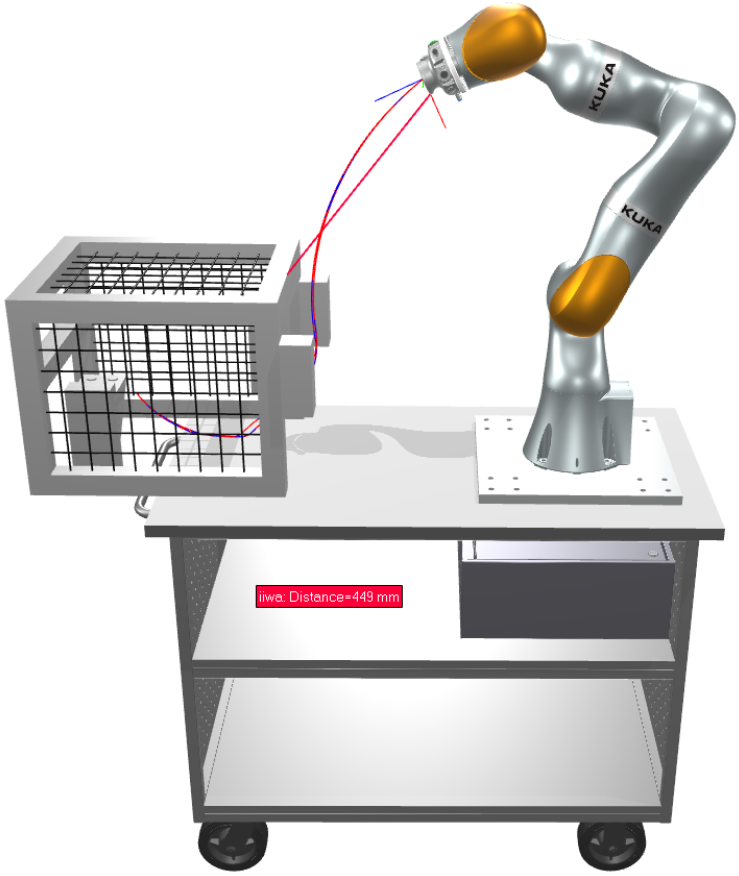
CHALMERS
UNIVERSITY OF TECHNOLOGY



Including a Collaborative Robot in Digital Twin Manufacturing Systems

Master's thesis in Master Programme in Systems, Control and Mechatronics

CHRISTIAN LARSEN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Including a Collaborative Robot in Digital Twin Manufacturing Systems

CHRISTIAN LARSEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2019

Including Collaborative Robots in Digital Twin Manufacturing Systems
CHRISTIAN LARSEN

Supervisor: Johan Carlson, Fraunhofer Chalmers Centre
Examiner: Bengt Lennartsson, Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: 3D rendition of a KUKA LBR iiwa R800.

Typeset in LaTeX
Gothenburg, Sweden 2018

Abstract

Great potential lie in unlocking the capabilities of industrial robots through simulation in terms of cost saving, productivity, and flexibility. The concept of digital twin has emerged in recent year which aims to further improve the impact of simulations. A digital twin is an virtual counter part of a physical system that is capable of accurately simulating the physical system using data from the real system. The knowledge gained from the simulation can thus be used for real-time optimization and decision making.

In this thesis it is investigated how one can create the needed capabilities to allow a KUKA iiwa robot within a digital twin manufacturing system. This includes investigating and developing the necessary capabilities to be able to simulate the motion of the iiwa robot depending on given motion commands as well as creating a interface to a real world iiwa robot adhering to the concept of digital twins.

The developed interface with the real iiwa robot is used to evaluate the developed simulation capability in terms of how well it can capture the geometrical path of the motion as well as cycle time. The results shows that the utilized method is capable of accurately capturing the geometrical path of the robot as well as estimating the cycle-time within 15% depending on the given motion commands.

Acknowledgements

It has been a great experience working at the Fraunhofer-Chalmers Centre (FCC). This thesis would not have been possible without the help from my colleagues at FCC. I express my gratitude towards Johan Carlson, director at FCC, for giving me the opportunity to do my thesis and all the valuable advice during the process.

I would also like to thank my colleagues in the Geometry and Motion Planning group, specifically Daniel Gleeson, Robert Bohlin, Staffan Björkenstam, Jonas Kressin, Simon Vajedi and Domenico Spensieri for all the valuable input and advice.

Great appreciation goes to Kristofer Bengtson at Chalmers for all the help and advice regarding any robotics and networking. I'm very grateful that he allowed me to use the robot for experiments throughout the thesis.

Gothenburg, March 2019
Christian Larsen

Contents

1 Introduction

The concept of digital twins, which aims to improve the impact and fidelity of simulation capabilities, has gained traction in recent years [1]. A digital twin is a digital model of real-world physical systems. A digital twin of a physical system is utilized to increase the performance of the product throughout its life cycle. This is done through the employment of data driven simulation during the design and planning phase as well as while the system is in operation. This process enables the user to optimize the performance during the whole lifetime of the product.

The conceptual definition of a digital twin is first introduced by the National Aeronautics and Space Administration (NASA) [2]. NASA defined a digital twin as a digital representation of a physical flying aircraft that uses the best available physical models, sensor data, and flight history. The goal of this process was to continuously update the digital representation with data from the real aircraft, which are then used in simulations of the digital twin. NASA contends that this process reduced the errors by simulating future outcomes based on the updated digital representation. If the simulations determine it necessary, the mission profile could be changed to increase the mission success.

In recent years, the digital twin concept has been used in the field of manufacturing engineering[3]. In this context, the digital twin is described as a virtual counterpart of the manufacturing system. This digital representation contains information, such as functional descriptions and computer-aided design (CAD) models, that can be useful throughout the lifecycle – from product design to production execution and production intelligence [4, 5]. One important aspect of the digital twin concept is the possibility to synchronize the state of the digital representations with the measured state of the real-world counterparts [3]. This feature allows users to perform real-time optimization and decision-making based on simulation outcomes of the digital representation given the initial conditions of the real-world counterpart. As such, digital twin has increasingly gained traction as devices are fitted with more and more sensors, along with internet connections following the lines of *The Internet of Things* [6].

For example, a production cell proposed by Söderberg et al. [7] and Bohlin et al. [8] employs a digital twin for geometry assurance. The authors outlined in these works the composition of a production cell in which the quality of the incoming parts is improved by employing the ideas of digital twin. They proposed a cell consisting of spot welding robots, handling robots, fixtures, and cameras working together and they used a digital twin to decrease the geometric quality without tightening the tolerances by adjusting the welding process. This process is to be performed by simulating the quality of the products and by learning based on the outcome of the digital representation

of the production cell.

In such a system, the fidelity of the simulation if affected by how accurately the motions of the robot can be described. However, for third-party simulation software, accurate simulations of industrial robots are only available through the manufacturers' proprietary offline programming software or through the standardized Realistic Robot Simulation (RRS) interface [9]. The robot examined in this thesis is the KUKA LBR iiwa 7 [10]. Currently, the manufacturer of this robot has not provided simulation capabilities for simulating motion programs. Thus, great potential exists in unlocking simulation capabilities to allow advanced optimization algorithms to be independently developed.

1.1 Purpose and goal

In this thesis, the goal is to create the needed capabilities to allow the iiwa robot to work within a digital twin of a production cell. Given the definition of a digital twin manufacturing system, two main functionalities are needed, a virtual representation of the robot and an interface to the real robot allowing synchronization and re-planning.

Virtual representation: A digital representation that shares the functionality of the real robot is needed. The functionality of an industrial robot is to perform tasks such as arc welding and glue joining [11]. To enable the simulation of such tasks, a method for simulating how the robot moves to accomplish aforesaid processes needs to be determined. This includes how the motions of the robot can be predicted based on the given commands.

Physical robot interface: The virtual representation should also allow for synchronization with the physical robot. This process includes updating the joint values of the robot based on the position of the real robot as well as updating the current state of input and output (IO) signals and available sensor data when needed. The ability to synchronize enables the execution of possible optimization algorithms based on the virtual representation with the initial state from the real robot.

The state of the robot, including its joint position, IO data, and sensor data needs to be made available to synchronize with the virtual representation. A network connection will be utilized for this process. The prospect to control the robot from an external application needs to be possible to allow for adaptation and flexibility during production.

In summary, the goal of this thesis is to attain the ability to employ a KUKA iiwa robot within a digital twin manufacturing system. In order to achieve this goal, this thesis will determine how one can accurately simulate the movements

and cycle-time of the robot based on the given commands. This thesis will also demonstrate how users can monitor and control the real robot remotely, allowing for optimization and production re-planning to be performed, adhering to the digital twin concept.

1.2 Limitations

The following limitations apply to the thesis:

- The focus of this work is on the creation of needed capabilities in order to use the iiwa within a digital twin manufacturing system. Overlaying intelligence, such as optimization and production re-planning, will not be considered.
- Only high-level motion instructions in configuration space will be considered during the simulations.
- On-line trajectories and the possibility to recalculate trajectories based on sensor input will not be covered. Only static off-line motions will be considered during this work.
- Simulation in compliance mode and reaction to unforeseen events will not be considered.
- No control aspects are handled in this thesis. Only nominal trajectories are calculated. The control accuracy of the internal controller of the robot is taken for granted.

1.3 Ethical and sustainability aspects

The ethical question one might face when working with robotics and automation is if these processes are taking over manual labor jobs in factories. Automation is a substitute for labor, however research has shown that automation can lead to a higher demand of labor by raising the productivity [12]. The KUKA LBR iiwa 7 robot employed in this thesis is a so-called collaborative robot. It is designed to work with and alongside humans, supporting the workers, contrary to traditional industrial robots which can be considered dangerous. By letting the robot work alongside the human performing the repetitive and straining tasks by assisting the worker it can create a more sustainable work environment.

1.4 Thesis structure

The report is structured in the following way:

2 Virtual Representation: This chapter describes how to create a virtual representation of the real-world robot. Theory regarding robot motion is described.

3 Physical Robot Interface: This chapter describes how the interface with the real physical iiwa robot has been developed.

4 Evaluation of Blending Methods: This chapter ties back to the theory of robot motion and applies it to the iiwa robot. Different methods for calculating the blending motion of the robot is evaluated. An extension is proposed to capture the motion of the iiwa robot.

5 Results: The simulation methods are evaluated, utilizing the developed physical robot interface.

6 Conclusion: Concluding remarks and possible future work is presented.

2 Virtual Representation

This section presents the theory behind a virtual representation of the iiwa robot. It starts with an overview how the motion of a robot is calculated and controlled. Based on this the simulation approach is motivated. Then, theory regarding kinematics and motion generation is presented.

2.1 Robot motion overview

In order to understand how an industrial robot moves, one can look at how such robot is controlled. A simplified graphical representation of how a robot is controlled is drawn in Fig. 1. This illustration is based on the presented drawing in [13].

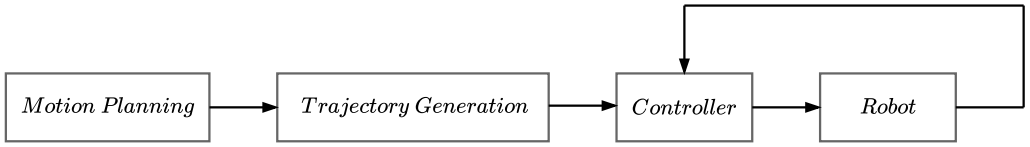


Figure 1: Simple conceptual drawing of a robot control system.

Motion planning defines, on a high-level, how the robot should move. For example, the path of the robot can be established to perform specific tasks, such as arc welding and glue dispensing [11]. The path of the robot represents how it should geometrically move between a set of points to fulfill its task. These points can either be defined as joint positions or as positions where the tool of the robot is located. This path can be programmed by the operator or calculated by path planning algorithms where metrics such as collision avoidance is considered [14].

Trajectory generation introduces the concept of time to the path. At this stage, the overall path of the robot is known. The trajectory is generated along this path with respect to some criteria on velocity, acceleration and the actuator limit of the robot [15]. The actuator limit can be enforced with difference fidelity. In fact, the methods where the maximum permissible velocity and acceleration of every actuator is enforced is a typical occurrence [15]. Limits on the acceleration do not typically capture the non-linear dynamics of the robot. Given these limitations, more involved methods where the torque limit of the every joint have been developed [16, 17]. The downside of torque limit-based methods are the need for dynamical models. The dynamic parameters of the robot are typically not provided by the manufacturer and need to be identified using system identification methods. An overview of system identification for robots can be found in [18].

To allow the robot to traverse a path fast, consecutive motions can be blended

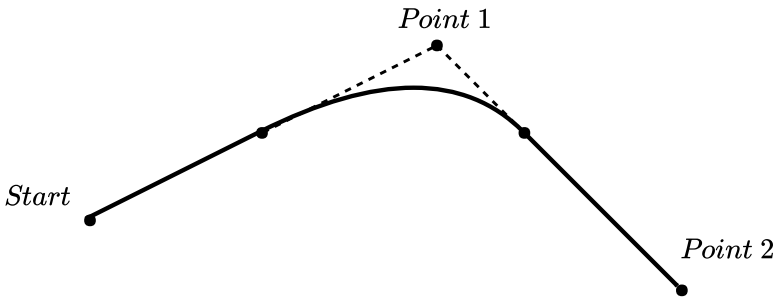


Figure 2: Example of motion blending.

together as can be seen in Fig 2. For example, during the *motion planning* stage, it is determined that the robot should go via point 1 and continue to point 2. In industry, the robot is typically allowed to take a shortcut, allowing it to not decelerate fully and traverse the path faster. There are various approaches to calculate the shape of the blend. One of which is purely geometrical, where the shape of the blend is not a function of time. Such methods usually describe the path using a normalized scalar path parameter, examples of this can be found in [19] and [20]. Methods where the shape of the blend is a function of time and velocity, such as the parabolic blend method [21]. This method assumes that the timing of the way-points is known and therefore the velocity is already known. When the blend is a function of time, it essentially creates a coupling between the motion planning and the trajectory generation since the path might change slightly during the trajectory generation.

The trajectory is executed by the *controller* of the robot. The controller tries to follow the calculated trajectory and corrects the control input to the robot based on the state of the robot in order to achieve high accuracy when following the trajectory. More information regarding the control of the robots can be found in [15]. The four simplified steps illustrated in Fig. 1 enables the robot to move and perform its programmed task.

Generally, industrial robots have their own language in which high-level motion instructions may be specified to accomplish a given task. If the robot-specific language is utilized, then trajectory generation is performed internally in the robot controller – adhering to the constraints specified by the manufacturer. There are middle-ware, such as the Robotic Operating System (ROS) [22], in which motion planning and trajectory generation are implemented based on adaptive sensor input. Instead of high-level instructions, the trajectory is passed to the controller and is then executed provided that the robot has such functionalities.

2.2 Simulation approach

One desired outcome of this thesis is to investigate how one can simulate the movements and cycle time of the iiwa robot. The cycle time is the time it takes for the robot to finish executing its motion program.

The approach taken in this thesis is to mimic how the motion planner and the trajectory generator translate high-level instructions into trajectories. Once the trajectory is calculated, the accuracy of the robot when following such trajectory is assumed to be perfect. According to the specification of the iiwa robot, its accuracy and repeatability when following the controller-generated trajectory is high [10]. It is therefore a question of capturing the path and time evolution that internal robot controller generates. If the trajectory is known, the movements with respect to time are also known. As such, these movements can be simulated under the assumption that the robot will follow the trajectory perfectly.

The manufacturer-specified high-level instructions defining the path of the iiwa robot are found in the programming manual [23]. High-level movement instructions, such as point-to-point (PTP) in configuration space, linear tool (LIN), circular tool movement (CIRC), and interpolating spline (SPL) can be programmed. The movement instructions may be associated with options such as velocity for the path and how consecutive motions should be blended together, either as a Cartesian distance or as a percentage in the configuration space.

A delimitation has been implemented in this work to limit the considerations to PTP instructions only. The blend is specified as either a Cartesian distance or a percentage in configuration space. The geometric path and the motion blending are calculated based on high-level instructions in configuration space without any consideration of time. Once the path is known, the trajectory is calculated with respect to joint velocity and acceleration bounds. This allows the velocity to be changed along the path without changing the shape of the path. This approach is common for industrial robots since it is desirable to verify the motions with lower velocities before the motions are executed with its full velocity in production.

In reality, the trajectory generation needs to consider the effects of jerk and joint torque for the robot to accurately follow the trajectory. However, as the simulation assumes ideal conditions, this process is omitted. Thus, it becomes a question of achieving a movement execution time that is as close as possible to the manufacturer-created trajectory generation with as little information as possible.

The following sections in this chapter therefore present the necessary theory needed to accurately describe the path and trajectory of the robot.

2.3 Geometric Path

The geometric path is calculated based on high-level instructions, taking into consideration the kinematics of the robot. The geometric path represents how the robot moves geometrically without any consideration of time. The following section introduces the necessary theoretical concepts to calculate the geometric path of the robot.

2.3.1 Serial robot kinematics

The kinematics of the robot describes the analytical relationship between the joint positions and the rotation and translation of the end effector (tool) of the robot. This relationship can be modelled using homogeneous transformations [15]. The relative distance and rotation between two frames can be represented with homogeneous transformations, $\boldsymbol{T}_b^a$, which belongs to the special Euclidean group $SE(3)$. The superscript represents the frame wherein the subscript frame is expressed. The homogeneous transformation, $\boldsymbol{T}_b^a$, is made up by a translation $\boldsymbol{t}_b^a$ and a rotation $\boldsymbol{R}_b^a$. The translation, $\boldsymbol{t}_b^a$, defines the distance between the origin of frame b and frame a expressed in frame a, defined as a 3×1 vector

$$\boldsymbol{t}_b^a = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}. \tag{1}$$

The rotation $\boldsymbol{R}_b^a$ defines how frame b is oriented with respect to frame a. It is a 3×3 matrix, where the columns of $\boldsymbol{R}_b^a$ define the position of the the unit axis of frame b expressed in frame a when the origins coincide

$$\boldsymbol{R}_b^a = \begin{bmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{bmatrix}. \tag{2}$$

The rotational and translation relationship between two frames are combined to form the homogeneous 4×4 matrix $\boldsymbol{T}_b^a$

$$\boldsymbol{T}_b^a = \begin{bmatrix} \boldsymbol{R}_b^a & \boldsymbol{t}_b^a \\ 0\ 0\ 0 & 1 \end{bmatrix}. \tag{3}$$

The transformation can be visualized as in Fig. 3 where frame b is expressed in frame a. Consecutive homogeneous transformations can be compounded as in Eq. (4). If the transformations between frame a and frame b as well as

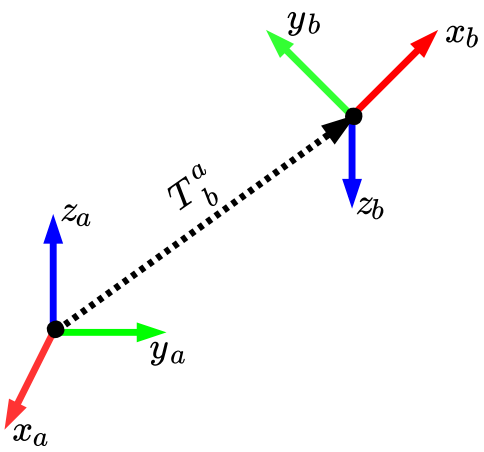


Figure 3: Simple illustration of a homogeneous transformation.

frame b and frame c are known, they can be used to find the transformation between frame a and frame c. This is done by simply post-multiplying the homogeneous transformations together

$$\boldsymbol{T}^a_c = \boldsymbol{T}^a_b \boldsymbol{T}^b_c. \tag{4}$$

Most conventional industrial robots can be seen as serial manipulators, where n + *1* rigid links are connected by n revolute joints in a series. The base of the manipulator, link 0, is typically static. An end effector, such as a welding gun, sealant dispenser, or gripper, is normally attached to link $n+1$. Every link i has an associated frame i that is fixed with respect to the link. The transformation between consecutive link $i-1$ and link i can be parameterized with respect to the joint value θ_i.

The end effector position of the robot with respect to the base frame, $\boldsymbol{T}^0_E$, is calculated using the compound rule given in Eq. (4). Consecutive link transformation parameterized with respect to the joint values are compounded to reach the end effector frame

$$\boldsymbol{T}^0_E(\boldsymbol{\theta}) = \boldsymbol{T}^0_1(\theta_1)\boldsymbol{T}^1_2(\theta_2)...\boldsymbol{T}^{N-1}_N(\theta_N)\boldsymbol{T}^N_E. \tag{5}$$

This allows us to calculate the forward kinematics, meaning given the joint values $\boldsymbol{\theta}$ of the robot, what is the position of the end effector $\boldsymbol{T}^0_E(\boldsymbol{\theta})$. The transformations, with their respective joint value parameterization between links, can be determined systematically using the Denavit-Hartenberg (DH) convention [24]. To describe the relationship using the DH convention, the frames of the respective links are first assigned. This is done by choosing z_i along the axis of joint $i+1$. The origin of frame i is placed at the intersection

of z_i with the common normal between z_i and z_{i-1}. The x_i axis is assigned along the common normal between z_i and z_{i-1} in the direction of the common normal from $i-1$ to i. The y_i axis is assigned to complete the right-handed frame. An example of the frame assignment is shown in Fig. 4.

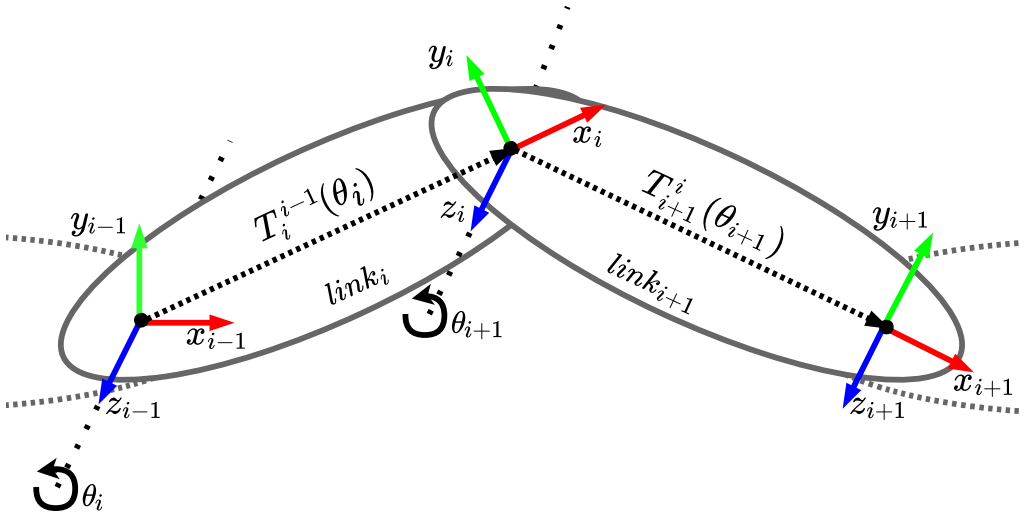


Figure 4: Frame assignment following the DH convention for a serial manipulator.

When all the frames are assigned to every link, the homogeneous transformations, Eq. (3), between links can be obtained. If the DH convention is employed, only four parameters are required to be determined to describe the relationship between consecutive frames. The four parameters a_i, α_i, d_i, and θ_i are determined as follows:

d_i: Distance between the respective origin of frame $i-1$ and i along the axis z_{i-1}.

θ_i: Angular displacement between axis x_{i-1} and x_i around axis z_{i-1}.

a_i: Distance between the respective origin of frame $i-1$ and i along the axis $-x_i$.

α_i: Angular displacement between axis z_{i-1} and z_i around axis x_i.

The relationship between frame $i-1$ and i can now be calculated as in Eq. (8), following the DH convention. Since the z-axis is chosen along the axis of the joint, for a revolute joint, the transformation is parameterized with respect to the joint value.

$$\boldsymbol{T}_i^{i-1}(\theta_i) = Translation_z(d_i)Rotation_z(\theta_i)Translation_x(a_i)Rotation_x(\alpha_i)$$

(6)

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(7)

$$= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

2.3.2 iiwa kinematics

The KUKA iiwa robot is a serial manipulator with seven revolute joints. A graphical rendition of the robot can be seen in Fig. 5. The forward kinematics of the iiwa robot are determined in order to calculate the position of the end effector following the method described in Section 2.3.1. The end effector is the tool carried by the robot. The frame associated with the tool defines the tool center point (TCP). The link frames and the DH parameters for the iiwa robot have been identified and are given in Table 1 and Fig. 6, respectively.

Table 1: DH-parameters for the KUKA iiwa lbr 7 R800.

$Link_i$	a_i	α_i	d_i	θ_i
1	0	$-\frac{\pi}{2}$	$d_1 = 340\ mm$	θ_1
2	0	$\frac{\pi}{2}$	0	θ_2
3	0	$\frac{\pi}{2}$	$d_3 = 400\ mm$	θ_3
4	0	$-\frac{\pi}{2}$	0	θ_4
5	0	$-\frac{\pi}{2}$	$d_5 = 400\ mm$	θ_5
6	0	$\frac{\pi}{2}$	0	θ_6
7	0	0	$d_7 = 152\ mm$	θ_7

The end effector position relative to the base of the robot, Eq. (5), can now easily be calculated as a function of the seven joints, $\boldsymbol{\theta}$

$$\boldsymbol{T}_E^0(\boldsymbol{\theta}) = \boldsymbol{T}_1^0(\theta_1)\boldsymbol{T}_2^1(\theta_2)\boldsymbol{T}_3^2(\theta_3)\boldsymbol{T}_4^3(\theta_4)\boldsymbol{T}_5^4(\theta_5)\boldsymbol{T}_6^5(\theta_6)\boldsymbol{T}_7^6(\theta_7)\boldsymbol{T}_E^7. \quad (9)$$

2.3.3 Path representation

This section presents the theory and concepts which describe the process to geometrically represent the path of a robot. The path of the robot is made up

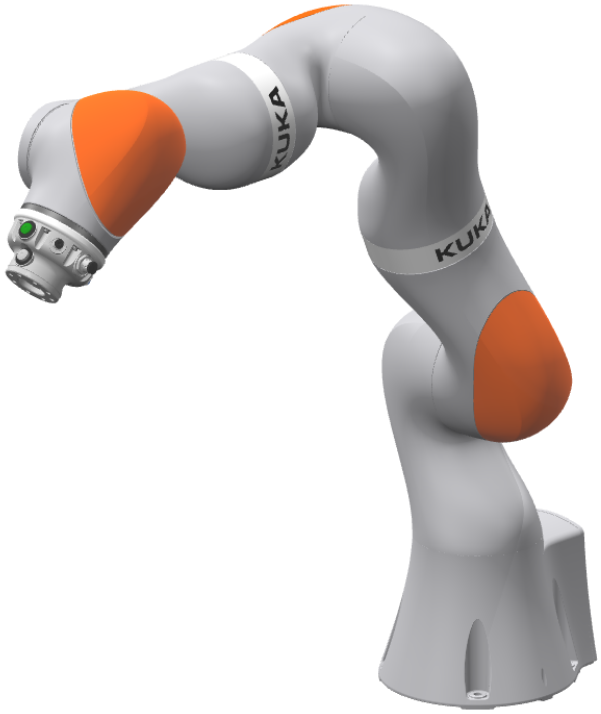


Figure 5: 3D rendering of the KUKA LBR iiwa R800 robot.

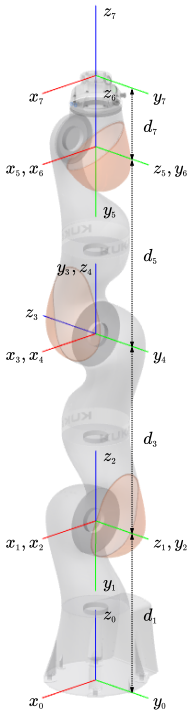


Figure 6: Link frames following the DH convention. Joint θ_i rotates around axis z_{i-1} in a counter-clockwise direction.

by a set of segments which together creates a path. A segment, $\boldsymbol{P}_i(s)$, describes how the robot moves between two positions defined in the configuration space of the robot, $\boldsymbol{\theta}_{i-1}$ and $\boldsymbol{\theta}_i$. The segment is conveniently parameterized with the scalar parameter s which is defined between 0 and 1. The procedure of describing the segment with a scalar parameter s allows one to separate the notion of time from the path, [25]. Once the path is known, the time evolution can be calculated.

In this work, the PTP or point-to-point instruction has been considered for simulation. The PTP instruction defines a straight line segment in configuration space. A PTP segment, $\boldsymbol{P}_i(s)$, between two joint position vectors, $\boldsymbol{\theta}_{i-1}$ and $\boldsymbol{\theta}_i$, can be linearly interpolated as

$$\boldsymbol{P}_i(s) = \boldsymbol{\theta}_{i-1} + s(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}), \quad s \in [0,\ 1]. \tag{10}$$

To allow for the robot's smooth transition between two consecutive segments, motion blending can be used. When the high-level motion instruction is programmed, it is typically defined alongside a joint-position vector $\boldsymbol{\theta_i}$ and a radius of where the blend will start, b_r. This radius defines a sphere in the operational space centered at the corresponding Cartesian position c_i of the joint-position vector $\boldsymbol{\theta_i}$ as can be seen in Fig. 7.

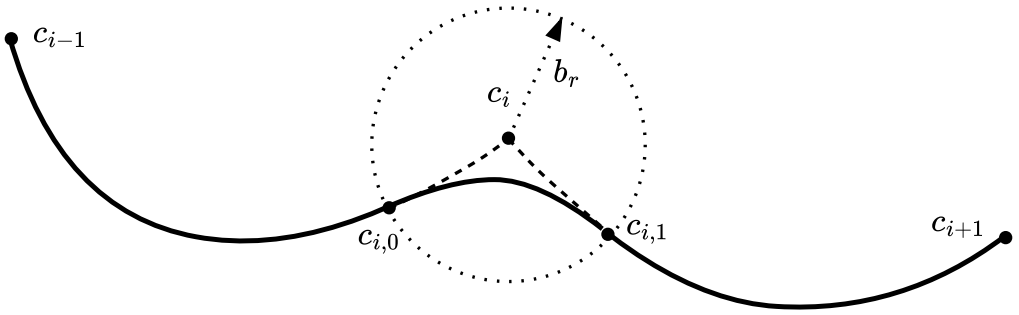


Figure 7: Geometrical path of the TCP when the robot is allowed to blend the consecutive segments.

The path of the TCP will at some point intersect with the incoming segment $\boldsymbol{P}_i(s)$ and the outgoing segment $\boldsymbol{P}_{i+1}(s)$. The Cartesian positions of the end effector $\boldsymbol{c}_{i,0}$ and $\boldsymbol{c}_{i,1}$ correspond to the joint position vectors $\boldsymbol{\theta}_{i,0}$ and $\boldsymbol{\theta}_{i,1}$. In turn, these joint position vectors correspond to where the motion blending will start and end. These intersection can be found by searching along the segment until the absolute distance of the TCP is displaced at the distance of the radius b_r from c_i. An example of such a search is given by Algorithm 1 which has been proposed in [26]. The algorithm is applied to the incoming and outgoing segments respectively; although the outgoing segment is reversed. The algorithm returns values for s that correspond to the joint values $\boldsymbol{\theta}$ that make the TCP intersect with the sphere.

Algorithm 1 Determines where translation of the end effector defined by the segment $\boldsymbol{P}(s)$ intersects a sphere with the radius b_r within a tolerance ϵ.

Require: $(\boldsymbol{P}(1) = \boldsymbol{\theta}_i) \wedge (b_r > 0)$
$c_i \leftarrow \boldsymbol{T}_E^0(\theta_i)$ Eq. (9)
$\Delta L \leftarrow 0$
$\Delta s \leftarrow 1$
$s \leftarrow 1$
while $|\Delta L - b_r| > \epsilon$ **do**
  if $\Delta L \geq b_r$ **then**
    $s \leftarrow s + \frac{\Delta s}{2}$
  else
    $s \leftarrow s - \frac{\Delta s}{2}$
  end if
  $\Delta s \leftarrow \frac{\Delta s}{2}$
  $\Delta L \leftarrow ||c_i - \boldsymbol{T}_E^0(\boldsymbol{P}(s))||$
end while
return s

When the start and stop of the blend is know, it is a question of connecting the segments with a smooth transition. A geometric method for performing motion blending between two consecutive segments has been proposed in [20]. This method works by constructing a new blend segment $\boldsymbol{P}_b(s)$ that blend together an incoming linear segment $\boldsymbol{P}_0(s)$ and outgoing linear segment $\boldsymbol{P}_1(s)$. The incoming segment is defined from the start of the blend to the joint position that is to be blended away

$$\boldsymbol{P}_0(s) = \boldsymbol{\theta}_{i,0} + s(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0}). \tag{11}$$

The outgoing segment is defined from the joint position that is to be blended away to the end of the blend

$$\boldsymbol{P}_1(s) = \boldsymbol{\theta}_i + s(\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i). \tag{12}$$

To achieve a smooth transition between $\boldsymbol{P}_0(s)$ and $\boldsymbol{P}_1(s)$, the method presented in [20] introduces a function $\alpha(s)$, together with Eq. (11) and Eq. (12), to form the blend segment

$$\boldsymbol{P}_b(s) = \boldsymbol{P}_0(s) + \alpha(s)(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)). \tag{13}$$

An conceptual illustration of the resulting blend together with the used notation is given in Fig. 8.

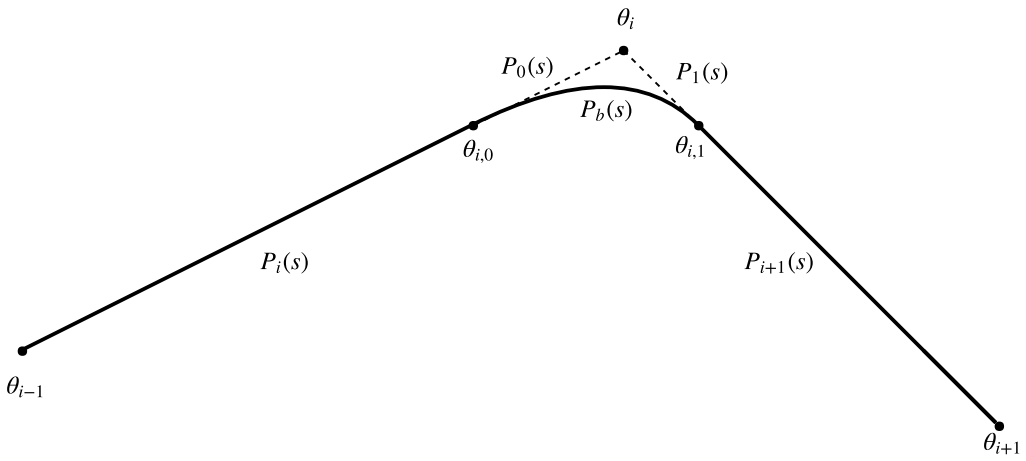


Figure 8: Conceptual drawing of a blending motion in configuration space.

The goal is to calculate a blend segment, $\boldsymbol{P}_b(s)$, that ensures a smooth transition between $\boldsymbol{P}_0(s)$ and $\boldsymbol{P}_1(s)$. The resulting blended segment $\boldsymbol{P}_b(s)$ has to be continuous during the start and end transition in the zone. Continuity of the N:th degree means that the the blend segment has to have continuity of the derivatives up the the N:th degree

$$\begin{aligned} \frac{d^n \boldsymbol{P}_b(0)}{ds^n} &= \frac{d^n \boldsymbol{P}_0(0)}{ds^n}, \qquad n = 1, 2, ..., N \\ \frac{d^n \boldsymbol{P}_b(1)}{ds^n} &= \frac{d^n \boldsymbol{P}_1(1)}{ds^n}, \qquad n = 1, 2, ..., N. \end{aligned} \tag{14}$$

The function $\alpha(s)$, which fulfills the constraints in Eq. (14), is found by differentiating $\boldsymbol{P}_b(s)$, as in Eq. (15) & Eq. (16).

$$\frac{d\boldsymbol{P}_b(s)}{ds} = \frac{d\boldsymbol{P}_0(s)}{ds} + \frac{d\alpha(s)}{ds}\left(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)\right) + \alpha(s)\left(\frac{d\boldsymbol{P}_1(s)}{ds} - \frac{d\boldsymbol{P}_0(s)}{ds}\right) \tag{15}$$

$$\begin{aligned} \frac{d^2\boldsymbol{P}_b(s)}{ds^2} =& \frac{d^2\boldsymbol{P}_0(s)}{ds^2} + \frac{d^2\alpha(s)}{ds^2}\left(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)\right) + \\ & 2\frac{d\alpha(s)}{ds}\left(\frac{d\boldsymbol{P}_1(s)}{ds} - \frac{d\boldsymbol{P}_0(s)}{ds}\right) + \alpha(s)\left(\frac{d^2\boldsymbol{P}_1(s)}{ds^2} - \frac{d^2\boldsymbol{P}_0(s)}{ds^2}\right) \end{aligned} \tag{16}$$

By analyzing Eq. (15) and Eq. (16), it can be seen that the constraints in Eq. (14) are fulfilled if $\alpha(s)$ is chosen so that it satisfies the following constraints

$$\begin{aligned}
\alpha(0) &= 0 \\
\alpha(1) &= 1 \\
\frac{d^n\alpha(0)}{ds^n} &= 0, && n = 1, 2, ..., N \\
\frac{d^n\alpha(1)}{ds^n} &= 0, && n = 1, 2, ..., N.
\end{aligned} \tag{17}$$

If $\alpha(s)$ is chosen as a polynomial of degree $2N+1$, where N specifies the degree of continuity in the zone, the $\alpha(s)$ that adheres to the constraints in Eq. (17) can be solved for. Some examples of such polynomials, where $N = 1,\ 2\ \&\ 3$, are given in Eq.(18).

$$\begin{aligned}
\alpha(s) &= 3s^2 - 2s^3, && N = 1 \\
\alpha(s) &= 10s^3 - 15s^4 + 6s^5, && N = 2 \\
\alpha(s) &= 35s^4 - 84s^5 + 70s^6 - 20s^7, && N = 3
\end{aligned} \tag{18}$$

This method described above has been used in works such as [19] and [26] to describe the paths of industrial robots. The described methods provides a smooth transitions between consecutive segments, but it is not possible to control the shape of the transition. A method to allow control of the shape of the transition is presented as well in [20]. This is done by introducing another polynomial, $\beta(s)$, and a fixed vector $\boldsymbol{u}$ to Eq. (13)

$$\boldsymbol{P}_b(s) = \boldsymbol{P}_0(s) + \alpha(s)(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)) + \beta(s)\boldsymbol{u}. \tag{19}$$

To ensure a continuous transition between the incoming and outgoing segment, the following constraint has to apply to the polynomial $\beta(s)$ to ensure continuity of N order using the same reasoning as for $\alpha(s)$

$$\begin{aligned}
\beta(0) &= 0 \\
\beta(1) &= 0 \\
\frac{d^n\beta(0)}{ds^n} &= 0, && n = 1, 2, ..., N \\
\frac{d^n\beta(1)}{ds^n} &= 0, && n = 1, 2, ..., N.
\end{aligned} \tag{20}$$

A polynomial of order $2N + 1$ will satisfy the constraints in (20). The polynomial $\beta(s)$ is although chosen as a polynomial of order $2N + 2$ to get one more

degree of freedom. This extra degree of freedom allows the shape of the blend to be controlled. Example of such polynomials adhering to the constraints in Eq. (20) can be calculated as follows

$$\begin{aligned}
\beta(s) &= k(s^2 - 2s^3 + s^4), && N = 1 \\
\beta(s) &= k(-s^3 + 3s^4 - 3s^5 + s^6), && N = 2 \\
\beta(s) &= k(s^4 - 4s^5 + 6s^6 - 4s^7 + s^8), && N = 3.
\end{aligned} \tag{21}$$

A polynomial of order $2N+2$ is obtained that is scaled by a free scalar variable k. The term that is unknown in Eq. (26) is $\beta(s)\boldsymbol{u}$, k is set to one in Eq. (21) and is instead seen as the scaling of the magnitude of $\boldsymbol{u}$. In [20], it is proposed to calculate the vector $\boldsymbol{u}$ by minimizing the average acceleration during the blend

$$min \int_0^1 \left\| \frac{d^2 \boldsymbol{P}_b(s)}{ds^2} \right\|^2 ds. \tag{22}$$

The optimal value of $\boldsymbol{u}$ will depend on the order of $\alpha(s)$ and $\beta(s)$. In [20], has been shown that if the continuity order N is chosen as two, the average acceleration is minimized by assigning the vector $\boldsymbol{u}$ to the following

$$\boldsymbol{u} = -\frac{15}{2}\left(\frac{d\boldsymbol{P}_2(s)}{ds} - \frac{d\boldsymbol{P}_1(s)}{ds}\right) \tag{23}$$

$$= -\frac{15}{2}\left((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0})\right) \tag{24}$$

$$= -k\left((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0})\right) \tag{25}$$

The optimal vector $\boldsymbol{u}$ is scaled with the scalar k which at its optimum in terms of minimizing the average acceleration is set to $\frac{15}{2}$. The final equation for the blending motion using this method now becomes

$$\boldsymbol{P}_b(s) = \boldsymbol{P}_0(s) + \alpha(s)(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)) - k\beta(s)\left((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0})\right). \tag{26}$$

The so far described methods to achieve a smooth transition between consecutive segments utilize polynomials, (13) and (26). Another method that exists in the literature simply creates a circular segment between the consecutive segment. This method has been proposed in [27]. A graphical representation is given in Fig. 9. The incoming, $\boldsymbol{P}_i(s)$, and outgoing, $\boldsymbol{P}_{i+1}(s)$, segment

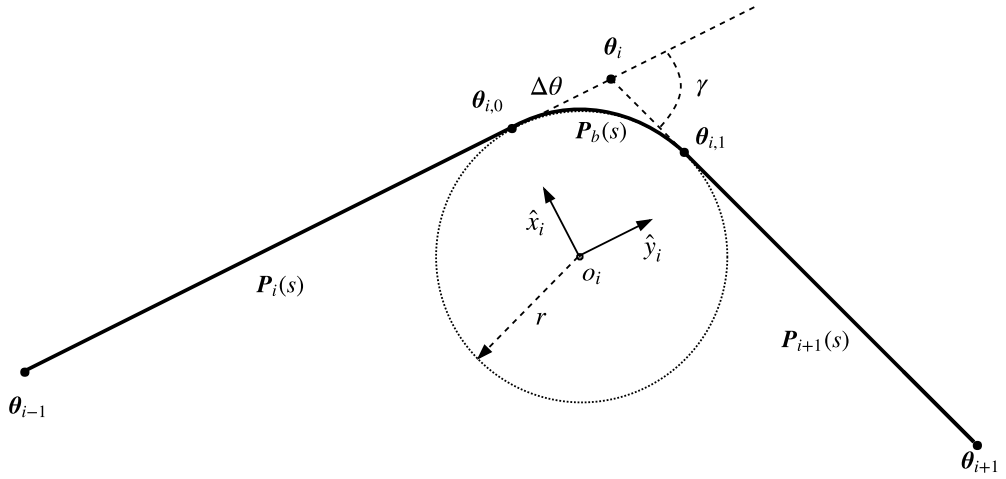


Figure 9: Conceptual drawing of the resulting circular blend.

is simply blended together by following the circumference of a circle in the configuration space.

The circular segment is calculated by first defining the vector $\hat{\boldsymbol{y}}_i$ as the unit vector specifying the direction from $\boldsymbol{\theta}_{i-1}$ to $\boldsymbol{\theta}_i$

$$\hat{\boldsymbol{y}}_i = \frac{\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}}{||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}||}. \tag{27}$$

The angle between the incoming and outgoing segment, γ_i, is calculated as

$$\gamma_i = cos^{-1}(\hat{\boldsymbol{y}}_i \circ \hat{\boldsymbol{y}}_{i+1}). \tag{28}$$

Given a defined distance in the configuration space, $\Delta\theta$, from where the blend starts and stops, the radius of the circle can be calculated

$$r_i = \frac{\Delta\theta}{tan(\frac{\gamma_i}{2})}. \tag{29}$$

When r_i and γ_i is known, the center of the circle, $\boldsymbol{o}_i$, can be calculated

$$\boldsymbol{o_i} = \boldsymbol{\theta_i} + \frac{\hat{\boldsymbol{y}}_{i+1} - \hat{\boldsymbol{y}}_i}{||\hat{\boldsymbol{y}}_{i+1} - \hat{\boldsymbol{y}}_i||} \frac{r_i}{cos(\frac{\gamma_i}{2})}. \tag{30}$$

The vector $\hat{\boldsymbol{x}}_i$ is orthogonal to $\hat{\boldsymbol{y}}_i$ and gives the direction from the center of the circle, $\boldsymbol{o}_i$, to the start of start of the blend $\boldsymbol{\theta}_{i,0}$

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{\theta}_i - \Delta\theta\hat{\boldsymbol{y}}_i - \boldsymbol{o}_i}{||\boldsymbol{\theta}_i - \Delta\theta\hat{\boldsymbol{y}}_i - \boldsymbol{o}_i||}. \tag{31}$$

Utilizing these defined quantities, one can now calculate the joint values of the robot throughout the circular blend

$$\boldsymbol{P}_b(s) = \boldsymbol{o}_i + r_i(\hat{\boldsymbol{x}}_i cos(s\gamma_i) + \hat{\boldsymbol{y}}_i sin(s\gamma_i)). \tag{32}$$

Three existing methods for calculating the path during a motion blend has now been described. One method simply blends the motions together using a polynomial, Eq. (13), another method that minimizes the average acceleration Eq. (26) during the blend and a third method that simply follows the circumference of a circle to achieve a smooth transition.

2.4 Trajectory generation

The trajectory calculation problem is to assign timestamps to the s-parameterized path described in the previous section. The method used during this work for assigning timestamps has been proposed in [28]. This methods solves the trajectory generation as a optimization problem with consideration to constraints to the joint velocity, joint acceleration and joint torque. To consider the joint torque, knowledge of the dynamical parameters is needed. The manufactures of industrial robots do typically not disclose to what extent the dynamics is considered during trajectory generation. The consideration to joint torque is therefore not considered since the treatments is unknown and will require system identification. This sections first presents the used existing algorithm and later describes how it has been applied to the iiwa.

2.4.1 Trajectory calculation algorithm

The time stamp assignment algorithm works by for every segment in the path, $\boldsymbol{P}_i(s)$, uniformly sample every segments with N samples on the interval of $s \in [0\ 1]$. This results in a original time assignment, $t_{i,0}$, for every sample $\boldsymbol{\theta}_i$. The resulting path as a function t_0 can be seen in Fig. 10.

Every sample, $\boldsymbol{\theta}_i$, represents a geometric position defined in configuration space that has the same dimension as the number of joints, n. Every sample, $\boldsymbol{\theta}_i$, has an assigned joint specific maximum velocity and maximum acceleration that constrains the velocity and acceleration as can be seen in Eq. 33. Different samples can have different velocity and acceleration constraints.

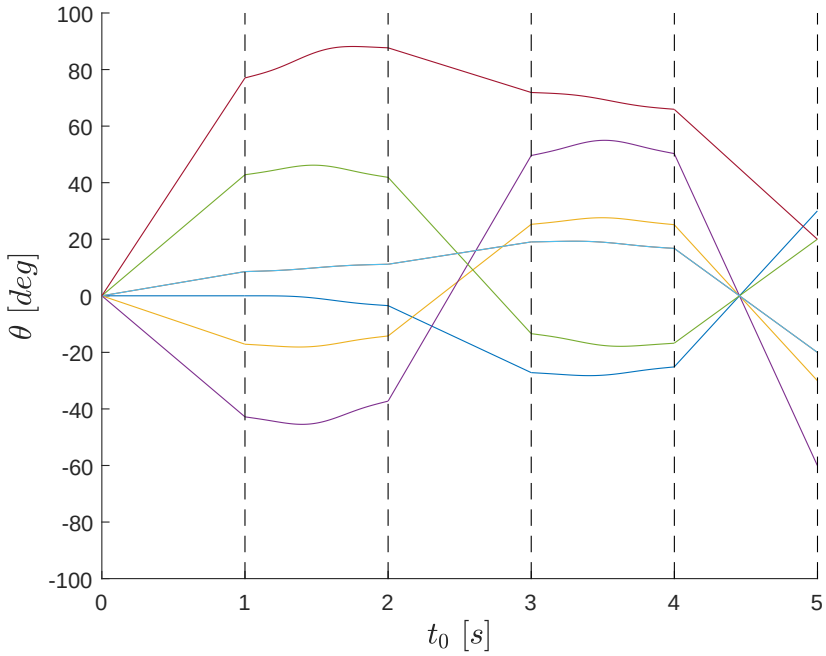


Figure 10: Example of sampling a random path by the s-parameterization.

$$\begin{aligned} -\dot{\theta}^k_{i,max} \leq \dot{\theta}^k_i \leq \dot{\theta}^k_{i,max} \\ -\ddot{\theta}^k_{i,max} \leq \ddot{\theta}^k_i \leq \ddot{\theta}^k_{i,max}, \end{aligned} \qquad k = 1, 2, ..., n \tag{33}$$

The algorithm sweeps over the samples, one joint at the time, and calculates what the updated time stamp t^k_{i+1} that adheres to the velocity and acceleration constraint of sample $i+1$. The updated time stamp for sample $\boldsymbol{\theta}_{i+1}$ is selected as the maximum permissible time-stamps among the joints, Eq. 34.

$$t_{i+1} = max[t^1_{i+1}, t^2_{i+1}, ..., t^k_{i+1}], \qquad k = 1, 2, ..., n \tag{34}$$

This means that at least one joint is always running in saturation, either in terms of velocity or acceleration. An illustration of how the velocity and acceleration constraints effects the updated time stamp for joint k is given in Fig. 11.

2.4.2 iiwa trajectory calculation

To enable the use of the trajectory calculation algorithm described in the previous section, the joint velocity and acceleration limits for the iiwa robot

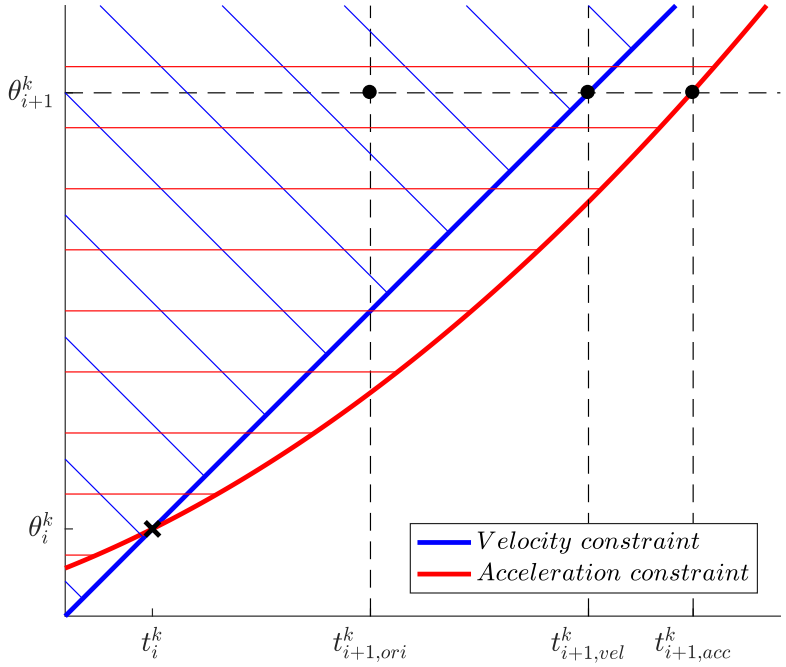


Figure 11: Illustration of how timestamps are shifted with respect to velocity and acceleration constraints. In this case, the permissible time-stamp is determined by $t^k_{t+1,acc}$.

needs to be determined. The maximum joint velocity for the iiwa 7 is given in the technical specification. The maximum acceleration has been determined experimentally by moving the individual joints short distances at max speed while sampling the resulting path. From the resulting path, the acceleration profile is obtained by differentiating twice and observing the maximum value, see Appendix A for more details. The resulting maximum joint velocity and acceleration is summarized in Table 2

Table 2: Velocity and acceleration limits for the iiwa.

Joint k	1	2	3	4	5	6	7
$\dot{\boldsymbol{\theta}}_{max}$ [deg/s]	98	98	100	130	140	180	180
$\ddot{\boldsymbol{\theta}}_{max}$ [deg/s^2]	490	490	500	650	700	900	900

When programming the robot, the high-level instructions can be assigned a velocity specifying how fast the robot moves during this instruction. For a PTP instruction, this is done by specifying the *relative velocity*($v_{\%} \in (0\ 1]$). The *relative velocity* is defined as a percentage relative to the maximum joint velocity, $\dot{\boldsymbol{\theta}}_{max}$. This is incorporated in the trajectory calculation algorithm through the velocity constraint as given in Eq. 35.

$$
\begin{aligned}
-v_{\%}\dot{\theta}^k_{i,max} \leq \dot{\theta}^k_i \leq v_{\%}\dot{\theta}^k_{i,max} \\
-\ddot{\theta}^k_{i,max} \leq \ddot{\theta}^k_i \leq \ddot{\theta}^k_{i,max}, \qquad\qquad k = 1, 2, ..., n
\end{aligned} \tag{35}
$$

The constraint is applied to the samples associated with the segment defined by the PTP instruction. When two segments are blended together, the samples associated with the blend segment is constrained by the velocity of the following PTP instruction. This has been determined experimentally, see Appendix B. If there is no blend defined for the PTP instruction, the robot is kept stationary at the point associated with the instruction for 0.05 seconds, see Appendix C. This is likely imposed by the manufacturer to decrease the vibration and tracking error.

Utilizing this algorithm, it is now possible to calculate the time evolution along the geometrical path and obtain a cycle time estimation. The next chapter will deal with how the robot can be remotely controlled and sampled to get the real cycle time.

3 Physical Robot Interface

This chapter presents how the communication with the real iiwa robot has been establish to allow synchronization and control of the robot. An application called iiwaDrive has been created that enables one to communicate with the real robot over a network. This chapter presents an overview of the implementation and the underlying design choices.

3.1 iiwaDrive implementation

To allow the KUKA iiwa robot to work within a digital twin manufacturing system, basic functionalities need to be made available for external applications to connect to the real robot. The state of the virtual representation needs to be synchronized with the real-world iiwa robot to allow the intelligence behind the digital twin to simulate possible outcomes. The state of the real iiwa robot that needs to be monitored, to allow synchronization, has been identified as follows:

Joint positions state: The position of the robots seven axis. Utilizing the forward kinematics, this defines the position of the robot in space.

Input/Output(IO) state: The robot is likely to carry some kind of tool. The tool can be controlled through the IO interface. The state of the IO defines the state of the tool.

Sensor state: Various sensors can be mounted on the robot, which data can be useful for external applications. For example, the iiwa robot has temperature sensors in every joint. The temperature reading can be useful for diagnostic purposes and possible fault detection such as overheating.

In a digital twin manufacturing system, there is some form of intelligence and decision-making process connected to the real robot. There is therefore a need to remotely control the robot, allowing the decision maker to act upon its decision by moving the robot. This can be done on the iiwa robot by specifying high-level motion instructions such as PTP that the robot then executes. It is also possible to specify a trajectory, timestamped joint positions, that the iiwa robot can execute.

The iiwa robot is controlled through the KUKA Sunrise Cabinet[29]. The Sunrise cabinet is the next generation of robot controllers from KUKA. It is running the Sunrise Operating System (OS) [23] which allows the user to program the robot through an extensive JAVA interface. The JAVA interface exposes all the features needed to acquire the states mentioned, as well as the possibility to run high-level motion instructions. These features are however only available through the code running on the Sunrise cabinet. If the iiwa

robot is to be part of a digital twin manufacturing system, these features need to be exposed to external applications. Luckily, the Sunrise cabinet has two Ethernet ports that can be used to communicate with other applications over a network.

The iiwaDrive has been created specifically to unlock the features of monitoring and control of real iiwa robot from external application over a network. The iiwaDrive utilizes the Fast Robot Interface (FRI) [30] that gives real-time access to the KUKA Sunrise OS. The FRI connection allows for monitoring of joint positions at up to 1 ms intervals from an external PC. No other protocol currently offer the speed that is achievable with FRI since it is interfacing with the real-time system of the Sunrise OS. It is also possible to command the robot through the FRI connection by cyclically sending new joint positions to the robot. This effectively bypasses the internal robot trajectory generation that translates high-level motion instruction. The developed iiwaDrive offers both capabilities in terms of running trajectories through FRI and letting the internal trajectory generator move the robot though high-level motion instructions. Besides FRI, iiwaDrive uses the ZeroMQ [31] messaging library to interface with the robot from an external PC. The ZeroMQ was chosen as the messaging library because of its asynchronous messaging capability, meaning that the applications or processes that share messages will not become blocked.

External applications that have a virtual representation of the real iiwa robot can connect to the iiwaDrive through the ZeroMQ interface. The iiwaDrive can stream the joint positions of the real robot allowing the virtual representation to be synchronized with the real robot as well as instruct the robot to execute motions. The iiwaDrive has three main parts: a JAVA program running on the iiwa Sunrise Cabinet, a C++ application called iiwaDrive running on a external PC and a external application that has a virtual representation of the robot. The external application can either be running on the same PC as the iiwaDrive or running on a second PC connected over a network with the PC running the iiwaDrive.

A conceptual overview of the iiwaDrive is given in Fig. 12. The following sections present a more detail description of its main parts.

3.1.1 Sunrise Cabinet Robot Program

A robot program written in JAVA has been created that is running on the Sunrise Cabinet. When the robot program is started, the FRI communication is established and two ZeroMQ sockets are created operating on two different ports. One ZeroMQ socket handles incoming instruction from the iiwaDrive and another is used to synchronize the state of the robot program with the iiwaDrive. The iiwaDrive and the robot program can enter into three different states: *idle*, *runningMotion* and *runningTrajectory* that are synchronized

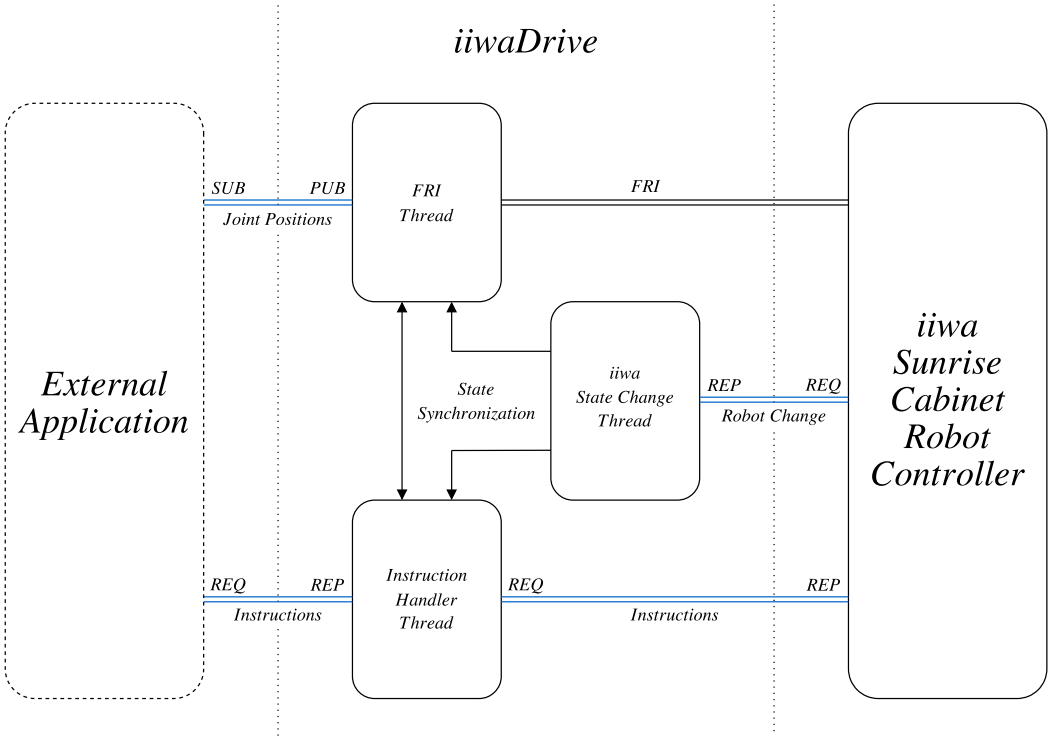


Figure 12: Graphical overview of the iiwaDrive implementation. The blue line represents a ZeroMQ connection while the black is the FRI connection.

through the two sockets. The instructions and state synchronization utilize the *Request(REQ)-Reply(REP)* pattern provided by ZeroMQ. It is a simple server-client pattern where the REQ socket act as a client and the REP socket act as a server. The REQ socket sends a request message to the REP socket, the message is processed and a reply is sent back to the REQ socket. The socket pair is in lockstep, meaning that the REQ socket has to receive a reply from the REP socket before a new request can be sent. This behavior has been used in the iiwaDrive to make the robot program acknowledge the instruction that are sent from he iiwaDrive application, making sure that they have been received. The same goes for when the state is changed at the robot program end, the change of state then has to be acknowledge by the iiwaDrive.

Once the communication has been initialized the application enters the passive idle state where it is waiting for instruction from the iiwaDrive application. The instructions can either be a set of high-level PTP motion commands with an associated motion settings, trajectory execution request, sensor readings request, IO get or IO set. The sensor readings request, IO get or IO set can be executed when the robot program is in the *idle* state. When the iiwaDrive sends high-level motion commands the state changes to *runningMotion*. When the real robot has executed its motions, the robot program sends a state change message to the iiwaDrive changing both the applications state to idle. When a request to run a trajectory is received, the robot program and iiwaDrive enters a *runningTrajectory* mode when the robot is commanded through timestamped joint positions sent through FRI. When the trajectory execution is done, the ii-

waDrive sends a instruction specifying that the trajectory is done and the robot program cancels the trajectory execution and enters the idle state again.

3.1.2 iiwaDrive application

The iiwaDrive application act as a middle layer between the robot and the external application that wants to interface with the robot. When the iiwaDrive is started, three threads are created. One handling incoming instructions, one waiting for state changes from the iiwa robot and one handling the FRI communication.

The FRI communication between the iiwaDrive and the sunrise cabinet has to be running cyclically for the connection to not deteriorate. The Sunrise cabinet sends out data containing information about the current state of the robot such as joint positions and expects a reply from the iiwaDrive within a fixed time interval. The FRI communication is therefore running in its own thread to minimize the computational burden allowing the iiwaDrive to reply within the fixed time interval. The FRI can operate in two modes, monitoring or commanding. If the mode is monitoring, the reply is just an acknowledgment. If the FRI is in commanding mode the reply will contain an updated joint position set-point which the robot will track, effectively allowing the iiwaDrive to execute trajectories.

The ZeroMQ implementation of the *Publish(PUB)-Subscribe(SUB)* pattern has been utilized to share the joint position of the robot with the external application. In the PUB-SUB pattern, the publisher is sending a constant stream of messages that a number of subscribers can listen to. The PUB-SUB pattern is useful since the publish operation is asynchronous in the ZeroMQ implementation. The asynchronous property means that the FRI-thread will not become blocked whenever it has to share the current state of the robot. Every cyclic message that the FRI thread receives contains the current joint position of the robot. Before a reply is sent back to the robot, the current joint positions are published on the ZeroMQ PUB socket. This allows the external application to subscribe to a stream of joint values whenever it is desired. Multiple subscribers can subscribe to the stream of joint positions if desired, but in this thesis, only one has been evaluated. The PUB-SUB pattern is very useful when synchronizing the joint values of the virtual representation with the real robot. Whenever synchronization is desired, the external application holding the virtual representation subscribe to the joint values of the real robot and update the graphical representation accordingly at a fast rate.

The instruction handler thread is constantly waiting for incoming instruction on a REQ-REP socket. An external application is able to connect and send requests in the shape of instructions to the iiwaDrive. As previously mentioned, the instructions can either be a set of high-level motion instructions, trajectory execution request, sensor readings request, IO get or IO set. The iiwaDrive can

enter into three state *idle*, *runningMotion* and *runningTrajectory*. The *idle* state is adopted during initialization. This state is mainted until a request to execute a set of high-level motion instructions or a request to execute a trajectory is received from the external application. If a request to execute a set of high-level motion instructions is received, the iiwaDrive enters the *runningMotion* state. The samples received in the FRI thread is now buffered locally, as well as published. The instruction handler thread forwards the high-level motion instructions to the robot program which acknowledge the message and starts to execute the motions. When the motions have been executed, the robot program signals a state changed to the iiwaDrive. The buffered samples are now sent back as a reply to the external application before the state of the application become *idle* again.

If a trajectory execution request is received by the instruction handler thread, the application enters in to the *runningTrajectory* state. The timestamped joint positions are fed to the FRI thread and an instruction to start the trajectory execution is passed to the robot program. The trajectory is executed and the samples received by the FRI thread is buffered locally. When the trajectory is done executing, instruction handler thread signal to the robot program to enter the *idle* state again. Before the iiwaDrive enters the *idle* state, the buffered samples are sent back as a reply to the external application.

Whenever a sensor reading or IO request is received by the instruction handler thread, they are simply forwarded to the robot program. The robot program either execute the IO set or answers with the resulting sensor reading or IO state.

3.1.3 External application

The messages that are exchanged between the iiwaDrive and the external applications have been defined using the JSON format. The JSON format is a language independent way of defining data in a human-readable way. The idea is to keep the communication with the iiwa robot to any outside application generic as much as possible. If one wants to use another communication library instead of ZeroMQ to communicate with the iiwaDrive it is possible as long as the defined JSON format is kept.

An external application can connect communicate with the iiwaDrive through the instruction REP-REQ socket and receive a constant stream of joint position of the robot through the PUB-SUB socket. This allows for remote control of the robot as well as monitoring and state synchronization of the virtual representation.

4 Evaluation of Blending Methods

To be able to simulate the motion of the iiwa robot, a good match between the geometrical path in simulation and the real robot controller is required. In this section, it is therefore evaluated how well the blending methods described in section 2.3.3 is able to capture the geometrical path of the real iiwa robot. Based on the result from the evaluation, a extension to the described blending methods is presented. Firstly, the high level motion commands possible for the iiwa robot is translated to be able to calculate the geometrical path of the blends.

4.1 iiwa path representation

A PTP segment for the iiwa robot is calculated utilizing Eq. (10). There exist two ways of defining a motion blend for the iiwa robot, *Cartesian blend* (b_r in mm) and a *relative blend* ($b_\% \in [0, 0.5]$). The *Cartesian blend* defines a sphere around the point c_i while the *relative blend* defines a percentage in configuration space before the joint position that is to be blended away.

The start, $\boldsymbol{\theta}_{i,0}$, and the end, $\boldsymbol{\theta}_{i,1}$ defined with a *relative blend*, $b_\%$, can easily be found as

$$\begin{aligned} \boldsymbol{\theta}_{i,0} &= \boldsymbol{\theta}_{i-1} + (1 - b_\%)(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}) \\ \boldsymbol{\theta}_{i,1} &= \boldsymbol{\theta}_i + b_\%(\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i). \end{aligned} \tag{36}$$

When it comes to determining the start and end of the *Cartesian blend*, Algorithm 1 could be adopted. This algorithm can although not guarantee that the right intersection with the sphere is found. This is due to the fact that the sphere is defined in operational space while the path is interpolated in configuration space. Algorithm 1 has therefore been modified to search along the incoming and outgoing segments respectively until an intersection is found. This is done by incrementally decreasing s until the first intersection is found closest to the blending position c_i. As long as two intersections do not occur for one increment the right intersection will be found. While this algorithm cannot guarantee that the right intersection is found, it has been deemed more likely to succeed. This simple algorithm is given in Algorithm 2. To limit two consecutive blend to not overlap, s is limited to 0.5.

When Algorithm 2 has been applied to the incoming segment, $\boldsymbol{P}_i(s)$, and the reversed outgoing segment, $\boldsymbol{P}_{i+1}(s)$, the joint-values of the intersection can be found

Algorithm 2 Determines where translation of the end effector defined by the segment $\boldsymbol{P}(s)$ intersects a sphere with the radius b_r using incremental decrease of s.

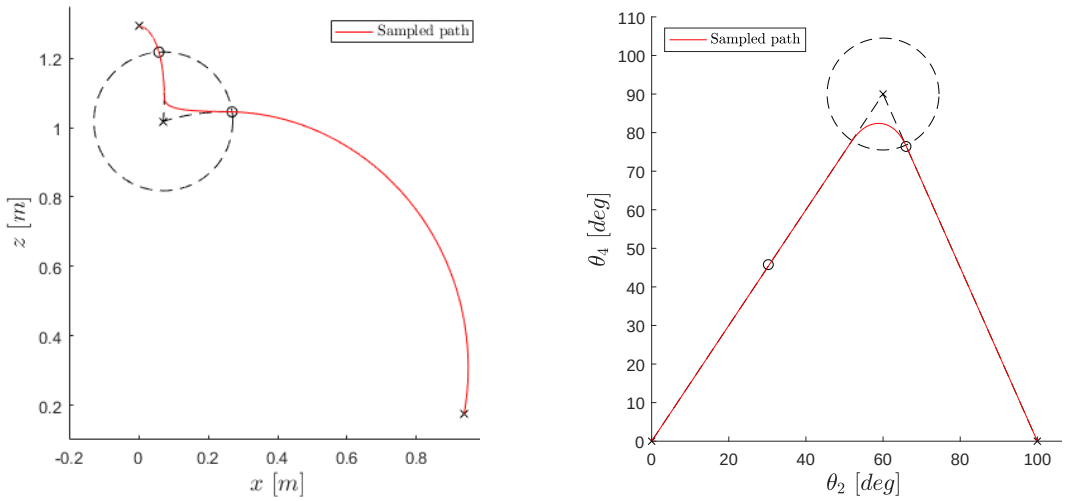
Require: $(\boldsymbol{P}(1) = \boldsymbol{\theta}_i) \wedge (b_r > 0)$
1: $c_i \leftarrow \boldsymbol{T}_E^0(\boldsymbol{\theta}_i)$ Eq. (9)
2: $\Delta c \leftarrow 0$
3: $s_{b_r} \leftarrow 1$
4: **while** $\Delta c < b_r$ **do**
5: $\quad s_{b_r} \leftarrow s_{b_r} - 0.001$
6: $\quad \Delta c \leftarrow ||c_i - \boldsymbol{T}_E^0(\boldsymbol{P}(\boldsymbol{s_{b_r}}))||$
7: **end while**
8: **if** $s_{b_r} < 0.5$ **then**
9: $\quad s_{b_r} \leftarrow 0.5$
10: **end if**
11: **return** s_{b_r}

$$\begin{aligned} \boldsymbol{P}_i(s) &= \boldsymbol{\theta}_{i-1} + s(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}) \\ \boldsymbol{P}_{i+1}(s) &= \boldsymbol{\theta}_{i+1} + s(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1}) \\ \boldsymbol{\theta}_{i,0} &= \boldsymbol{P}_i(s_{b_r,i}) \\ \boldsymbol{\theta}_{i,1} &= \boldsymbol{P}_{i+1}(s_{b_r,i+1}). \end{aligned} \tag{37}$$

How well these methods of calculating the start and stop of the blend given b_r or $b_{\%}$ applies to the iiwa robot has been experimentally tested. A path consisting of two PTP instructions where only joint two and four are moved has been tested. For the middle PTP instruction, the robot is allowed to blend with a b_r of 200 mm. The resulting end effector path of the real iiwa robot can be seen in Fig. 13a. The circles in Fig. 13a represents the start and stop of the blend if Algorithm 2 is used. It can be seen that it does not capture the start and stop of the blend since the blend does not start where the circles are. If one instead looks at the path in configuration space, Fig. 13b, one can see that the blending of the sampled path from the real iiwa robot starts and stops at an absolute distance $\Delta\theta$ from the point $\boldsymbol{\theta_i}$. One can therefore conclude that the shortest absolute distance will control where the blending starts and stops if a match with the real iiwa robot path is desired.

This behavior can easily be achieved by scaling the longest absolute distance of the two. Given the joint position to be blended away, $\boldsymbol{\theta}_i$, and an initial start and stop of the blend calculated either with Eq. (36) or (37), the updated start and stop position of the blend can be found using Algorithm 3.

If Algorithm 3 is applied to the test case mentioned above, where the blend starts and stops are captured accurately as can be seen in Fig. 14. It is now known at what joint configuration the blend starts, $\boldsymbol{\theta}_{i,0}$, and stops, $\boldsymbol{\theta}_{i,0}$.



(a) End effector path during experiment with a blend b_r of 200 mm. The circles represents the calculated start and stop of the blend, $c_{i,0}$ and $c_{i,1}$.

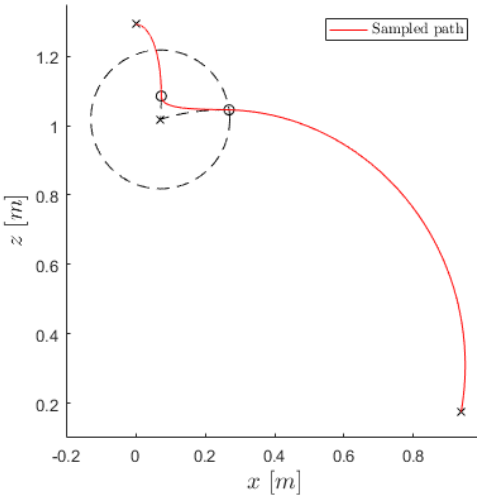
(b) Corresponding path in configuration space. The circles represents the calculated start and stop of the blend in configuration space $\boldsymbol{\theta}_{i,0}$ and $\boldsymbol{\theta}_{i,1}$

Figure 13: Resulting path of two PTP instructions sampled on the real iiwa robot.

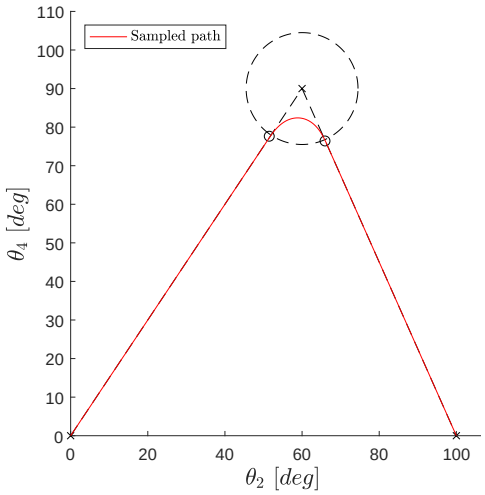
Algorithm 3

Require: $\boldsymbol{\theta}_i$, & $\boldsymbol{\theta}_{i,0,initial}$, & $\boldsymbol{\theta}_{i,1,initial}$

$\Delta\theta_{i,0} \leftarrow ||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0,initial}||$
$\Delta\theta_{i,1} \leftarrow ||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,1,initial}||$
if $\Delta\theta_{i,1} \geq \Delta\theta_{i,0}$ **then**
    $\boldsymbol{\theta}_{i,0} \leftarrow \boldsymbol{\theta}_{i,0,initial}$
    $\boldsymbol{\theta}_{i,1} \leftarrow \boldsymbol{\theta}_i + \Delta\theta_{i,0} \frac{\boldsymbol{\theta_{i,1,initial}} - \boldsymbol{\theta_i}}{||\boldsymbol{\theta_{i,1,initial}} - \boldsymbol{\theta_i}||}$
else
    $\boldsymbol{\theta}_{i,0} \leftarrow \boldsymbol{\theta}_i + \Delta\theta_{i,1} \frac{\boldsymbol{\theta}_{i,0,initial} - \boldsymbol{\theta_i}}{||\boldsymbol{\theta}_{i,0,initial} - \boldsymbol{\theta_i}||}$
    $\boldsymbol{\theta}_{i,1} \leftarrow \boldsymbol{\theta}_{i,1,initial}$
end if
return $\boldsymbol{\theta}_{i,0}$ & $\boldsymbol{\theta}_{i,1}$



(a) The circles represent the calculated start and stop of the blend, $c_{i,0}$ and $c_{i,1}$



(b) The circles represents the calculated start and stop of the blend in configuration space $\boldsymbol{\theta}_{i,0}$ and $\boldsymbol{\theta}_{i,1}$

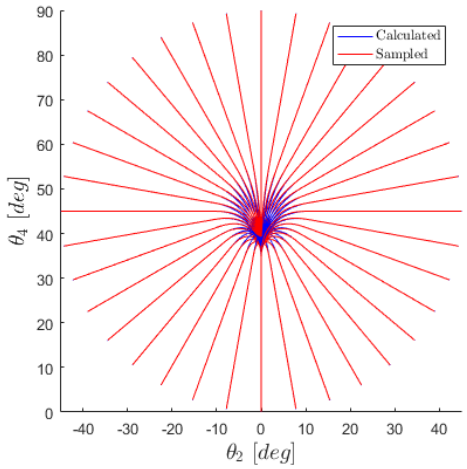
Figure 14: Result after applying algorithm 3. The circles represents the updated start and stop of the blend.

4.2 Evaluating existing methods

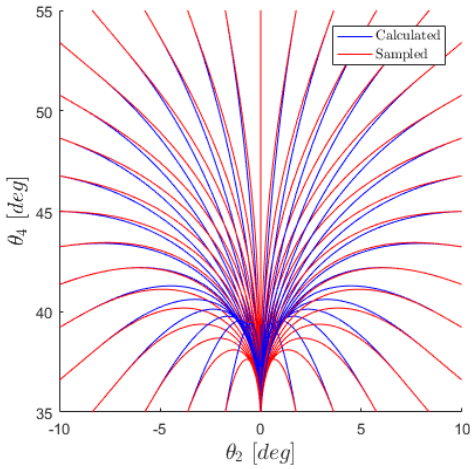
Once the start and stop of the blend is know, it is possible to calculate the shape of the blend. In chapter 2.3.3, three existing methods in the literature for calculating the geometrical path during the blend has been described. These consist of Eq. (13) where a polynomial $\alpha(s)$ is used, Eq. (26) where the polynomial $\alpha(s)$ together with a scaled polynomial, $\beta(s)$ that minimizes the acceleration and Eq. (32) where a the blend follows the perimeter a circle.

It is desired to achieve an as accurate geometrical path as possible to ensure an accurate simulation. An experiment has been conducted to evaluate if any of these methods is capable of achieving a similar geometrical path during the blend as the real iiwa robot. This has been done by instructing the robot with two PTP instructions where only joint two and four are moving while sampling the real path. This allows one to easily visualize the path in configuration space. The first PTP instruction is defined with a *relative blend*, $b_{\%}$, of 0.25. The second PTP instruction is varied in a circular fashion around the point to be blended away resulting in 36 blending motions.

How the calculated blend utilizing Eq. (13) captures the blending motion can be seen in Fig. 15. This method utilizes a blending polynomial $\alpha(s)$ to achieve a smooth transition between the consecutive segments. It is clear that this method fails to capture geometrical path during the blend, there is a clear deviation between the calculated and the sampled path.



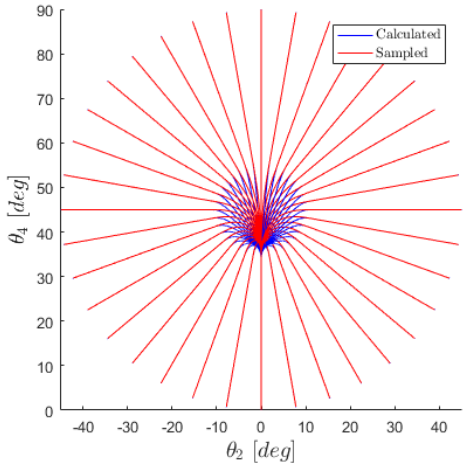
(a) Overview in configuration space of the resulting paths.



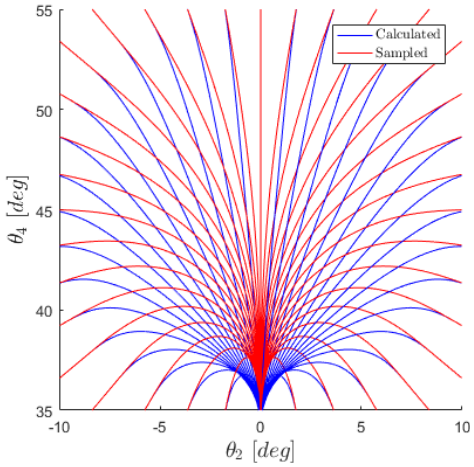
(b) Enlarged view of the blending motions.

Figure 15: Comparison between the calculated blend utilizing Eq. (13) and the sampled path.

How the calculated blend utilizing Eq. (26) captures the blending motion can be seen in Fig. 16. This methods utilizes a blending polynomial $\alpha(s)$ as well as a polynomial $\beta(s)$ that theoretically minimizes the average acceleration during the blending motion if κ is chosen as $\frac{15}{2}$. This method fails as well to capture the geometrical path of the real robot.



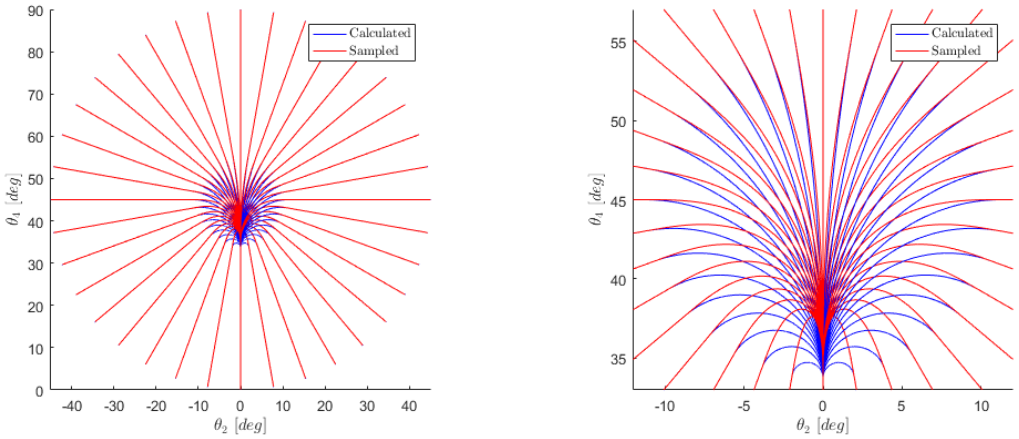
(a) Overview in configuration space of the resulting paths.



(b) Enlarged view of the blending motions.

Figure 16: Comparison between the calculated blend utilizing Eq. (26) and the sampled path.

How the calculated blend utilizing Eq. (32) captures the blending motion can be seen in Fig. 17. It is clear that this method fails to reproduce the path of the real robot. The sampled path does not follow a circular path as can be seen in Fig. 17b.



(a) Overview in configuration space of the resulting paths.

(b) Enlarged view of the blending motions.

Figure 17: Comparison between the calculated blend utilizing Eq. (32) and the sampled path.

4.3 Extension of existing method

None of the evaluated methods are able to reproduce the same path as the real robot. Out of the methods evaluated, only the method described by Eq. (26) offers the possibility to control the shape of the blend. This can be done by varying the free variable k. If one analyzes Fig. 15b, it can be seen that the calculated path at some instances overshoots and at others undershoots the sampled path. This behavior varies depending on the angle between the incoming and outgoing segment. Varying k will not affect the continuity requirements, Eq. (20), since $\beta(s)$ it zero at the beginning and end of the blending segment. The fact that the continuity constraints for different values of k are not violated can be seen in Fig. 18.

Varying k will have the effect of guiding the blend closer or further away from the point to be blended away, $\boldsymbol{\theta}_i$, since it will scale the vector $((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0}))$, Fig. 19.

It is therefore proposed to control the shape of the blend by varying k depending on the angle between the incoming and outgoing segment. Let κ define a magnitude that is dependent on the angle γ_i. A relationship between the scaling magnitude κ and the angle γ_i has been experimentally determined by manually tuning and observing the values that result in the highest correspondence between the sample and calculated path. The experiment was performed on the path used for evaluating the blending methods. The observed relationship is shown in Fig. 20.

It can bee seen that there is a relationship between the scaling κ and the angle γ_i. This relationship can be modeled with a second degree polynomial

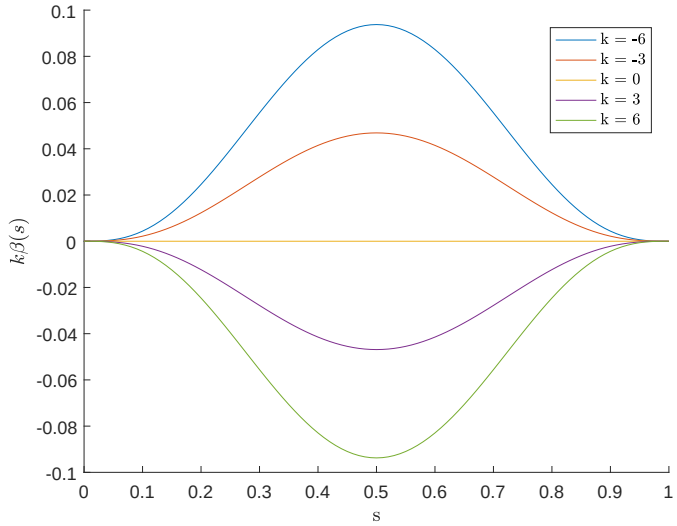


Figure 18: Effect of scaling the polynomial $\beta(s)$ with different values of k.

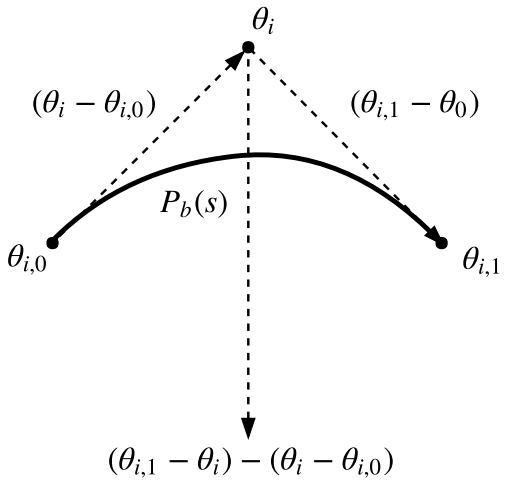


Figure 19: When the value of κ is varied, it will guide the blend along the vector $((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0}))$.

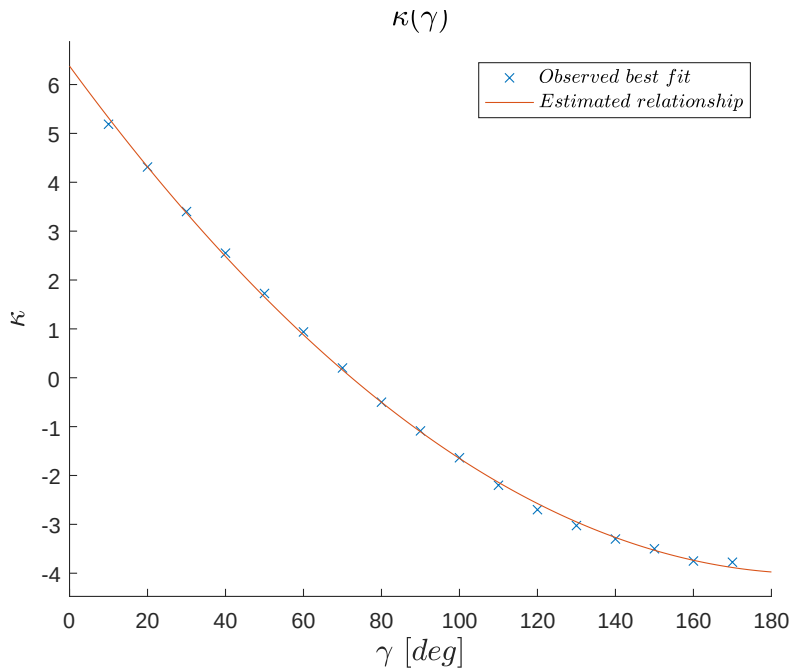


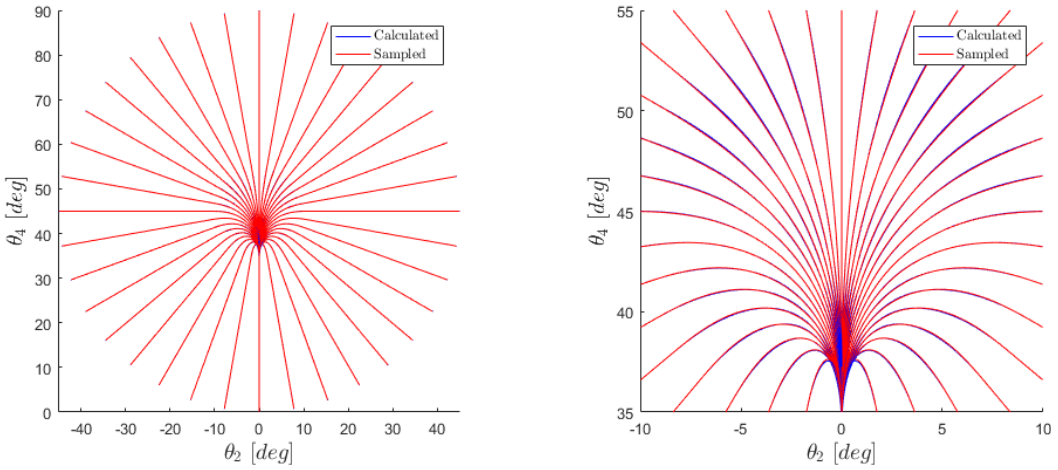
Figure 20: Manually tuned κ that results in the highest correspondence between the sample and calculated path

$$\kappa(\gamma_i) = 0.00028472\gamma_i^2 + 0.00626016\gamma_i - 3.97426471 \tag{38}$$

It is therefore proposed that the blending motion for the iiwa robot can be calculated as follows

$$\begin{aligned}\boldsymbol{P}_b(s) =& \boldsymbol{P}_0(s) + \alpha(s)(\boldsymbol{P}_1(s) - \boldsymbol{P}_0(s)) \\ &- \kappa(\gamma)\beta(s)\left((\boldsymbol{\theta}_{i,1} - \boldsymbol{\theta}_i) - (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i,0})\right).\end{aligned} \tag{39}$$

The proposed method in Eq. (39) is capable accurately describing the geometrical path of the iiwa robot during the blend as can be seen in Fig. 21. There are some differences between sampled and calculated path as can be seen in Fig. 21b, but utilizing this proposed method, the geometrical path can more accurately be described compared to the methods in the literature.



(a) Overview in configuration space of the resulting paths.

(b) Enlarged view of the blending motions.

Figure 21: Comparison between the calculated blend utilizing Eq. (39) and the sampled path.

5 Results

In this section, the proposed method for simulating the movements is evaluated, this includes combining the geometrical path calculation described in the previous chapter with the trajectory generation to estimate the cycle time. The virtual representation and communication with the real robot through the iiwaDrive has been implemented in the *IPS*-software developed by Fraunhofer-Chalmers Centre[32]. A brief conceptual utilization case of the developed capabilities of simulating motions and interfacing with the real robot is lastly demonstrated.

5.1 Evaluation of proposed simulation method

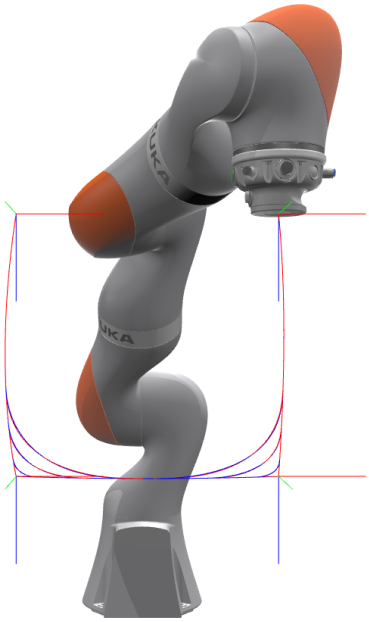
The approach for simulating the movements has been implemented in the IPS-software. The interface with the iiwaDrive has been implemented which allows one to execute motions and sample motions from the IPS software. These capabilities have been utilized to validate the simulation method against the motion of the real robot.

5.1.1 Geometric end-effector path estimation

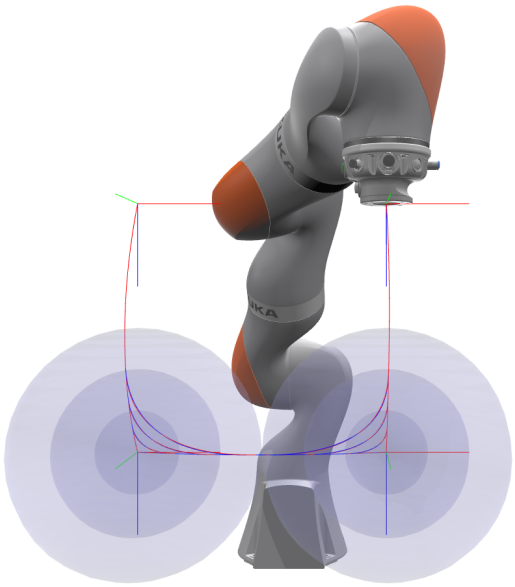
A comparison between the estimated and real geometric path of the end-effector can be seen in Fig. 22. The estimated path was calculated using the proposed method described in section 4.3. The motion program is made up of three PTP instructions. Fig. 22a shows the comparison when two of the three PTP instruction are defined together with a *relative blend* and Fig 22b show the result when two of the three PTP instruction are defined together with a *Cartesian blend*. It can be seen in Fig. 22 that the geometrical path of the robot is captured well.

5.1.2 Cycle time estimation

The proposed simulation capability in terms of cycle-time has been evaluated by looking at the difference in simulated cycle-time, T_s, and the cycle time when the real robot executes the same motion program, T_r. Five motion programs consisting of ten PTP instructions each were randomly generated. The evaluation has been set up so that the motion programs have three degrees of freedom, relative blend($b_{\%}$), relative velocity($v_{\%}$) and path length(λ). The length between every consecutive joint move instruction can be controlled with the path length parameter λ that has been introduced for the experiment. A low λ-value results in short paths while a larger value results in longer paths.



(a) Relative blend $b_{\%} \in [0,\ 0.1,\ 0.3,\ 0.5]$



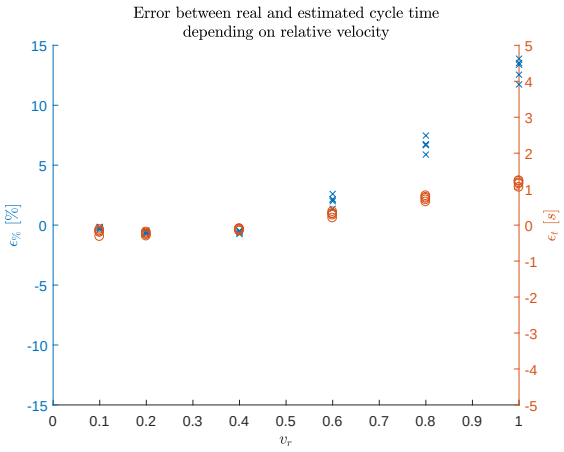
(b) Cartesian blend $b_r \in [0,\ 50mm,\ 100mm,\ 150mm]$. The circle represents a sphere with radius b_r.

Figure 22: TCP trace when blending the segments with *relative blend* and *Cartesian blend*. The blue path is the calculated path and the red is the real robot path.

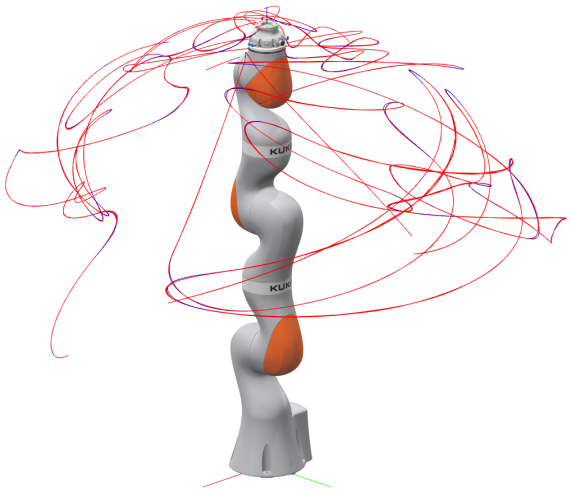
The error in cycle-time has been evaluated as a time difference, ϵ_t, and as a percentage, $\epsilon_{\%}$

$$\begin{aligned} \epsilon_t &= T_s - T_r \\ \epsilon_{\%} &= \frac{\epsilon_t}{T_r}. \end{aligned} \tag{40}$$

The effect of varying the relative velocity can be seen in Fig. 23 for a given path length and relative blend. The error, ϵ_t, stays around 0% for low relative velocities while for high relative velocities it differs as much as 15%. This is expected since running the robot at low relative velocity, the robot is going to maintain its allowed maximum relative velocity for the majority of the path and therefore not a minimal amount of time will be spent decelerating and accelerating. If the estimated geometrical path is close to the real path, the robot will maintain a low constant velocity along the path resulting in a good estimation. For high relative velocities, the time spent accelerating and decelerating throughout the blend can be an explanation for the difference in cycle time. The utilized trajectory generation algorithm maintains the acceleration within the specified limits, but higher order constraints such as jerk and torque imposed by the robot controller can cause the two to differentiate.



(a) The error, ϵ, in terms of percentage and seconds.



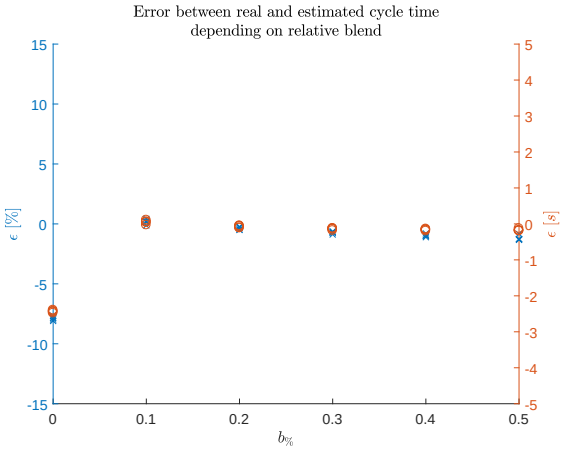
(b) Resulting TCP path. The blue path is the simulated path and red is the sampled from the real robot.

Figure 23: How a varying relative velocity affects the error in cycle time estimation given a fixed path length ($\lambda = 200$) and relative blend ($b_\% = 0.25$) for 5 random paths.

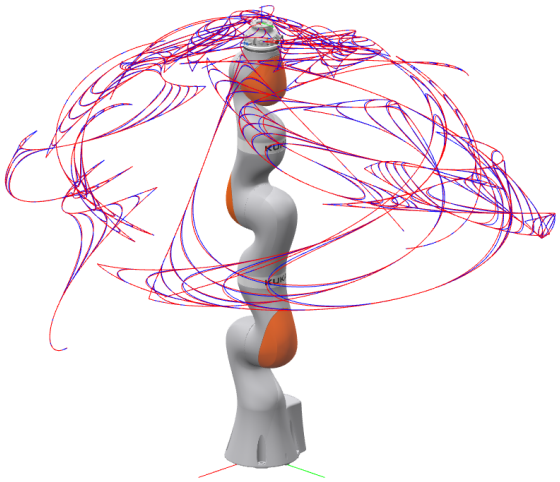
The effect of varying the relative blend can be seen in Fig. 24 for a given path length and relative velocity. When no blend is used, meaning that the geometrical paths are identical, the simulated cycle time is less than cycle time of the real robot. This is expected since the real robot considers the effects of limited jerk during trajectory calculation. As the blend becomes bigger, the simulated cycle time is close to the cycle time of the real robot. When blending occurs there is a relatively small change in velocity between consecutive instructions, this allows the robot to maintain is velocity which results in a good cycle time estimation.

The effect of varying the path length can be seen in Fig. 25 for a given relative blend and relative velocity. The error, ϵ_t, stays constant within 2% when varying the path length for longer paths. For shorter paths, there is a marginally bigger difference in the cycle time estimation. When the path is shorter, it will cause the acceleration and deceleration phase to make up a larger portion of the total cycle-time. The fact that jerk is not considers is a plausible cause for this behaviour.

In summary, the cycle-time can be estimated for low velocities and when blending is present within a few percent. The worst cycle-time estimation is obtained when motions are programmed with high velocities and no blending.



(a) The error, ϵ, in terms of percentage and seconds.



(b) Resulting TCP path. The blue path is the simulated path and red is the sampled from the real robot.

Figure 24: How a varying relative blend affects the error in cycle time estimation given a fixed path length ($\lambda = 200$) and relative velocity ($v_{\%} = 0.4$) for 5 random paths.

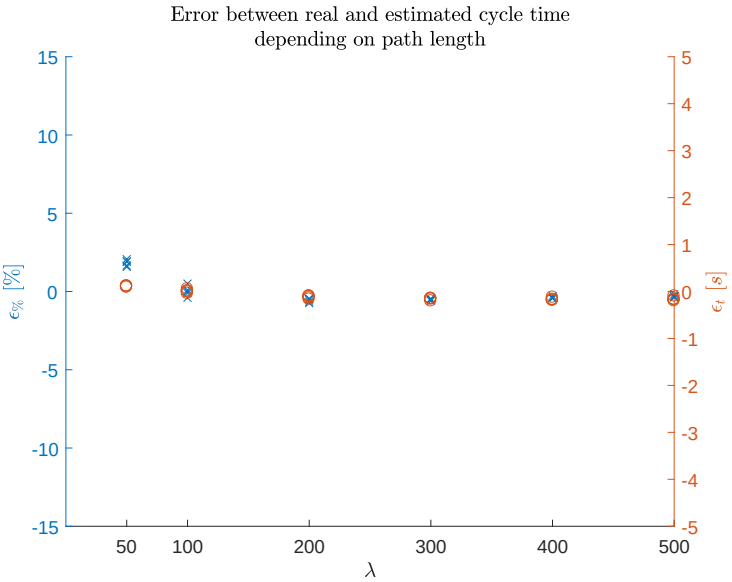
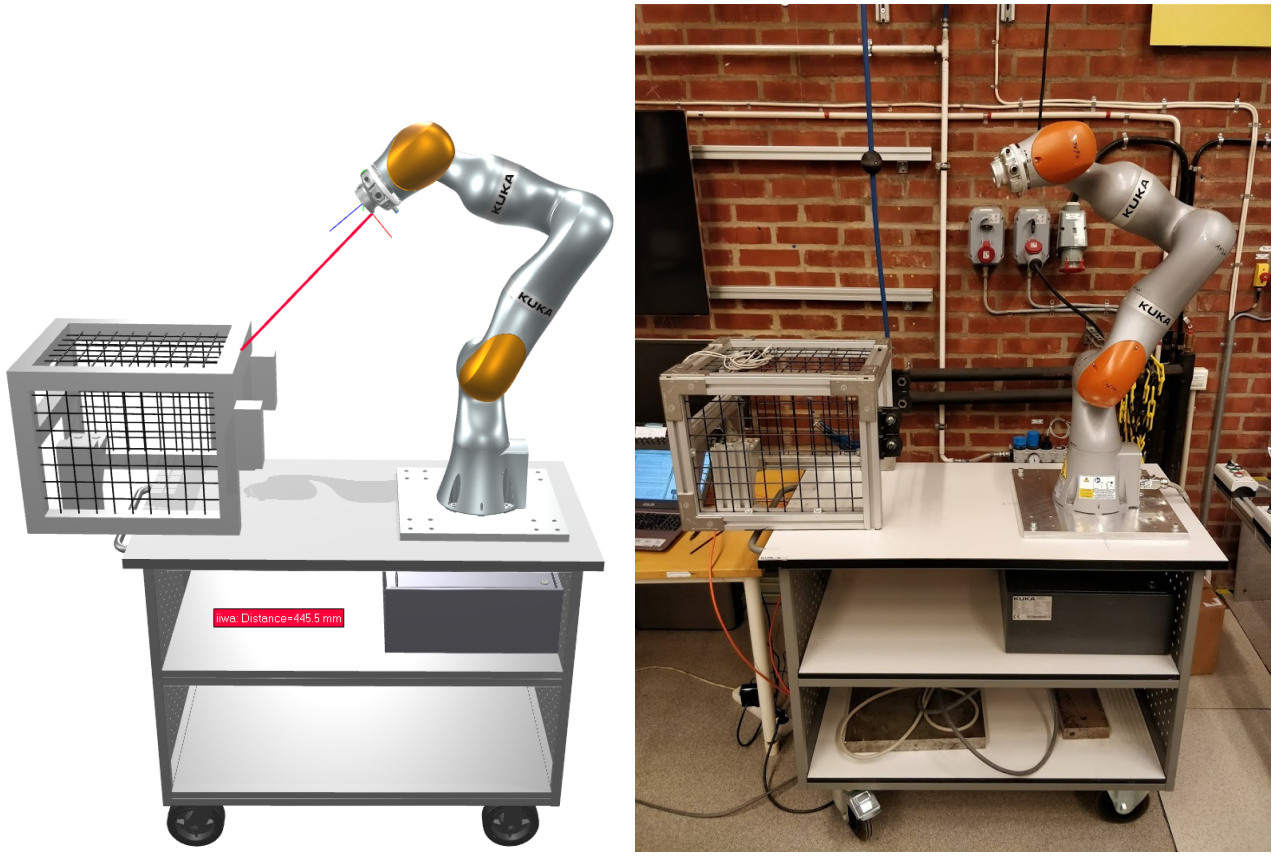


Figure 25: How a varying path length affects the error in cycle time estimation given a fixed relative blend ($b_{\%} = 0.25$) and relative velocity ($v_r = 0.4$) for 5 random paths

5.2 Demonstration of capabilities within a digital twin

To put everything in to context, a short demo has been created in the IPS software to highlight the results. A digital twin of a real-world setup has been modeled in the IPS software. The digital twin contains CAD representations of the environment as well as the iiwa robot with the developed capabilities. The virtual representation can be seen in Fig. 26a and the physical counterpart can be seen in Fig. 26b.



(a) Virtual representation rendered in IPS.

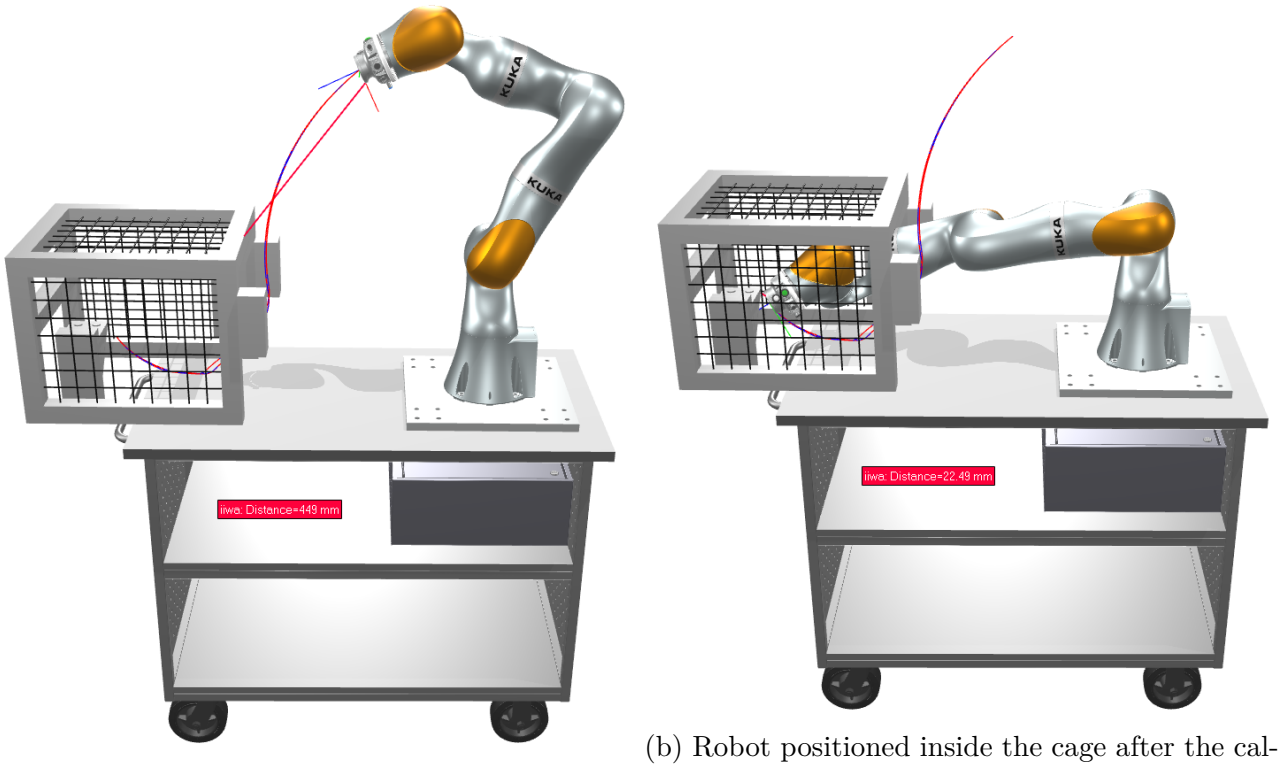
(b) Real iiwa robot.

Figure 26: Digital twin example.

The joint position of the virtual representation can be synchronized with the real robot through the developed iiwaDrive joint value streaming interface. When the robot moves in reality, it is continuously updated in IPS. This enable one to utilize the functionality of IPS such as real time distance measures between the robot and environment and perform collision-free path planning.

When the robot moves in reality, the virtual representation of the robot is updated in IPS. Positions of the robot can be saved in the software and collision-free robot programs can be calculated and simulated. When the simulation outcome is satisfactory, the robot program can be sent to the robot and executed. The resulting trajectory from the real robot is sent back to the software

and can be replayed if desired. An example is shown in Fig. 27 where the robot goes from a position outside the cage to a position inside the cage. The motion can be simulated with the developed method and verified before it is executed.



(a) Robot positioned outside the cage.

(b) Robot positioned inside the cage after the calculated final position.

Figure 27: Example where a the robot is moved inside the cage.

6 Conclusion

During this work, the needed capabilities to allow the KUKA iiwa robot to work within a digital twin manufacturing system has been developed. This includes adapting existing methods for simulating the movements resulting from high-level motion commands. An application named iiwaDrive has been developed that enables external application to communicate with the real robot. The developed features include being able to synchronize virtual representation with the state of the real robot as well as allowing external applications to execute high-level motion commands on the real robot. The proposed interface with the real robot has been successful in allowing external monitoring of the real robot as well as allowing external applications to interface with the real robot. By utilizing the FRI protocol together with ZeroMQ, external monitoring at down to 1 ms is possible.

Existing methods for calculating the geometrical path has been evaluated and built upon to create a method for describing the geometrical path of the iiwa robot. Utilizing this method it is possible to achieve accurate estimations of how the robot will move geometrically based on the given commands.

The evaluation performed in terms of estimating cycle time shows that the methods utilized to simulation the motion is capable of estimating the cycle time of the real robot for a given motion program defined with PTP instructions within 15%. No consideration to jerk or torque limits of the manipulator has been considered. This is a drawback of the utilized trajectory generation method since it fails to capture the cycle-time when the robot is programmed with high velocities.

6.1 Future work

No tool has been considered to be mounted on the robot during this work. The extra load on the robot will most likely cause the internal robot trajectory generator to at some degree limit the velocity and acceleration during the motion. One important future aspect to consider is therefore how this behaviour can be included in the cycle time estimation.

Only PTP instructions in the configuration space has been considered during this work. A natural extension is to include motions defined in the operational space such as linear and circular movements of the end-effector.

When the robot is operating, large amount of data is constantly being sent from the iiwaDrive application in the form of joint value positions. ZeroMQ might not be ideal if one wants to share and keep track of large amounts of data. It might therefore be of interest to integrate the developed iiwaDrive

to communicate with external applications using a messaging software such as *Apache Kafka* [33] which is capable of efficiently streaming data as well as storing if over time.

References

[1] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, "Reengineering aircraft structural life prediction using a digital twin," *International Journal of Aerospace Engineering*, vol. 2011, 2011.

[2] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, and L. Wang, "Modeling, simulation, information technology & processing roadmap," *National Aeronautics and Space Administration*, 2012.

[3] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017.

[4] R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 567–572, 2015.

[5] S. Boschert and R. Rosen, "Digital twin—the simulation aspect," in *Mechatronic Futures*. Springer, 2016, pp. 59–74.

[6] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, p. 1101, 2012.

[7] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist, "Toward a digital twin for real-time geometry assurance in individualized production," *CIRP Annals-Manufacturing Technology*, 2017.

[8] R. Bohlin, J. Hagmar, K. Bengtsson, L. Lindkvist, J. S. Carlson, and R. Söderberg, "Data flow and communication framework supporting digital twin for geometry assurance," in *ASME 2017 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2017, pp. V002T02A110–V002T02A110.

[9] R. Bernhardt, G. Schreck, and C. Willnow, "Realistic robot simulation," *Computing & Control Engineering Journal*, vol. 6, no. 4, pp. 174–176, 1995.

[10] KUKA. Lbr iiwa series. [Online]. Available: https://www.kuka.com/en-my/products/robotics-systems/industrial-robots/lbr-iiwa.html

[11] S. Y. Nof, *Handbook of industrial robotics*. John Wiley & Sons, 1999, vol. 1.

[12] H. David, "Why are there still so many jobs? the history and future of

workplace automation," *Journal of Economic Perspectives*, vol. 29, no. 3, pp. 3–30, 2015.

[13] E. Wernholt, "Multivariable frequency-domain identification of industrial robots," Ph.D. dissertation, Institutionen för systemteknik, 2007.

[14] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[15] L. Sciavicco and B. Siciliano, *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.

[16] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014.

[17] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[18] J. Wu, J. Wang, and Z. You, "An overview of dynamic parameter identification of robots," *Robotics and computer-integrated manufacturing*, vol. 26, no. 5, pp. 414–419, 2010.

[19] D. Forsman, "Bangenerering för industrirobot med 6 frihetsgrader," 2004.

[20] J. Lloyd and V. Hayward, "Trajectory generation for sensor-driven and time-varying tasks," *The International journal of robotics research*, vol. 12, no. 4, pp. 380–393, 1993.

[21] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson-/Prentice Hall Upper Saddle River, NJ, USA:, 2005, vol. 3.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[23] KUKA. Sunrise.os. [Online]. Available: https://www.kuka.com/en-se/products/robotics-systems/software/system-software/sunriseos.html

[24] R. S. Hartenberg and J. Denavit, *Kinematic synthesis of linkages*. McGraw-Hill, 1964.

[25] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[26] T. Ustyan and V. Jönsson, "Implementation of a generic virtual robot controller," *Master, Chalmers University of Technology*, 2011.

[27] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, 2012.

[28] D. Gleeson, S. Björkenstam, R. Bohlin, J. S. Carlson, and B. Lennartson, "Towards energy optimization using trajectory smoothing and automatic code generation for robotic assembly," *Procedia Cirp*, vol. 44, pp. 341–346, 2016.

[29] KUKA. Sunrise sunrise cabinet. [Online]. Available: https://www.kuka.com/en-se/products/robotics-systems/robot-controllers/kuka-sunrise-cabinet.html

[30] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the kuka lightweight robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010)*, 2010, pp. 15–21.

[31] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[32] Fraunhofer-Chalmers Centre. IPS. [Online]. Available: http://www.fcc.chalmers.se/software/ips

[33] N. Garg, *Apache Kafka*. Packt Publishing Ltd, 2013.

Appendix A Acceleration and jerk limit experiment

The limit on the joint acceleration and joint jerk has been determined experimentally. This was done by placing the robot at different poses and individually commanding the joints to go from -20 degrees to +20 degrees relative to the point at max speed. The commanded joint values were sampled using FRI and numerically differentiated to obtained acceleration and jerk values for the movement. An example can be seen in Fig. 28 where a movement for joint 1 is tested.

Depending on the joint positions, the torque acting on the joints will be different. The experiment is therefore performed for 20 initial positions, given in Table 3, to see if it affects the maximum acceleration and jerk. The results are presented in Table 4 and Table 5. The experiment shows that the commanded max acceleration remain the same independent of the initial position. Notable is that no tool was mounted on the robot during the experiments which might effect the result.

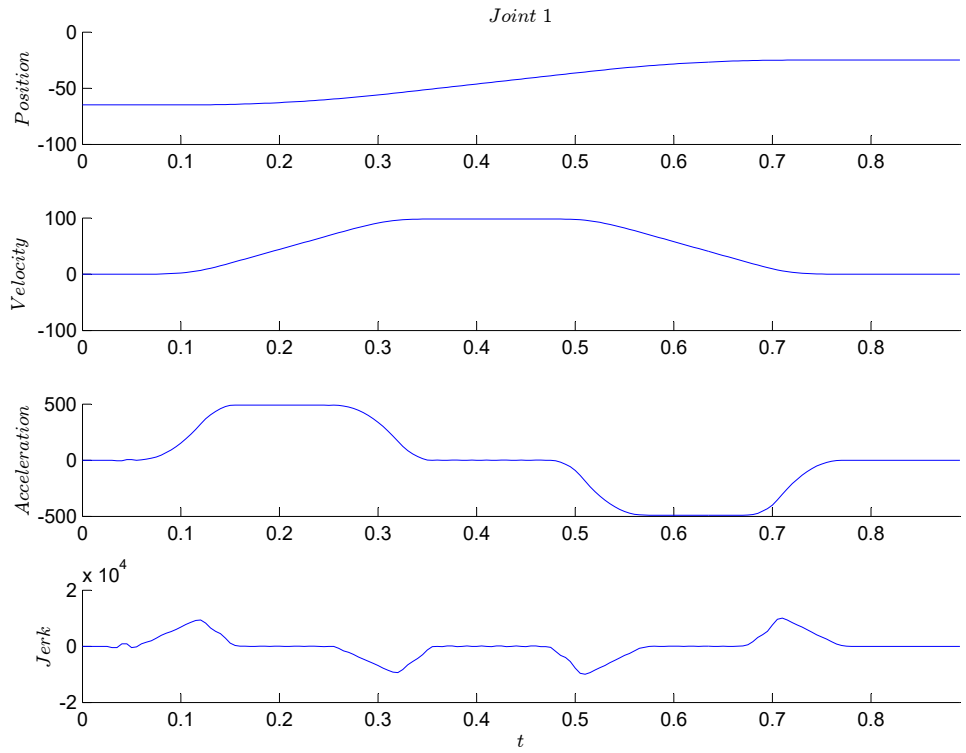


Figure 28: Example experiment for joint 1.

Table 3: Initial joint positions during experiment

	Initial joint position [Deg]						
Run	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
1	-44.91	27.72	96.85	49.32	-97.77	71.79	65.26
2	4.06	-13.72	-145.50	-81.72	-40.66	-70.54	-103.57
3	146.56	-3.80	-114.28	-99.07	-147.33	-24.42	9.82
4	21.36	7.12	32.15	-66.75	48.91	-9.84	-45.84
5	-132.89	7.54	85.00	60.52	5.96	-39.61	116.55
6	68.00	31.91	127.72	7.87	-107.30	-7.58	-82.05
7	108.67	-20.33	83.90	68.73	149.04	99.94	34.56
8	-32.27	-16.37	-60.82	68.03	-142.88	-24.83	-126.29
9	53.16	-31.06	-147.36	83.76	-67.23	-45.42	27.25
10	57.35	23.63	67.95	-3.01	-88.39	48.75	-9.78
11	-12.61	31.44	73.33	-78.34	29.71	-22.95	72.85
12	32.69	5.07	-41.60	-69.69	-82.47	-14.97	93.89
13	5.13	34.30	75.46	-30.89	-99.31	31.46	-2.51
14	-130.94	13.98	1.44	-70.50	134.88	-71.68	125.59
15	57.87	-13.79	-22.03	-85.92	139.98	36.64	-107.50
16	113.18	22.52	24.61	-61.73	-96.63	63.44	-7.67
17	-103.33	0.27	69.61	-18.88	-66.13	13.75	56.49
18	76.76	15.53	-7.41	-75.40	-39.66	66.94	-144.12
19	5.10	11.41	-22.13	-79.06	134.80	84.28	15.36
20	-46.20	-1.98	-37.51	69.40	-54.94	-8.78	-70.71

Table 4: Measured maximum acceleration

	Max Acceleration [Deg/s^2]						
Run	$\ddot{\theta}_1$	$\ddot{\theta}_2$	$\ddot{\theta}_3$	$\ddot{\theta}_4$	$\ddot{\theta}_5$	$\ddot{\theta}_6$	$\ddot{\theta}_7$
1	490.45	491.02	500.77	650.31	700.73	900.69	900.69
2	490.45	490.45	500.77	650.88	700.73	900.69	900.69
3	490.45	491.02	500.77	650.31	700.73	900.69	900.69
4	491.02	491.02	500.77	650.88	700.73	900.12	900.69
5	491.02	490.45	500.77	650.88	700.73	900.69	900.69
6	490.45	491.02	500.77	650.88	700.73	900.69	900.69
7	491.02	491.02	500.77	650.88	700.73	900.69	900.69
8	491.02	490.45	500.77	650.88	700.73	900.69	900.69
9	491.02	491.02	500.77	650.88	700.73	900.69	900.69
10	491.02	490.45	500.77	650.88	700.73	900.69	900.69
11	491.02	491.02	500.77	650.31	700.73	900.69	900.69
12	491.02	490.45	500.77	650.88	700.73	900.69	900.69
13	490.45	491.02	500.77	650.88	700.73	900.69	900.69
14	490.45	491.02	500.77	650.31	700.73	900.69	900.69
15	490.45	490.45	500.77	650.88	700.73	900.69	900.69
16	491.02	490.45	500.77	650.88	700.73	900.69	900.69
17	491.02	491.02	500.77	650.88	700.73	900.69	900.69
18	490.45	491.02	500.77	650.31	700.73	900.69	900.69
19	490.45	491.02	500.77	650.31	700.73	900.69	900.69
20	491.02	490.45	500.77	650.88	700.73	900.69	900.69
Mean	490.77	490.80	500.77	650.71	700.73	900.66	900.69
Standard deviation	0.29	0.29	0.00	0.27	0.00	0.13	0.00

Table 5: Measured maximum jerk

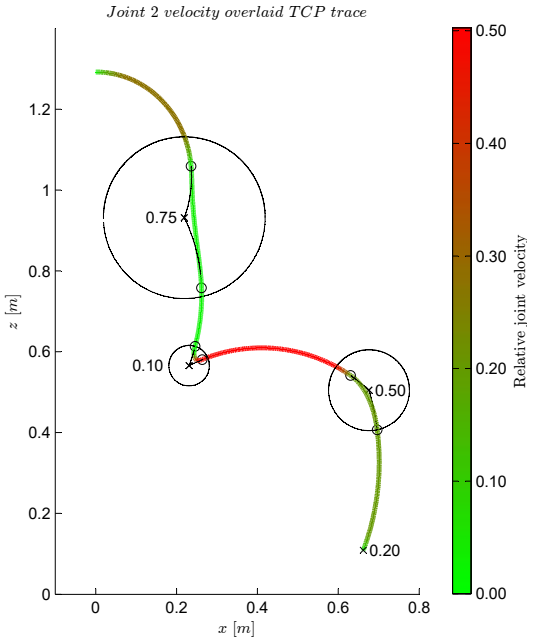
Run	Max Jerk [Deg/s^3]						
	$\dddot{\theta}_1$	$\dddot{\theta}_2$	$\dddot{\theta}_3$	$\dddot{\theta}_4$	$\dddot{\theta}_5$	$\dddot{\theta}_6$	$\dddot{\theta}_7$
1	10026.76	7276.56	9854.87	8880.85	19537.86	32314.82	33174.26
2	9969.47	8307.89	9740.28	10141.35	19251.38	31569.97	32887.78
3	10714.31	9453.80	10657.01	9854.87	19480.57	32945.07	33231.55
4	10313.24	7505.75	10828.90	9740.28	19423.27	32658.59	33403.44
5	11000.79	7849.52	9625.69	10313.24	19365.97	33002.37	33460.74
6	9912.17	6531.72	10255.94	8708.96	19652.45	31569.97	33403.44
7	10599.72	8422.48	9969.47	9224.62	19365.97	32601.30	32658.59
8	10255.94	8250.59	9511.10	9224.62	19537.86	31569.97	33288.85
9	11287.27	8078.70	9167.32	8995.44	19365.97	32429.41	32486.71
10	10886.20	6589.01	10313.24	9052.73	19537.86	32486.71	33002.37
11	9396.51	8594.37	10313.24	10313.24	19365.97	32028.34	33231.55
12	10599.72	8307.89	9625.69	9682.99	19423.27	32314.82	33460.74
13	9396.51	6818.20	10141.35	11000.79	19480.57	31627.27	32658.59
14	9969.47	7505.75	9682.99	9912.17	19251.38	31971.04	33231.55
15	11344.56	7677.63	9396.51	9797.58	19480.57	32372.12	32601.30
16	9339.21	7333.86	9969.47	9740.28	19194.09	32314.82	33403.44
17	11974.82	6703.61	10084.06	8995.44	19251.38	32200.23	33231.55
18	9854.87	7620.34	10886.20	11000.79	19194.09	32601.30	32486.71
19	10198.65	8365.18	9625.69	10313.24	19423.27	31856.45	32658.59
20	10657.01	7964.11	10084.06	9110.03	19537.86	32773.19	32601.30
Mean	10384.86	7757.85	9986.65	9700.18	19406.08	32260.39	33028.15
Standard deviation	697.75	754.50	464.77	674.89	128.92	453.62	357.80

Appendix B Zone velocity experiment

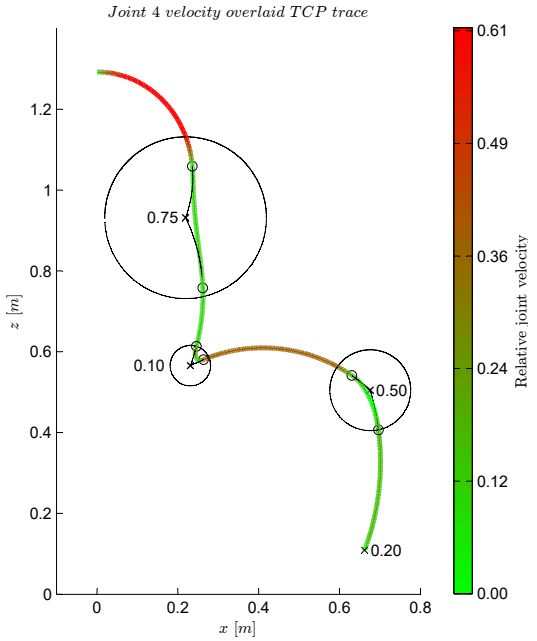
When programming the robot, two consecutive instructions can have different assigned velocities when blended together. An experiment was performed on the real iiwa robot to determine how the velocity varies throughout the blend. A program with varying velocities and blend sizes moving joint 2, 4 & 6 was tested resulting in a TCP trace in the XZ-plane.

- Initial position(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
- PTP(0.0, -0.337023, 0.0, -0.984366, 0.0, 1.685115, 0.0), $b_r = 200.0$ mm, $v_{\%} = 0.75$
- PTP(0.0, -0.474031, 0.0, -1.991944, 0.0, 1.531526, 0.0), $b_r = 50.0$ mm, $v_{\%} = 0.1$
- PTP(0.0, 0.739496, 0.0, -0.778068, 0.0, 1.578825, 0.0), $b_r = 100.0$ mm, $v_{\%} = 0.5$
- PTP(0.0, 1.110378, 0.0, -1.156979, 0.0, 0.895354, 0.0), $v_{\%} = 0.2$

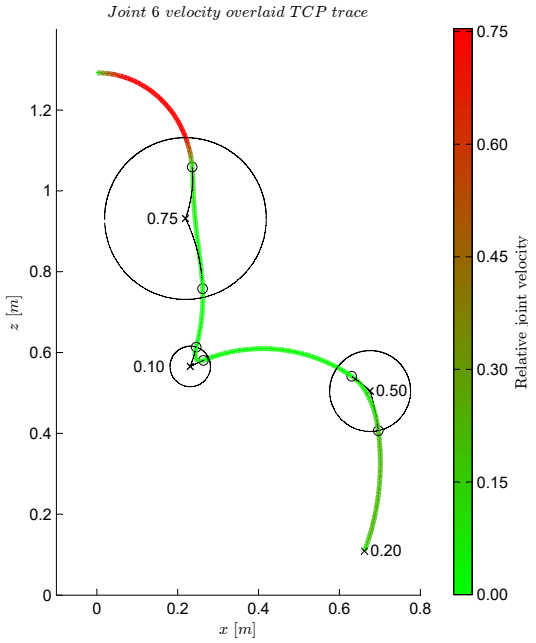
The sampled path in joint space was numerically differentiated to obtain the joint velocities. The velocity for the three moving joint was overlaid the TCP trace individually as can be seen in Fig. 29. It can be seen that the deceleration occurs before entering the blend segment and a constant velocity is kept during the blend segment. The experiment shows that the velocity throughout the blend segment is constrained by the assigned velocity of the following instruction.



(a) Joint 2



(b) Joint 4



(c) Joint 6

Figure 29: Joint velocity overlaid TCP trace. The circles represents where the blend segment starts and stops(c_1 & c_2). The numbers in the graph is the relative velocity assigned to the instructions.

Appendix C Settling time experiment

It has been observed that the trajectory generator of the controller adds a time to let the robot settle when a instruction is programmed without any blend. The added time was experimentally tested on the real robot by instructing the robot to go to 200 random points with random velocity and no blend. The result is given in Table 6. The time stationery time added was observed to be around 0.005 second.

Table 6: Experimentally determined stationary time applied when entering a point without blend.

Mean Stationary Time	Standard Deviation Stationary Time
0.049608 s	0.009818 s