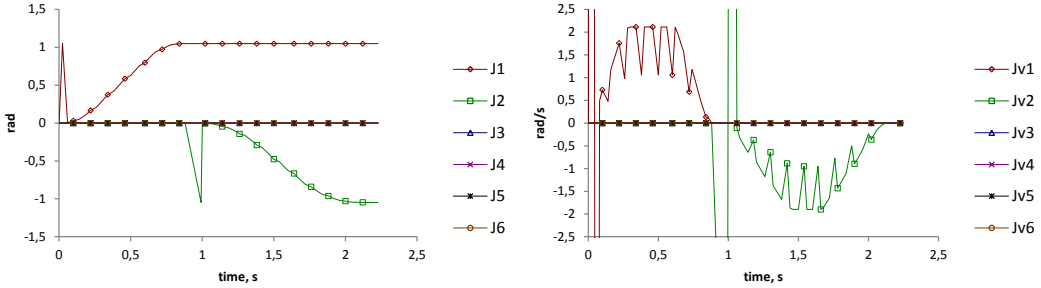
This section presents graphs for the test cases in joint space where only cycle times where presented in the result chapter.

Track 1: Sharp turn, 0 *mm* zone radius



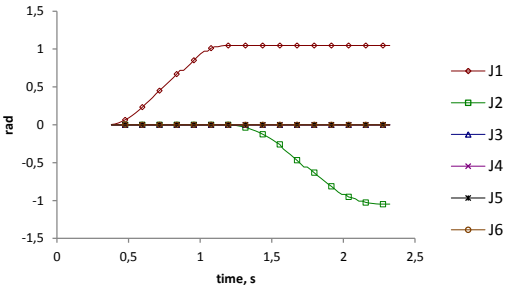
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

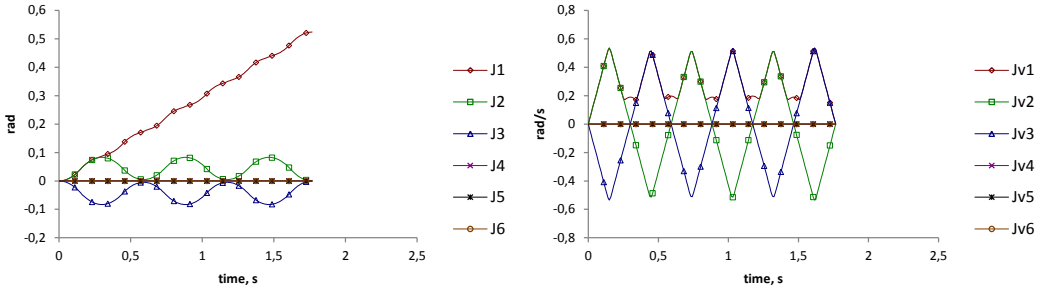
(d) *Process Simulate*, joint velocities



(e) *RobotStudio*, joint values

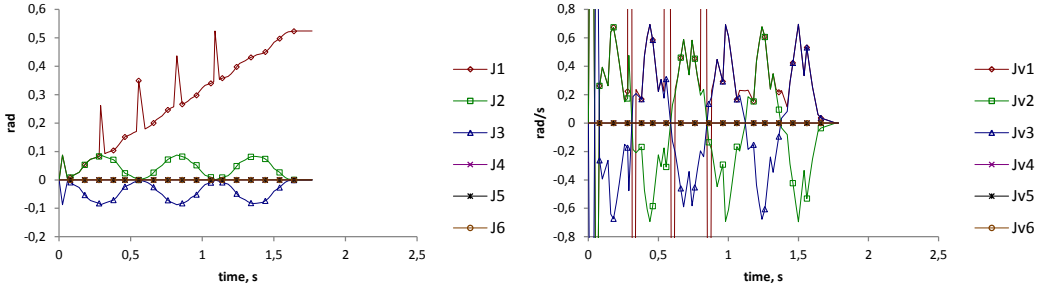
Figure A.1: Joint values and velocities profiles for sharp turn test with $R_z = 0$ *mm*

Track 2: Zigzag, non overlapping zones



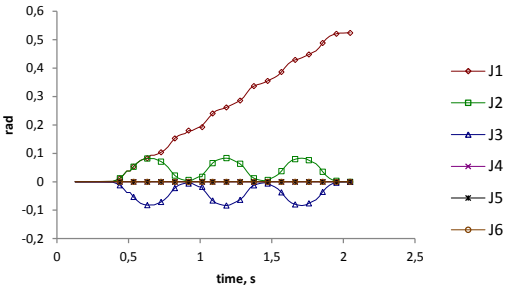
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

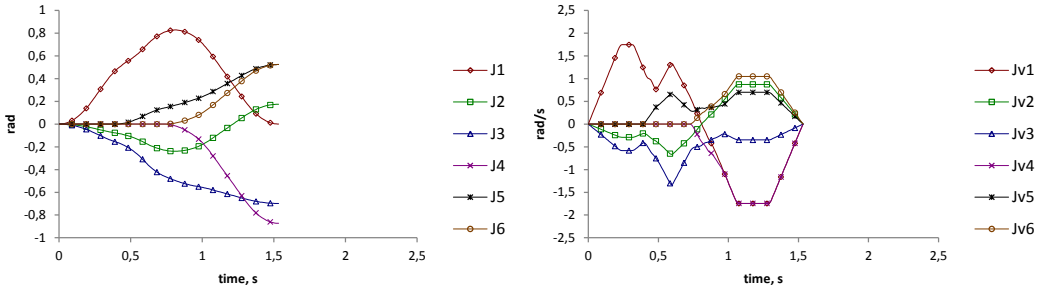
(d) *Process Simulate*, joint velocities



(e) *RobotStudio*, joint values

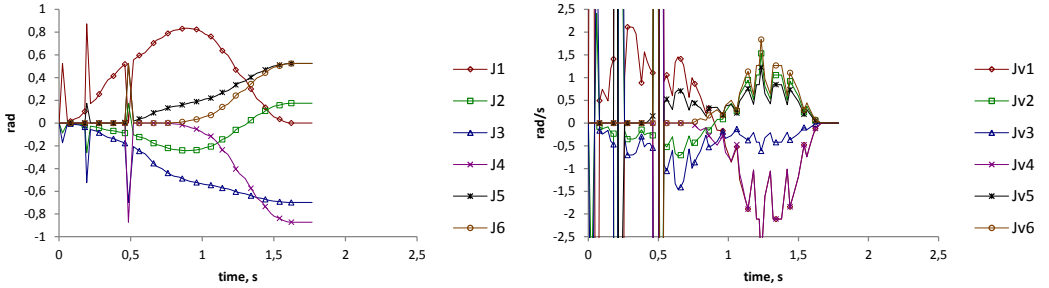
Figure A.2: Joint values and velocities profiles for zigzag test with non overlapping zones

Track 3: A general track, unlimited V_{max}



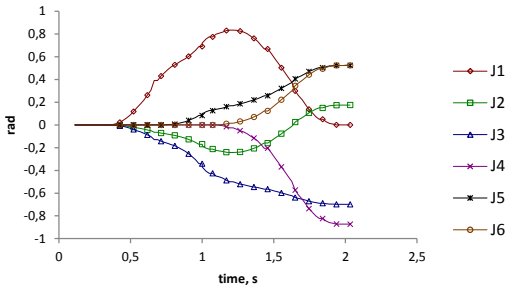
(a) VRC, joint values

(b) VRC, joint velocities



(c) *Process Simulate*, joint values

(d) *Process Simulate*, joint velocities



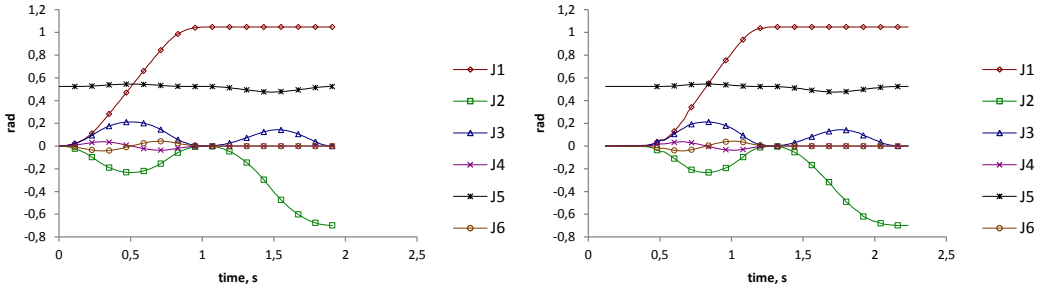
(e) *RobotStudio*, joint values

Figure A.3: Joint values and velocities profiles for general track test with unlimited V_{max}

A.2 Graphs for additional tests in Cartesian space

This section presents graphs for additional test cases in Cartesian space where only cycle times where presented in the result chapter.

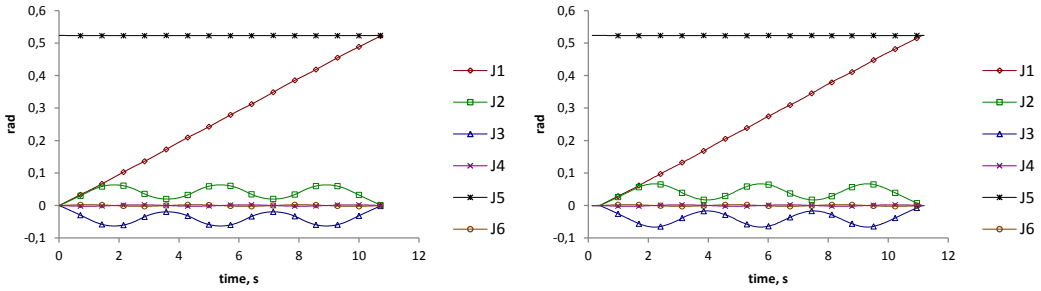
Track 1: Sharp turn, 0 mm zone radius, unlimited V_{max}



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.4: Joint values for a sharp turn with way points specified in Cartesian space and $R_z = 0$ mm, unlimited V_{max}

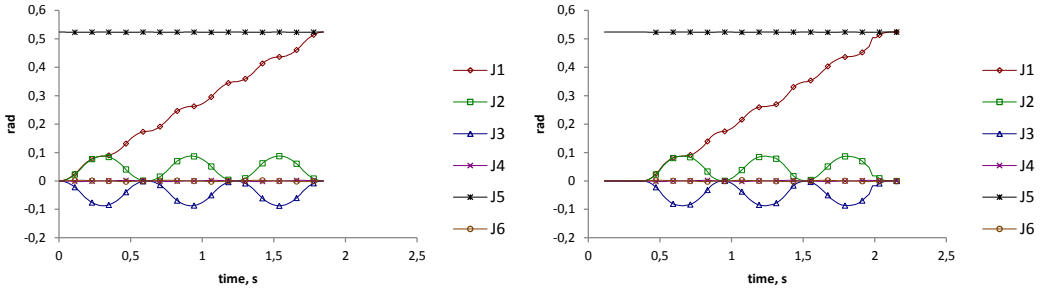
Track 2: Zigzag, 200 mm zone radius, limited V_{max}



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.5: Joint values for a Zigzag track with way points specified in Cartesian space and $R_z = 200$ mm, limited V_{max}

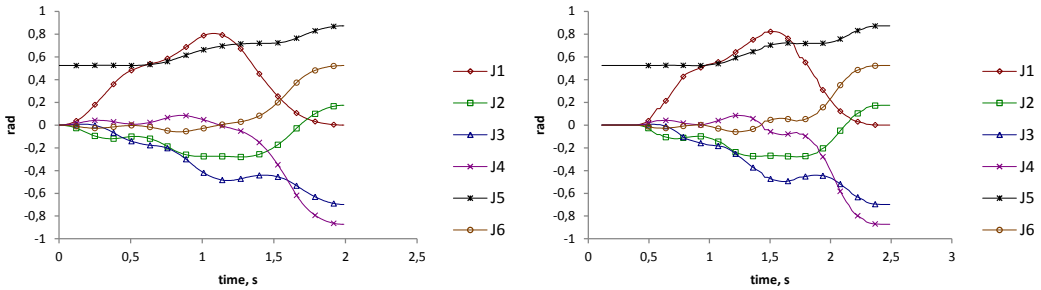
Track 2: Zigzag, 0 mm zone radius, unlimited V_{max}



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.6: Joint values for a Zigzag track with way points specified in Cartesian space and $R_z = 0\ mm$, unlimited V_{max}

Track 3: A general track, unlimited V_{max}

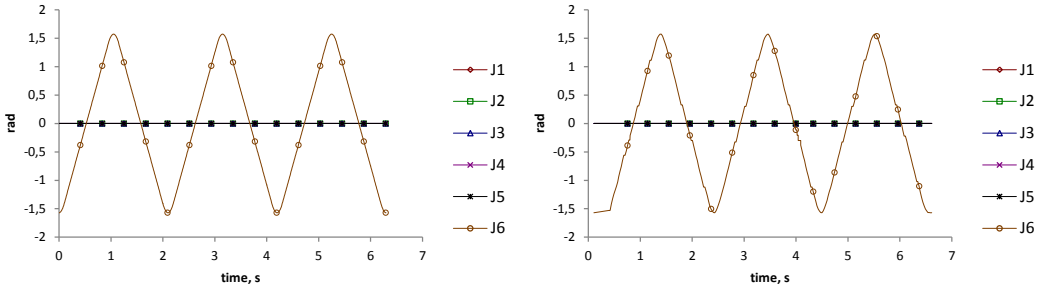


(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.7: Joint values for a general track with way points specified in Cartesian space, unlimited V_{max}

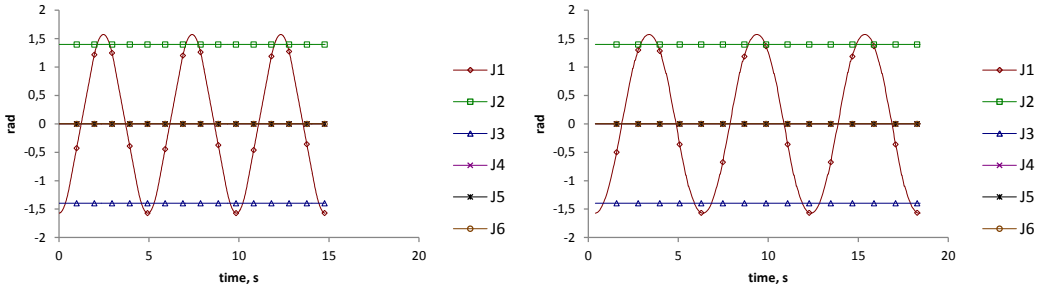
A.3 Graphs for movement with a tool

This section presents the graphs of the joint values over time from two tests in Section 6.5.



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.8: Joint values for a rotation of joint 6, with the rod attached across the direction of the robot's arm



(a) VRC, joint values (b) *RobotStudio*, joint values

Figure A.9: Joint values for a rotation of joint 1, with the rod attached along the direction of the robot's arm

A.4 Tests of movement close to singularities

The IRB 6640 has a singularity point where the joint value $J_i = 0$, and i = 1,2,..,6. In more detail, this singularity point will always appear whenever joint four and six are aligned collinear (i.e. joint 5 has value 0). To test the behavior close to this singularity, a track moving with a varied distance above the singularity has been evaluated against *RobotStudio*. Table A.1 summarizes the cycle times for an increasing distance above the singularity point.

Another singularity point occurs when the robot's wrist (joint 5) lies on the first joint axis. Table A.2 presents the cycle times when moving away from this singularity.

Distance	*RobotStudio*	VRC	Error value
mm	*sec*	*sec*	%
0	error	19.07	-
10	error	9.86	-
25	9.02	8.68	3.8
50	8.26	7.83	5.2
100	7.15	6.86	4.1
200	5.74	5.61	2.3
500	3.89	3.87	0.5

Table A.1: Summary of cycle times moving above the singularity at $J_i = 0$, and i = 1,2,..,6

Distance	*RobotStudio*	VRC	Error value
mm	*sec*	*sec*	%
0	error	error	-
10	13.42	14.38	7.2
25	11.88	12.51	5.3
50	10.85	10.84	0.1
75	10.06	9.79	2.7
100	9.34	8.98	7.1
150	8.09	7.75	4.2
200	7.08	6.78	4.2

Table A.2: Summary of cycle times moving above the singularity where the fifth joint are on the first joint axis

Appendix B

Robot data estimation

This appendix describes ideas and routines to estimate necessary robot specific data, which is unknown.

B.1 Robot limits

This section describes the routines to obtain acceleration and torque limits from experimental data. It is necessary, since the robot specification lacks of this information.

B.1.1 Joint acceleration limits

The acceleration limit for each joint could be estimated using the following test routine: rotate the joint $\pm 10°$ several times to get timestamped joint values. Then use the obtained data to extract the acceleration limit. Some assumptions were made in this case:

1. When only one joint is moving, it accelerates on its limit.
2. Acceleration and deceleration have the same value.
3. The joint does not reach its velocity limit during the movement.

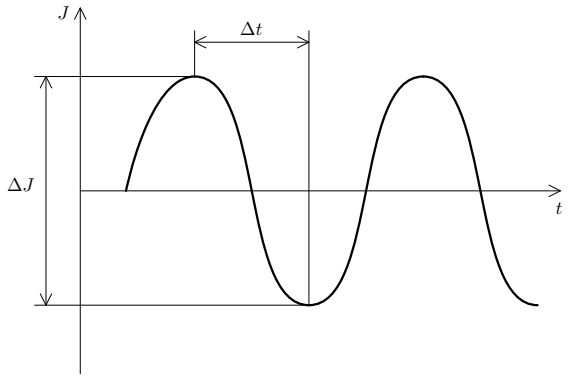


Figure B.1: Timestamped joint values

Then the obtained data could be approximated as set of parabolic curves, see Figure B.1. The equation of each parabola could be written as $J = k \cdot t^2$. Kinematic equation for this

part of movement is $J = \frac{\alpha \cdot t^2}{2}$, and $J = \frac{\Delta J}{2}$, $t = \frac{\Delta t}{2}$, so the formula to get acceleration limit is

$$\alpha_{lim} = 4\frac{\Delta J}{\Delta t}$$

It is also possible to use regression analysis to get the equation of a parabola, which fits the timestamped values, but it is more complicated and gives approximately the same result. Using several samples from timestamped data, the average α_{lim} is calculated to improve accuracy.

B.1.2 Joint torque limits

In fact, joint acceleration limits are not constant, they very much depend on the attached load and the robot's own inertia. So instead torque limits could be considered as actual physical limits. Since the masses of the robot's links are unknown, it is difficult to calculate the own moments of inertia. Instead several tests with different loads could be performed on the robot to find the torque limits. Using the procedure described in Section B.1.1, acceleration limits for that tests could be obtained. Since the load inertia is known, one could use two equal tests, but with different loads to obtain the torque limit:

$$\begin{cases} \tau_{lim} = I_0 \cdot \alpha_{lim,1} \\ \tau_{lim} = (I_0 + I_{load}) \cdot \alpha_{lim,2} \end{cases}$$

Torque limit is then:

$$\tau_{lim} = \frac{I_{load} \cdot \alpha_{lim,2}}{1 - \frac{\alpha_{lim,2}}{\alpha_{lim,1}}}$$

As before, several runs with different loads could be done, then the average torque limit could be calculated to improve accuracy.

B.2 Robot inertia

This section describes the idea of further improvement of the dynamic model of the robot by estimating links masses and inertia from experimental results.

In the trajectory calculation algorithm, the robot's own inertia for all links is assumed to be constant, despite of the robot's pose. This assumption is valid when a track puts the robot close to its home pose, but it gives quite a large error for joints 1-3 otherwise (see for example Table 6.5, last three cases). Therefore, calculation of the robot inertia as a function of its pose is desired. Since no information about link masses, inertia and center of mass positions is available, some estimation of these parameters is needed.

There are two possibilities to obtain the robot's inertia from experimental results:

1. Divide the configuration space into several regions and estimate the robot's whole inertia in every region by performing acceleration limit test (see Section B.1.1).

2. Estimate masses and inertia of separate links and calculate the whole robot inertia.

The first method has a trade-off between accuracy and number of regions. Since the mass and inertia of link 5 and 6 are quite small with respect to other links, and the value of joint 1 has no effect on the inertia, only joint 2 and 3 will affect the total inertia of the robot. The configuration space is therefore two-dimensional. Total number of regions $N = N_2 \cdot N_3$, where N_2 and N_3 in number of intervals to divide joint 2 and 3 range into respectively. Even if one takes 10 intervals for each joint, the total number of regions will be 100. That is rather big quantity, since each region will require to perform at least 1 acceleration limit test, which is time consuming.

The second approach seems to be more accurate and will probably require less number of tests. It is possible to simplify the robot model, since not all the data is necessary to obtain inertia around the joint axes. Link 1 could be represented by only one inertia component around joint axis 1. Links 3, 4, 5 and 6 could be merged together. Inertia of all the links except the first one could be neglected. Resulting model is shown in Figure B.2. Using this model, the robot's own inertia around the joint axes could be calculated

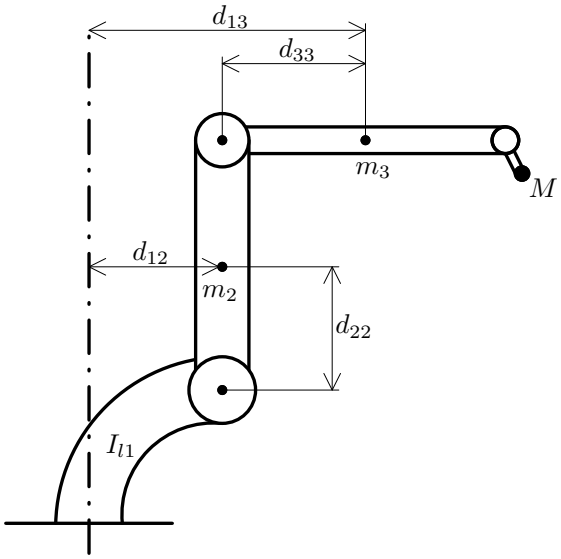


Figure B.2: Simplified robot model

as following:

$$I_1 = I_{l1} + m_2 \cdot d_{12}^2 + m_3 \cdot d_{13}^2 + M \cdot d_1^2$$
$$I_2 = m_2 \cdot d_{22}^2 + m_3 \cdot d_{23}^2 + M \cdot d_2^2$$
$$I_3 = m_3 \cdot d_{33}^2 + M \cdot d_3^2$$

where

I_i - robot's moment of inertia around joint axis i,

I_{l1} - inertia of the link 1 around joint 1 axis,

m_i - mass of the link i,

d_{ij} - distance from i-th link's center of mass to joint axis j,

M - mass of the load/tool,

d_i - distance from load/tool center of mass to joint axis i

In fact, the link's center of mass positions are unknown. But to get an approximate solution the links could be treated as solid cylinders, to enable the possibility to calculate the distances d_{ij}. Then, since the load/tool mass is known, the link parameters are calculated as following:

$$m_3 = \frac{I_3 - M \cdot d_3^2}{d_{33}^2}$$
$$m_2 = \frac{I_2 - M \cdot d_2^2 - m_3 \cdot d_{23}^2}{d_{22}^2}$$
$$I_{l1} = I_1 - m_2 \cdot d_{12}^2 - m_3 \cdot d_{13}^2 - M \cdot d_1^2$$

The robot inertia I_i is obtained from acceleration tests, as described in Section 5.3.1, equation 5.25. Furthermore, several test could be done to improve accuracy by averaging the results.

Note: A more complex robot model with all 6 links and inertia could also be used, but the results accuracy would still be limited by acceleration limit tests and robot torque limit calculation, which in fact is not very accurate.

Appendix C

RAPID code for test cases

This chapter provides the RAPID code for all test cases presented in the result section. The idea is to provide the possibility to recreate the tests in *RobotStudio* to get a better overview of the outline of the tracks. All tracks start in the home position of the robot. For more information on each command, see [22].

C.1 Validation tests in joint space

Track 1: Sharp turn

Sharp turn in joint space, both cases with 200 and 0 mm zone radius.

```
CONST jointtarget sharpturnJ_1:=[[60,0,0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget sharpturnJ_2:=[[60,-60,0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC sharpturnJ_200()
  CONST zonedata zon:=z200;
  MoveAbsJ sharpturnJ_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ sharpturnJ_2,vMax,zon,tool0\WObj:=wobj0;
ENDPROC

PROC sharpturnJ_0()
  CONST zonedata zon:=z0;
  MoveAbsJ sharpturnJ_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ sharpturnJ,vMax,zon,tool0\WObj:=wobj0;
ENDPROC
```

Track 2: Zigzag

Zigzag track in joint space, both with and without overlapping zones.

```
CONST jointtarget zigzagJ_1:=[[5,5,-5,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget zigzagJ_2:=[[10,0,0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget zigzagJ_3:=[[15,5,-5,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget zigzagJ_4:=[[20,0,0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget zigzagJ_5:=[[25,5,-5,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget zigzagJ_6:=[[30,0,0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC zigzagJ_overlap()
  CONST zonedata zon:=z200;
  MoveAbsJ zigzagJ_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_2,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_3,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_4,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_5,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_6,vMax,zon,tool0\WObj:=wobj0;
ENDPROC
```

```
PROC zigzagJ_nooverlap()
  CONST zonedata zon:=z20;
  MoveAbsJ zigzagJ_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_2,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_3,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_4,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_5,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagJ_6,vMax,zon,tool0\WObj:=wobj0;
ENDPROC
```

Track 3: General track

A general track in joint space, both with and without a limit on the V_{max}.

```
CONST jointtarget genJ_1:=[[30,-5,-10,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget genJ_2:=[[50,-15,-30,0,10,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget genJ_3:=[[0,10,-40,-50,30,30],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC genJ_lim()
  MoveAbsJ genJ_1,v100,z100,tool0\WObj:=wobj0;
  MoveAbsJ genJ_2,v100,z200,tool0\WObj:=wobj0;
  MoveAbsJ genJ_3,v100,fine,tool0\WObj:=wobj0;
ENDPROC

PROC genJ_unlim()
  MoveAbsJ genJ_1,vMax,z100,tool0\WObj:=wobj0;
  MoveAbsJ genJ_2,vMax,z200,tool0\WObj:=wobj0;
  MoveAbsJ genJ_3,vMax,fine,tool0\WObj:=wobj0;
ENDPROC
```

C.2 Validation tests in Cartesian space

Track 1: Sharp turn

A sharp turn in Cartesian space, with 200 and 0 mm zone radius and with and without a limit on the V_{max}.

```
CONST robtarget sharpturnL_1:=[[942.852540378443,1633.06850398085,1955.00000000001],
                               [0.433012701892221,-0.433012701892219,0.75,0.249999999999997],
                               [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget sharpturnL_2:=[[382.062098616613,661.751010984942,2686.51804412877],
                               [0.663413938532656,-0.321393815744621,0.556670399376687,0.383022228884307],
                               [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC sharpturnL_200_nolimit()
  CONST zonedata zon:=z200;
  MoveAbsJ sharpturnL_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ sharpturnL_2,vMax,zon,tool0\WObj:=wobj0;
ENDPROC

PROC sharpturnL_200_limit()
  CONST zonedata zon:=z200;
  MoveAbsJ sharpturnL_1,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ sharpturnL_2,v100,zon,tool0\WObj:=wobj0;
ENDPROC

PROC sharpturnL_0_nolimit()
  CONST zonedata zon:=z0;
  MoveAbsJ sharpturnL_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ sharpturnL,vMax,zon,tool0\WObj:=wobj0;
ENDPROC

PROC sharpturnL_0_limit()
  CONST zonedata zon:=z0;
  MoveAbsJ sharpturnL_1,v100,zon,tool0\WObj:=wobj0;
```

```
  MoveAbsJ sharpturnL,v100,zon,tool0\WObj:=wobj0;
ENDPROC
```

Track 2: Zigzag

Zigzag track in Cartesian space, both with and without overlapping zones, and with and without limit on the V_{max}.

```
CONST robtarget zigzagL_1:=[[1971.86529911065,172.515859672336,1950.90930044863],
                           [0.499524110790929,-0.0377754975558949,0.865201139495554,0.021809693682665],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget zigzagL_2:=[[1857.05698342389,327.449250890703,1955.00000000001],
                           [0.498097349045873,-0.0754790873051733,0.862729915662821,0.0435778713738261],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget zigzagL_3:=[[1911.95116980916,512.305771918103,1950.90930044863],
                           [0.495722430686906,-0.113038998321815,0.858616436401261,0.0652630961100228],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget zigzagL_4:=[[1771.98314936574,644.94912199041,1955.00000000001],
                           [0.492403876506105,-0.150383733180435,0.852868531952443,0.0868240888334622],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget zigzagL_5:=[[1793.94337170699,836.529532523368,1950.90930044863],
                           [0.488148003559968,-0.187442204055694,0.845497143779175,0.108219806969049],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget zigzagL_6:=[[1633.06850398085,942.85254037844,1955.00000000001],
                           [0.482962913144535,-0.224143868042013,0.836516303737808,0.129409522551258],
                           [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC zigzagL_overlap_nolim()
  CONST zonedata zon:=z200;
  MoveAbsJ zigzagL_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_2,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_3,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_4,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_5,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_6,vMax,zon,tool0\WObj:=wobj0;
ENDPROC

PROC zigzagL_overlap_lim()
  CONST zonedata zon:=z200;
  MoveAbsJ zigzagL_1,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_2,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_3,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_4,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_5,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_6,v100,zon,tool0\WObj:=wobj0;
ENDPROC

PROC zigzagL_nooverlap_nolim()
  CONST zonedata zon:=z20;
  MoveAbsJ zigzagL_1,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_2,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_3,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_4,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_5,vMax,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_6,vMax,zon,tool0\WObj:=wobj0;
ENDPROC

PROC zigzagL_nooverlap_lim()
  CONST zonedata zon:=z20;
  MoveAbsJ zigzagL_1,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_2,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_3,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_4,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_5,v100,zon,tool0\WObj:=wobj0;
  MoveAbsJ zigzagL_6,v100,zon,tool0\WObj:=wobj0;
ENDPROC
```

Track 3: General track

A general track in Cartesian space, both with and without a limit on the V_{max}.

```
CONST robtarget genL_1:=[[1483.31154979883,856.390322568438,2352.7361741764],
                         [0.588018380745341,-0.205334955110927,0.766320485043854,0.157559050255368],
                         [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget genL_2:=[[696.931732983846,830.57089657641,2961.86896084986],
                         [0.668200191556659,-0.285516759191569,0.61229266600778,0.311586866503505],[0,0,0,0],
                         [9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget genL_3:=[[1773.18663325767,-117.364817766692,2687.11532301843],
                         [0.611089206343482,0.15656972867144,0.726640317234206,-0.272120839562165],
                         [0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC genL_nolim()
MoveL genL_1,vMax,z100,tool0\WObj:=wobj0;
MoveL genL_2,vMax,z200,tool0\WObj:=wobj0;
MoveL genL_3,vMax,fine,tool0\WObj:=wobj0;
ENDPROC

PROC genL_lim()
MoveL genL_1,v100,z100,tool0\WObj:=wobj0;
MoveL genL_2,v100,z200,tool0\WObj:=wobj0;
MoveL genL_3,v100,fine,tool0\WObj:=wobj0;
ENDPROC
```