Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybenetics

Time Parameterization of the Manipulator Path

Kryštof Teissing

Supervisor: Ing. Pavel Krsek, Ph.D
Study program: Cybernetics and Robotics
May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Teissing Kryštof**	Personal ID number:	**474544**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Time Parameterization of the Manipulator Path

Bachelor’s thesis title in Czech:

Časová parametrizace dráhy manipulátoru

Guidelines:

1. Study the existing algorithm of path planning and mainly time parameterization of trajectory.
2. Test the existing algorithm implemented in ROS (Robot Operating System) and module MoveIt.
3. Explore the options of parameter settings and configuration of existing algorithms. Try to find the proper setup for robot KUKA iiwa.
4. As necessary, modify the existing implementation or develop and implement your own algorithm for time parameterization.
5. Write proper documentation.

Bibliography / sources:

[1] Haruhiko Asada, Jean-Jacques Slotine: Robot Analysis and Control. John Wiley and Son, New York, USA 1986, ISBN: 978-0471830290.
[2] Reza N. Jazar: Theory of Applied Robotics: Kinematics, Dynamics, and Control. Springer, 2010, ISBN: 978-1441917492.
[3] Hung Pham, Quang-Cuong Pham: A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis. June 2018, IEEE Transactions on Robotics 34(3): 645–659.
[4] B. Paden, K. Sullivan: Bounded deviation trajectory interpolation for robot manipulators. In Proceedings. 1988 IEEE International Conference on Robotics and Automation: 56–61, IEEE 1988, ISBN:0-8186-0852-8.

Name and workplace of bachelor’s thesis supervisor:

Ing. Pavel Krsek, Ph.D., Robotic Perception, CIIRC

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **05.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Pavel Krsek, Ph.D. Supervisor’s signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.
Date of assignment receipt Student’s signature

Acknowledgements

I would like to thank all the people who have helped me in making this thesis, especially my supervisor Ing. Pavel Krsek, Ph.D for valuable advice throughout the whole project, Ing. Vladimír Smutný, Ph.D and the rest of the CIIRC RMP team for consultations and technical support. I also owe thanks to my mother Mgr. Alžběta Soperová for proofreading this thesis and the rest of my family for their support along the way.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

Abstract

This thesis deals with the motion planning of an industrial KUKA LBR iiwa 7 collaborative manipulator implemented in the Robot Operating System and its package MoveIt, and in particular time parameterization. The time parameterization algorithms already implemented in MoveIt, namely 'Iterative Parabolic Time Parameterization' and 'Time Optimal Trajectory Generation', were compared with the recently introduced algorithm 'Time-Optimal Path Parameterization based on Reachability Analysis'. The aim was to improve the initial robot setup, which was suffering from jittery movement at higher speeds. Both the path planner and time parameterization settings have been tested and compared on a set of predefined paths. In order to subject the trajectory generation to the correct kinodynamic bounds, joint acceleration limits of the KUKA robot were estimated. Based on the experiments, we have made conclusions concerning the suitability of the individual algorithms for our setup.

Keywords: Motion planning, Time optimal path parameterization, Trajectory, MoveIt, ROS, IPTP, TOTG, TOPP-RA, KUKA LBR iiwa 7

Supervisor: Ing. Pavel Krsek, Ph.D
CIIRC, CTU,
Jugoslávských partyzánů 3,
160 00 Prague 6,
Czech Republic

Abstrakt

Tato práce se zabývá plánováním pohybu a především časovou parametrizací dráhy průmyslového kolaborativního manipulátoru KUKA LBR iiwa 7. Plánování pohybu je prováděno pomocí knihovny Robotic Operating System a v ní obsaženého softwaru MoveIt. Algoritmy „Iterative Parabolic Time Parameterization“ a „Time Optimal Trajectory Generation“, které jsou již součástí MoveIt softwaru, byly porovnány s nedávno představeným algoritmem „Time-Optimal Path Parameterization based on Reachability Analysis“. Cílem bylo zlepšit pohyb robota, který byl v původním nastavení při vyšších rychlostech trhaný. Jak algoritmus pro plánování dráhy, tak algoritmy generující časovou parametrizaci, byly otestovány na skupině předem určených drah. Byly rovněž odhadnuty limity kloubových zrychlení robota KUKA, aby generovaná trajektorie respektovala jeho kinetická a dynamická omezení. Na základě provedených experimentů byly učiněny závěry o použitelnosti jednotlivých algoritmů pro naše účely.

Klíčová slova: Plánování pohybu, Časově optimální parametrizace dráhy, Trajektorie, MoveIt, ROS, IPTP, TOTG, TOPP-RA, KUKA LBR iiwa 7

Překlad názvu: Časová parametrizace dráhy manipulátoru

Contents

Figures

Tables

Chapter 1

Introduction

A wide application of robotic manipulators across all industrial sectors has been a reality for quite a while now. The need for a strong, reliable and precise multipurpose machine capable of working continuously for several years on a predefined set of tasks drives the widespread automation in manufacturing. The automotive industry serves as a perfect example.

Every single manipulator executes a movement that has to be planned with respect to the manipulator's capabilities and its environment in order to avoid collisions with other objects and humans. Irrespective of whether the movement is planned on the go or beforehand, a similar process is required. A desired action or goal must translate into a set of commands for the manipulator to execute.

In this thesis, we will inspect the path planning process of a robotic cell which includes an industrial manipulator KUKA LBR iiwa7 r800 and runs an open-source framework called Robotic Operating Software for its control and planning. The first part of this thesis deals with the theory of path planning, and in particular time parameterization algorithms. Based on this knowledge, we will try to find the cause of a jittery movement of our manipulator at high speeds and propose a suitable solution for our setup. The focus will be on testing and comparing different path time parameterization algorithms and their settings on a set of paths, as this turns out to be one of the more challenging aspects of the path planning process. Further, we will try to identify the acceleration limits of the KUKA manipulator so that the algorithms employed to generate the trajectory take account of all existing constraints.

Chapter 2

Motion Planning in Robotics

2.1 Path and trajectory

Before exploring the motion planning of the serial manipulator KUKA LBR iiwa7 in more detail, we will define some fundamental terms and theory behind the task of motion planning.

Every motion is defined by a start and end point. In the field of robotics, the start and end points are usually defined by the *poses* or *states* $\mathbf{q} \in C$ of the manipulator from its' *configuration space* C which contains all possible configurations the manipulator can get in to. The number of independent parameters necessary to define the robot's configuration is called *degrees of freedom* (DOF). In the case of a serial manipulator with n rotation joints and thus having n-DOF, a state or pose is usually represented by an n-dimensional vector $\mathbf{q} \in \mathbb{R}^n$ of joint angle values. A configuration space of the manipulator represented by its joint values is called a *joint space* [1].

A spatial construct that connects the initial pose $\mathbf{q_{start}} \in C$ with the goal pose $\mathbf{q_{goal}} \in C$ is called a *path* P [2] and gives a pure geometric description of the motion. It can be described by a continuous function

$$\begin{aligned} P\colon [0,1] &\longrightarrow C \\ s &\longmapsto \mathbf{q}(s) \end{aligned} \tag{2.1}$$

that returns the pose of the manipulator at every position $s \in [0,1]$ along the path, where s is an arbitrary parameter; in our case we consider the path length scaled to 1, and all positions along the path can thus be described by a real number from the interval $[0, 1]$.

A *trajectory* Π is a path with added timing [2]. It can be represented by a continuous function of time

$$\begin{aligned} \Pi \colon [0, t_{end}] &\longrightarrow C \\ t &\longmapsto \mathbf{q}(s(t)) \end{aligned} \tag{2.2}$$

which returns the manipulator's pose at a specific time $t \in [0, t_{end}]$, where t_{end} denotes the duration of the trajectory and $s(t)$ is now a continuous function

$$\begin{aligned} s \colon [0, t_{end}] &\longrightarrow [0, 1] \\ t &\longmapsto s(t) \end{aligned} \tag{2.3}$$

that assigns every time stamp a certain position along the path. It is called a *time parameterization* of path P.

2.2 The motion planning pipeline

The motion planning of a robotic manipulator is usually decoupled in-to two steps. In the first step, a path describing the motion is generated through the path planning and, in the second step, the path is transformed into a trajectory using the path time parameterization algorithms.

2.2.1 Joint space and Cartesian space path planning

Before initiating the path planning, the desired start and goal points of the path have to be specified. This can be done by specifying the start $\mathbf{q}(0)$ and end $\mathbf{q}(1)$ poses of the manipulator from its joint space as stated in chapter 2.1. However, for practical application, it is convenient to first define the start and goal points in coordinates of the manipulator's operation space. The *operation space* of a manipulator is an Euclidean subspace of all possible positions of a reference point on the manipulator's flange or end effector. Depending on the space in which the path is computed, we distinguish between two methods of path planning, namely *joint space path planning* and *Cartesian path planning* [1].

The position of a rigid body in Euclidean space is determined by its translation and rotation component. Consequently, in order to fully describe the desired positions of the manipulator's flange, additional information about its orientation is needed. There are several ways to represent translation and rotation in Euclidean space, one of which is the transformation matrix $\mathbf{T}$.

The manipulator's configuration $\mathbf{q}$, defined by its joint values from a known flange or end effector position, is computed by the *inverse kinematic task* (IKT) [1], formally denoted as

$$\mathbf{T} \rightarrow \mathbf{q}(\mathbf{T}), \tag{2.4}$$

where $\mathbf{T}$ is the transformation matrix of the end effector or flange position. For a reverse transformation of the configuration in the joint variable space into the position of the manipulator's reference link in Cartesian space, the *forward* or *direct kinematic task* (DKT) is used [1].

In addition to the start and goal points, other *via* points can be given to further specify the desired path shape. Subsequently, the first algorithm, called *path planner* (usually referred to only as planner), is called to connect the start, goal and *via* points through a Cartesian or joint space path that respects the manipulator's *kinematics constraints.* The term kinematic constraints means all limitations of the manipulator's configuration, for example joint limits and other obstacles in the manipulator's operation space, as an admissible trajectory should be collision free [3]. The planner uses the IKT and DKT mentioned above to transform the Cartesian path into joint space path $P(s)$ and *vice versa.* However, the resulting path is not always continuous; indeed, it can be only piecewise continuous (e.g., a Cartesian path composed only of straight segments) or even represented by a mere list of configurations of the manipulator's poses sampled along the planned path. This necessitates additional tasks to be performed by the path time parameterization algorithm to compute a continuous joint trajectory.

2.2.2 Path time parameterization

The next step is to compute the trajectory $\Pi(t)$ based on the path $P(s)$ and the given initial condition represented by the start and end joint velocities. A path time parameterization algorithm serves for this purpose. The general principle is to add timing to the planned path in such a way so that both the kinematic and dynamic constraints of the manipulator are met. *Dynamic constraints* are determined by the bounds on joint velocities, accelerations, or torques. These conditions are also referred to as *kinodynamic constraints* [3]. Other criteria, such as the execution time or energy consumption, can be given to find the appropriate trajectory for a certain task. In our applications, the emphasis is placed on the execution time, which should be minimized, and we will thus focus on time optimal path parameterization algorithms. The resulting trajectory $\Pi(t)$ contains the information about the position, velocity and acceleration of each joint at a specific time t.

An aspect of time path parameterization is generating a *smooth* trajectory, i.e. that the manipulator's configuration changes smoothly in time, which usually requires continuous joint velocity, acceleration and sometimes even jerk (a derivative of joint acceleration) courses [2]. Usually, it is also necessary to ensure that the resulting trajectory is executed in a continuous motion, without stopping at each path point. This causes problems in processing piecewise continuous multi-segment Cartesian paths. To create a continuous trajectory, additional blends between Cartesian path segments need to be generated [1]. This results in a feasible trajectory which, however, deviates from the original path. A *feasible trajectory* means a trajectory that respects the manipulator's kinodynamic constraints.

Usually, the resulting trajectory $\Pi(t)$ needs to be sampled with a certain frequency to enable the control of the manipulator. For position-controlled manipulators, the resulting positions represented by joint values $\mathbf{q}(t_i)$ are forwarded at the relevant time $t_i \in [0, T]$ to the manipulator's controller and executed; T is the trajectory duration and $i \in [0, N]$ denotes the i-th time sample from N samples.

Chapter 3

Path Planning and Path Time Parameterization in MoveIt

3.1 Robot Operating System and MoveIt

If we need to program a robot manipulator, we can currently choose from a wide range of frameworks for robot software development that provide a collection of tools, libraries, and conventions simplifying the task at hand. One of the widely used frameworks is the *Robot Operating System* (ROS) which includes a *MoveIt* motion planner framework. We use both these open-source platforms to control the KUKA LBR iiwa 7 r800 manipulator. We will perform a set of experiments aimed to verify the functionality of the motion planning used in our setup and, if possible, remedy their faults.

ROS, namely the ROS 1 version, consists of a set of software frameworks providing services for robot control, including hardware abstraction, low-levIt implements a Computation Graph architecture for process management, where a peer-to-peer network of processes communicate using several different communication methods, including synchronous communication over *Services* and asynchronous streaming process data over *Topics* [4]. Bothmethods implement a publisher-subscriber model of message passing between computation processes, called *Nodes*. As a part of the Computation Graph architecture, the *ROS Master* provides name registration and lookup of the remaining processes and enables a peer-to-peer communication between them [5]. ROS 1 is not a real-time framework, but it can be interconnected with a real-time code [4]. The ROS software includes the MoveIt package, which serves for motion planning, robot kinematics, control and collision checking.

In our setup, we use the software distributions *ROS Melodic* and *MoveIt 1*, which was initially called "MoveIt!" or simply "MoveIt", but, after release of new versionsit is also labeled as MoveIt 1. In this thesis, we will refer to the first version only as MoveIt.

3.2 Implementation of path planning

The MoveIt framework is compatible with several types of path planning algorithms, or planners, as specified in their online documentation [6] The open-source library called *Open Motion Planning Library* (OMPL) , described in detail in [7], is the primary one, which is set as the default planner and will be used in all our experiments.

The OMPL implements the basics of sampling-based motion planning and provides several different state-of-the-art planners, listed in [8]. Sampling-based methods search for a feasible collision-free path in the robot's *free configuration space*, which is sampled as searching in a continuous configuration space would be excessively time consuming. A free configuration space is a subspace of the manipulator's configuration space where each configuration has to be collision-free [8]. There are several methods for connecting the start and goal configuration through sampled configurations into an admissible path, for example a tree-based planning. These are implemented in the available state-of-the-art OMPL planners.

The OMPL is not equipped with a collision checking program; MoveIt handles this task itself. There is a variety of collision detectors available in MoveIt, where the Flexible Collision Library [9] is used by default. MoveIt further handles the manipulator's inverse kinematics (IKT) using a MoveIt LMA (Levenberg-Marquardt) kinematics plugin, a numerical inverse kinematics solver provided by the Kinematics and Dynamic Library [10].

The OMPL planning adapter in MoveIt offers several configuration possibilities for adjusting the OMPL to meet the user's needs. Moreover, additional ROS packages were written by the CIIRC RMP team under the `Capek` project, which enable the user to set additional parameters for the path planning, namely the maximal Cartesian distance and maximal joint angle difference between two points along the generated path. The final path is then returned as a list of sampled joint path poses.

3.3 Time parameterization algorithms

Following the path planning, the second step of the motion planning in the MoveIt framework consists in generating a joint trajectory using a path time parameterization algorithm. There are several algorithms already included in MoveIt, which can be found on their website [11] and we will introduce in the following sections. We will nonetheless exclude the Iterative Spline Algorithm as there have been several reports of issues with its implementation [12]. As the planned path is returned as a list of robot's configurations defined by it's joint positions, the path parameterization is computed in the manipulator's joint space.

3.3.1 Iterative Parabolic Time Parameterization

The *Iterative Parabolic Time Parameterization* (IPTP)is the default option, which we also used in the initial setup of our KUKA LBR iiwa 7 r800 cell. Since no relevant article about this algorithm could be found, we made the following assumptions concerning the algorithms concept, based on the source code included in MoveIt and the theory of trajectory generation as described in the book [13] and article [14].

The main idea of this algorithm is to compute the path time parameterization in the manipulator's joint space, where an initial estimation of the execution time dt_i of all path segments between two adjacent path points is calculated for all the input path points; $i = 0, 1, .., n-2$ is the index of the path segment and n is the total number of path points. The calculation is based on an assumption that the movement of a manipulator with k DOF will be executed with the maximal possible velocity $v_{j_{max}}$ for each joint $j = 0, 1, ..., k-1$, and uses the following equation:

$$dt_{i,j_{min}} = \frac{q_{i+1,j} - q_{i,j}}{v_{j_{max}}},\ i = 0, 1, .., n-2,\ j = 0, 1, ..., k, \tag{3.1}$$

where $q_{i,j}$ is the j-th joint's start position of the i-th path segment.

For each path point, a time difference between the given point and the next point is selected with respect to the slowest joint, and the final estimated execution time of the i-th path segment for all joints thus equals

$$dt_{i_{min}} = \max(dt_{i,j_{min}}). \tag{3.2}$$

At this stage, all joint acceleration limits are disregarded.

In the second step, constant joint acceleration for each k joint in the relevant i-th path segment is computed as follows. The algorithm first considers constant joint velocity

$$v_{i,j} = \frac{q_{i+1,j} - q_{i,j}}{dt_i} \tag{3.3}$$

at each i-th path segment. Next, a constant acceleration $a_{i,j}$ is calculated from the joint velocity $v_{i-1,j}$ of the previous path segment with estimated execution time dt_{i-1} and the joint velocity $v_{i,j}$ of the current one with execution time dt_i. For this purpose, the algorithm considers a time interval $\frac{1}{2}(dt_{i-1} + dt_i)$ where the relevant joint changes its velocity with the acceleration

$$a_{i,j} = 2\frac{v_{i,j} - v_{i-1,j}}{dt_{i-1} + dt_i}. \tag{3.4}$$

as shown in figure 3.1.

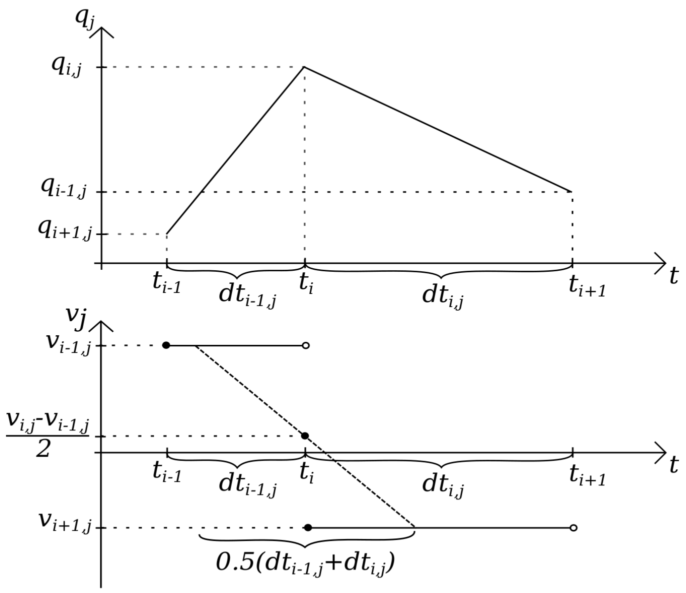


Figure 3.1: A forward/backward pass step of teh IPTP trajectory computation algorithm, where the resulting velocity and acceleration of trajectory points are computed based on the positions $q_{i,j}$ of j-th joint at i-th path point through adjusting the time interval $dt_{i,j}$ between adjacent path points and velocities $v_{i,j}$.

If these joint accelerations exceed the acceleration limits of the manipulator's joints at i-th path segment, then, in a forward pass, starting form the first path segment, the time differences dt_i are one after the other increased repeatedly by a small fraction until the resulting acceleration $a_{i,j}$ satisfies the limits. However, the time differences are not instantly changed to the computed value because the scope of change is limited by an input parameter which determines the maximum change for dt_i within a single iteration. For subsequent iterations, the value of $v_{i,j}$ is changed to

$$v_{i,j} = \frac{1}{2}(v_{i-1,j} - v_{i,j}) \tag{3.5}$$

which does not necessarily lead to a feasible trajectory. The reasons are twofold: firstly, dt_{i-1} can be much smaller than dt_i and $v_{i-1,j}$ thus cannot change to the newly computed $v_{i,j}$ while accelerating with constant acceleration $a_{i,j}$;secondly, the maximal change of the time interval, as determined by the input parameter, can result in the acceleration exceeding the acceleration bounds. To overcome both these factors, also the time difference dt_{i-1} is repeatedly increased in the backward pass i.e. starting from the end segment and handling the segments in opposite direction as in the forward pass, until all accelerations are within bounds. Moreover, once the forward and backward passes are repeated a sufficient number of times, the limitation of time change per iteration will no longer affect the feasibility of the final trajectory. The algorithm will iterate i.e. do forward and backward passes until the resulting trajectory is no longer updated or he maximal number of iterations is reached, which is also given as an input parameter.

In computing joint accelerations, only linear course of acceleration is considered. There are comments in the source code about solving quadratic equations to get the exact time interval and thus implement the parabolic time parameterization, but as it stands now, it does not represent polynomial time parameterization as described in [13]. This is probably also the reason why the word "parabolic" does not appear in the name of the source file.

The output of this algorithm consists in a list of the initial path points with added velocities and acceleration, but there is no option for resampling with a specific time period. This demands a trajectory interpolation by the robot controller, which can lead to trajectories, that deviate from the computed one or even violate the kinodynamic bounds of the manipulator.

3.3.2 Time Optimal Trajectory Generation

The *Time Optimal Trajectory Generation* (TOTG) algorithm is another parameterization algorithm available in MoveIt, which was introduced in the article [15]. This algorithm uses a numerical integration approach and should return a time optimal trajectory respecting all the manipulator's constrains, based on the initial condition specified in section 2.2.2.

The TOTG takes into account the possibility of a discontinuous path, consisting of path points connected by straight line segments. This is a quite common form of describing a planned path and it is also the case with the OMPL planner described in section 3.2. In order to obtain a differentiable path, a pre-processing takes place, where instantaneous changes of direction between two straight segments in the robot's configuration space are replaced with circular segments that start and end tangentially to the original path, as shown in figure 3.2. Without this step, the manipulator would have to stop at each waypoint, which would result in a slow, jerky movement. The method of constructing the circular blends around path points is described in detail in the article [15]. However, one parameter is important for the correct application of this algorithm, specifically the *maximal distance* δ of the original path point, i.e configuration, from the circular blend with which it is replaced. The replacement causes deviation from the originally collision-free path computed by the planner, so some additional checking might be necessary in high precision applications. The maximal distance δ is determined by an input parameter of the implemented TOTG algorithm.

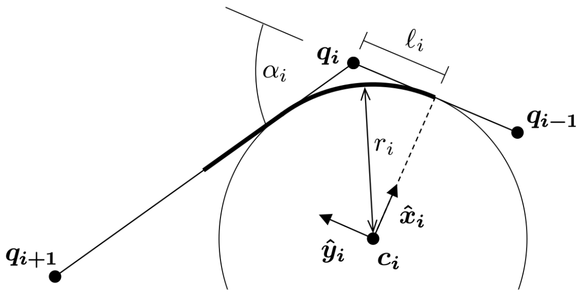


Figure 3.2: Circular blend around a waypoint as part of path preprocessing in the TOTG algoritm [15].

Through the pre-processing, we obtain a path $\mathbf{f}(s)$ which returns a pose $\mathbf{q}(s)$ for every position $s \in [0, s_f]$ along the path of the length s_f (the path $\mathbf{f}(s)$ corresponds to our definition of path 2.1; however, we used a scaled interval $s \in [0, 1]$ for the parameter s), formally expressed as

$$\mathbf{q} = \mathbf{f}(s). \tag{3.6}$$

As the path $\mathbf{f}(s)$ is continuous and differentiable, joint velocities $\dot{\mathbf{q}}(s, \dot{s})$ and accelerations $\ddot{\mathbf{q}}(s, \dot{s}, \ddot{s})$ can be derived as continuous functions of position s, velocity $\dot{s}$ and acceleration $\ddot{s}$ along the path,

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{f}'(s)\dot{s} \\ \ddot{\mathbf{q}} &= \mathbf{f}'(s)\ddot{s} + \mathbf{f}(s)\dot{s}^2 \end{aligned} \tag{3.7}$$

where $\square'$ is a derivative with respect to s and $\dot{\square}$ is a time derivative.

To reduce the problem of trajectory generation to one dimension, joint acceleration and velocity constraints $\dot{q}_j^{max}$ and $\ddot{q}_j^{max}$ of j-th joint are transformed into constraints on velocity $\dot{s}$ along the path using the equations 3.7.

These are divided in to path velocity constraints $\dot{s}^{max}_{acc}(s)$ determined by joint acceleration limits, on one hand, and path velocity constraints $\dot{s}^{max}_{vel}(s)$ determined by joint velocity limits, on the other hand, which are both functions of position s along the path.

Further, in order to obtain the time optimal trajectory, a principle is applied where path velocity $\dot{s}$ is maximized. This implies that at every step of the trajectory, the manipulator should either accelerate with the maximal/minimal acceleration possible ($\pm\ddot{s}_{max}$) or lie on a limit curve defined as the minimum of $\dot{s}^{max}_{acc}(s)$ and $\dot{s}^{max}_{vel}(s)$, as shown in figure 3.3. The points where the acceleration changes are called switching points and are highlighted with an arrow in figure 3.3.

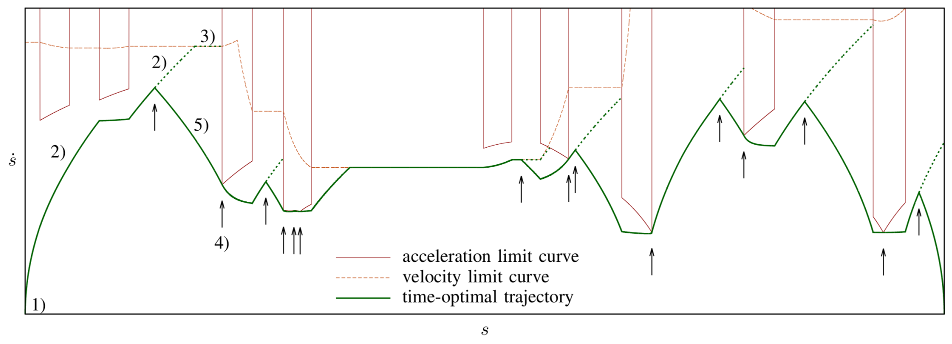


Figure 3.3: Integration of path velocity $\dot{s}$ with respect to constraints dictated by joint acceleration $\dot{s}^{max}_{acc}(s)$ and joint velocity $\dot{s}^{max}_{vel}(s)$ [15].

The final time optimal trajectory is then computed by numerically integrating forwards and backwards along the path, searching for path acceleration and velocity limit curves and finding the appropriate switching points. A final backward integration is performed to verify that the resulting trajectory is admissible. An admissible trajectory means a trajectory that respects the kinodynamic constraints of the manipulator.

The implementation of this algorithm in MoveIt [16] also enables to resample the final trajectory with a certain period `dt`, which is then returned as a list of joint positions, accelerations and velocities at according time stamps.

3.4 Execution of trajectory in ROS

An analysis of the manipulator's behavior when executing a movement computed by a trajectory generating algorithm should consider the process of execution of the trajectories. The ROS includes a `ros_controls` package

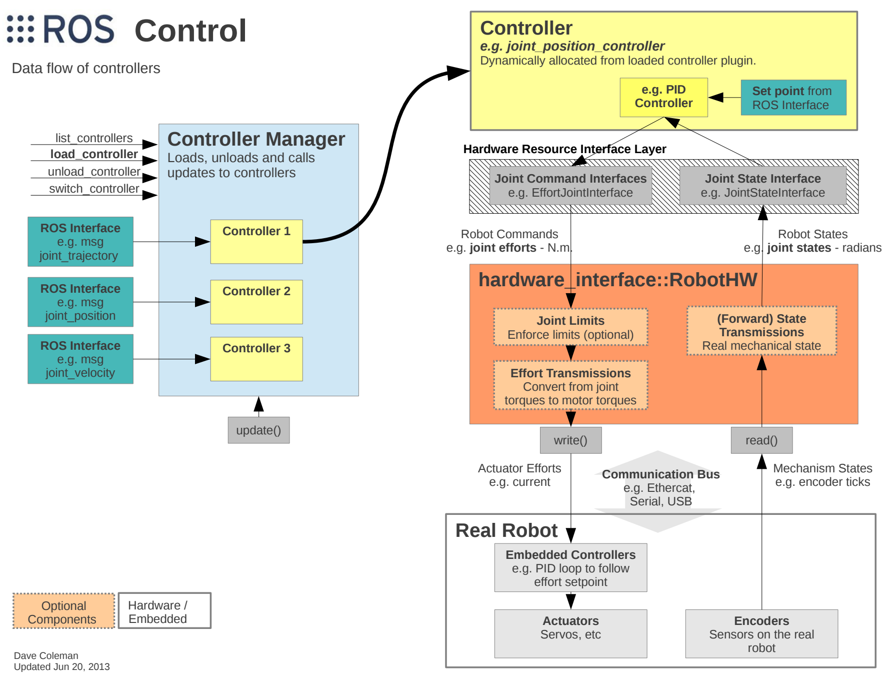

ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m.
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 20, 2013

Figure 3.4: Trajectory execution in ROS [18]

for this task. A controller called `JointTrajectoryController` is included in the `ros_controls` package, which provides an action interface for executing the by MoveIt computed joint trajectories on a group of joints.

This action interface is called `FolowJointTrajectory` and through it, the goals and tolerances for execution are specified. It also provides a feedback and possibility to cancel a request mid execution. A spline interpolator is implemented as a part of the `JointTrajectoryController` by default, where the input trajectory with defined joint velocities and accelerations is interpolated with a quintic function [17].

The `JointTrajectoryController` is managed by the `Controller Manager`. For position-controlled manipulators, which is also the case of our setup for KUKA LBR KUKA LBR iiwa 7 r800, the desired positions are then forwarded to the `JointStateController` and sent, via a `hardware_interface`, to the manipulator, which uses its own embedded controllers to steer itself to the desired goal pose. The position control of the `JointStateController` in our setup runs with the frequency of 1 kHz. The whole principle is illustrated in figure 3.4.

Chapter 4

Alternative Path Time Parameterization Algorithms

4.1 Alternative algorithms

We have encountered certain issues with the default path time parameterization algorithm IPTP included in MoveIt framework and described in section 3.3.1. As time parameterization plays a key role in the decoupling of the trajectory generation into path planning and path time parameterization, many algorithms and approaches have been developed to this end.

There are two main methods used for computing time optimal path parameterization. The first one is the Numerical Integration (NI), which was implemented *inter alia* in the TOTG algorithm described in chapter 3.3.2 and introduced in 2012. A more recent implementation is the *Time-Optimal Path Parameterization* (TOPP) algorithm introduced in the article[19] in 2014. The second method is a Convex Optimization (CO), most recently used in the algorithm called *Time Optimal Path Parameterization based on Reachability Analysis* (TOPP-RA) introduced in 2018 in the article [20], which claims promising results. It should succeed TOPP and other NI based algorithms and promises 100% success rate of finding a feasible trajectory if it is possible. Indeed, the MoveIt developers have suggested implementing TOPP-RA in MoveIt [21], and we will therefore test this algorithm in comparison to the ones already included in MoveIt.

4.2 TOPP-RA

The main idea of *Time-Optimal Path Parameterization based on Reachability Analysis* (TOPP-RA) algorithm is to "recursively compute reachable and controllable sets at discretized positions on the path by solving small Linear Programs"[20] and then with a greedy approach select the quickest trajectory possible from the controllable sets.

The algorithm considers generalized first- and second-order constraints of a manipulator

$$\begin{aligned} \mathbf{A}^v(\mathbf{q})\dot{\mathbf{q}} + \mathbf{f}^v(\mathbf{q}) \in \mathcal{C}^v(\mathbf{q}) \\ \mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{f}(\mathbf{q}) \in \mathcal{C}(\mathbf{q}) \end{aligned} \tag{4.1}$$

where $\mathbf{A}$, $\mathbf{A}^v$, $\mathbf{B}$, $\mathbf{f}$, $\mathbf{f}^v$ are transformations from $\mathbb{R}^n$ to $\mathbb{R}^{m\times n}$, $\mathbb{R}^{m\times n}$, $\mathbb{R}^{n\times m\times n}$, $\mathbb{R}^m$ and $\mathbb{R}^m$ respectively, $\mathcal{C}$ is a convex polytope and $\mathcal{C}^v$ is a convex set. General first-order constraints include, in particular, direct velocity bounds and momentum bounds; general second-order constraints include for example joint torque bounds or acceleration and velocity bounds. The meaning of the transformations in the equation 4.1 differs for different kinodynamic constraints e.g. for torque constraints on a fully actuated manipulator

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau \tag{4.2}$$

can be transformed in to 4.1 with $\mathbf{A} := \mathbf{M}$ as the manipulator inertia matrix, $\mathbf{B} := \mathbf{C}$ as the Coriolis tensor, $\mathbf{f} := \mathbf{g}$ as the vector of gravity forces and $\mathcal{C}(\mathbf{q}) := [\tau_1^{min}, \tau_1^{max}] \times ... \times [\tau_n^{min}, \tau_n^{max}]$ as a convex polytope generated by the torque τ_i constraints of each joint $i = 1, ..., n$ [20][22].

The constraints 4.1 are then transformed in to constraints on path position s, velocity $\dot{s}$ and acceleration $\ddot{s}$ by application of the same equations 3.7 as in TOTG, and projected on a $\dot{s}s$-plane. The path P, as defined in 2.1, is then divided in to N discrete segments $s_0, s_1, ..., s_N$ linked by $N + 1$ grid points. The number of grid points can be set as an input parameter when using the implementation of the TOPP-RA algorithm [23]. At each path segment s_i, a *state* (squared path velocity) $x_i = \dot{s}_i^2$ and a *control* (constant path acceleration) u_i are defined.

The Reachability Analysis, newly introduced to the optimal time parameterization, uses two main constructs, specifically the reachable and the controllable sets. A *reachable set* $\mathcal{L}_i(\mathbb{I}_0)$ is a set of all states x from admissible states $\mathcal{X}_i$ (states the manipulator can get in considering feasible controls) at i-stage that can be reached by choosing a sequence of admissible controls $u_0, ..., u_{i-1}$ from the initial state x_0 out of a set of starting states $\mathbb{I}_0$. A *controllable set* $\mathcal{K}_i(\mathbb{I}_N)$, on the other hand, is a set of all states $x \in \mathcal{X}_i$ from which the manipulator can be steered in-to a set of ending states $\mathbb{I}_N$ using a sequence of admissible controls $u_i, ..., u_{N-1}$. The Convex Optimization takes place at this stage as a set of Linear Programs needs to be solved to find the upper and lower bounds of reachable and controllable sets at each path step. A maximization/minimization of states x takes place subject to satisfying the projected general second-order constraints 4.1.

Firstly, in a backward pass, starting from the end of the path and reflecting the defined initial and final joint velocities, controllable sets are recursively computed for each grid point. Secondly, in a forward pass, at each grid point, the algorithm greedily selects from a controllable set the highest possible controls (i.e. controls that result in the quickest motion) that still satisfy the given constraints and are therefore admissible. The process is shown in figure 4.1. The principle on which the controls are selected is determined by a parametrizer. There are two parametrizers available in TOPP-RA, the spline parametrizer, which will try to fit a spline on the original waypoints while using the controls from the controllable sets. Next, a parametrizer which realizes a constant acceleration between the waypoints can be selected.

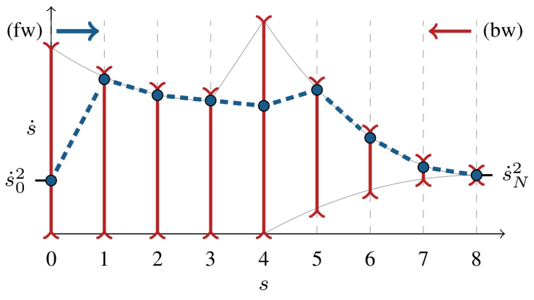


Figure 4.1: Forward and backward passes of the TOPP-RA algorithm with highlighted controllable sets and selected controls minimizing the execution time [20].

The implementation of this algorithm allows to define the second-order constraints as acceleration and velocity limits and returns the final trajectory as a list of trajectory points containing joint positions, velocities and acceleration, sampled with a frequency given as an input parameter. In addition, a specific trajectory duration can be requested.

Chapter 5

Identification of Acceleration Bounds of the KUKA Manipulator

5.1 KUKA LBR iiwa 7 r800 Robot

The industrial robot KUKA LBR iiwa 7 r800 [24] is a redundant serial manipulator with seven rotation joints; each joint is equipped with a torque sensor, in addition to a position sensor. It was designed as a human robot collaborative manipulator for lighter payloads. The robot used for the experiments in this thesis is depicted in picture 5.1 mounted on a table. This setup is a part of a `Capek` project implemented by the Robot and Machine Perception (RMP) team in the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC). Detailed specification and dimensions are contained in the tables 5.1 and 5.2.

Table 5.1: KUKA LBR iiwa 7 r800 parameters [25].

	KUKA LBR iiwa 7 r800
Number of axis [-]	7
Number of controlled axis [-]	7
Pose repeatability [mm]	±0.1
Maximum Reach [mm]	800
Rated payload [kg]	7

Table 5.2: KUKA LBR iiwa 7 r800 joint J_i, $i = 1, .., 7$ range of motion $\Delta\theta_{max}$, velocity υ_{max} and torque τ_{max} limits [26].

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
$\Delta\theta_{max}$ [deg]	±170	±120	±170	±120	±170	±120	±175
υ_{max} [deg/s]	98	98	100	130	140	180	180
τ_{max} [Nm]	176	176	110	110	110	40	40

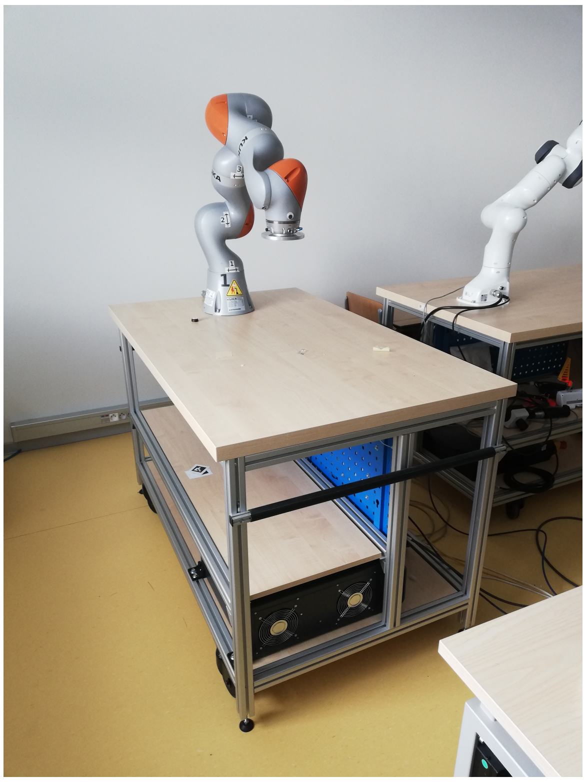


Figure 5.1: KUKA LBR iiwa 7 r800 manipulator.

5.2 Acceleration bounds of the KUKA robot

Having reviewed the initial setup of our robot cell, we found out that the acceleration limits used for specifying the joint limits of the KUKA robot were omitted in the MoveIt configuration file `joint_limits.yaml`, and the time parameterization algorithm set them internally as the default values. Consequently, the IPTP time parameterization, implemented in MoveIt, used the default joint acceleration limits defined by the global variable `DEFAULT_ACCE_MAX = 1.0` rad·s^{-2} for all joints [27].

Moreover, only velocity and torque bounds on the robot's joints were specified in the KUKA LBR iiwa 7 r800 robot documentation as listed in table 5.2. The acceleration limits indeed depend on the manipulator's configuration, as the arm's ratio of extension directly influences the torque applied on the joints. In this thesis, we will disregard pose related acceleration constraints as that would requre an identification of the manipulator's dynamic model which, goes beyond the scope of this thesis. Moreover, the implementations of the IPTP and TOTG algorithms use constant joint acceleration bounds.

Accordingly, we will consider the worst-case scenario i.e. the maximal possible extension of the manipulator's arm and poses, where the gravitation force applied on the extended part of the arm causes the biggest moment on the corresponding joint. We will use the torque sensors in the robot's joints to identify the acceleration bounds, based on the position step response of each joint. We will also take account of the maximal payload of the robot, which is specified in table 5.1.

5.3 Calculating the worst-case scenario acceleration limits

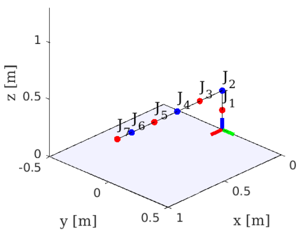
A pose has been chosen for each joint where the static torque acts upon the joint. They are shown in figures 5.2, and the joint values which define them are specified in table 5.3.

Table 5.3: The joint position values θ_i, $i = 1,..,7$ for all poses P_j, $j = 1,..5$ used for joint acceleration limits estimation.

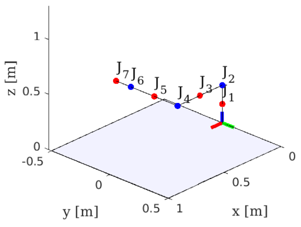
	$\mathbf{P_1}$	$\mathbf{P_2}$	$\mathbf{P_3}$	$\mathbf{P_4}$	$\mathbf{P_5}$
θ_1 [deg]	0	0	0	0	0
θ_2 [deg]	90	90	0	0	0
θ_3 [deg]	0	90	0	0	0
θ_4 [deg]	0	90	-90	-90	0
θ_5 [deg]	0	0	0	90	0
θ_6 [deg]	0	0	0	90	90
θ_7 [deg]	0	0	0	0	0

Table 5.4: The nominal distance from the manipulator's flange to the payload's center of gravity [25].

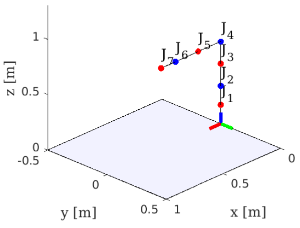
L_{xy} [mm]	35
L_z [mm]	60



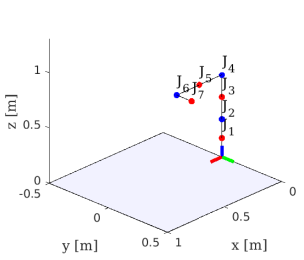
(a) : P_1 for J_1 and J$_2$



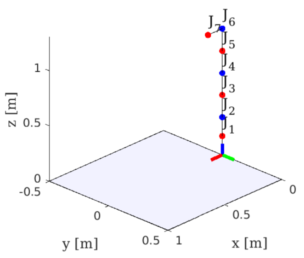
(b) : P_2 for J_3



(c) : P_3 for J_4



(d) : P_4 for J_5



(e) : P_5 for J_5 and J$_6$

Figure 5.2: Poses P_j, $j = 1, .., 5$ for identification of acceleration bounds of joint/-s J_i, $i = 1, ..., 7$ from a step response.

A simplified dynamics of the robot's arm reduced to a homogeneous rod with the mass m_{arm} was used for all poses, as shown in figure 5.3. A gravitational acceleration g causes gravitation force $F_{g_{arm}}$ on the rod's center of mass C_{arm} with the distance r_{arm} from the axis of rotation of the i-th joint. A payload with the mass $m_{payload}$ is fixed on the other end of the rod, having the center of mass $C_{payload}$ in the distance $r_{payload}$ from the axis of rotation. It is fixed in the maximal distance L_z (or L_{xy} for joint J_7) from the robot's arm tip as listed in table 5.4. A gravitational force $F_{g_{payload}}$ acts on the payload.

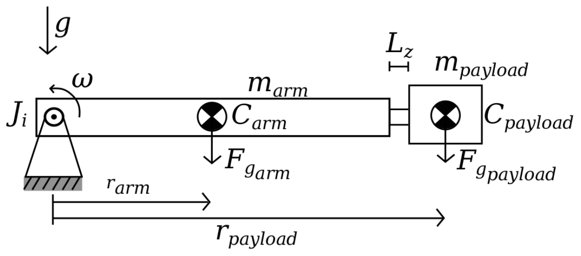


Figure 5.3: Simplified dynamics of robot's arm reduced to a homogeneous rod.

If we consider the arm stationary, static moments $M_{s_{arm}}$ and $M_{s_{payload}}$ are applied on the joint J_i. The static moment of the arm $M_{s_{arm}}$ can be measured with the torque sensor in the corresponding joint and the static moment of the payload $M_{s_{payload}}$ can be calculated from the general definition of moment $\mathbf{M} = \mathbf{F} \times \mathbf{r}$, where $\mathbf{F}$ is a force vector and $\mathbf{r}$ is a position vector. In our case, the gravitational force is perpendicular to the position vector, and thus the norm of the moment can be calculated using the following equation:

$$M_{s_{payload}} = F_{g_{payload}} \cdot r_{payload}. \tag{5.1}$$

In addition, moments caused by the angular acceleration ε_{arm} are applied on the joint if we consider the arm rotating around the axis of the i-th joint. The moment of inertia I_{arm} of the arm rotating around the joint axis can be calculated from the measured torque $M_{d_{arm}}$ and corresponding angular acceleration ε_{arm} of a certain motion. A response on a step change in joint position of the magnitude $\Delta\theta = 2^o$ was used for this purpose. The moment of inertia is then calculated from the known equation $M = I\varepsilon$ as

$$I_{arm} = \frac{M_{d_{arm}}}{\varepsilon_{arm}} \tag{5.2}$$

Next, the moment caused by the payload is calculated from its moment of inertia $I_{payload}$ in relation the corresponding joint's axis of rotation which lies in distance $r_{payload}$ as

$$\begin{aligned} I_{payload} &= m_{payload} \cdot r_{payload}^2 \\ M_{d_{arm}} &= I_{payload}\varepsilon_{arm}. \end{aligned} \tag{5.3}$$

The maximal torque τ_{max} applied on a joint by a rotating arm can then be expressed as

$$\begin{aligned} \tau_{max} = M_{max} =& F_{g_{arm}} r_{arm} + I_{arm}\varepsilon_{max} + F_{g_{payload}} r_{payload} + I_{payload}\varepsilon_{max} \\ =& M_{s_{arm}} + I_{arm}\varepsilon_{max} + M_{spayload} + I_{payload}\varepsilon_{max} \end{aligned} \tag{5.4}$$

The final acceleration bounds for the corresponding joint are subsequently computed from equation 5.4 as

$$\pm\,\varepsilon_{max} = \frac{\tau_{max + M_{s_{arm}} + M_{s_{payload}}}}{I_{arm} + I_{payload}} \tag{5.5}$$

This procedure was used to estimate the worst-case scenario acceleration bounds for all seven joints.

The norm of the position vectors r that correspond to the extended arm section and extended payload were calculated based on the dimensions of the KUKA LBR iiwa 7 r800 links specified in table 5.5. For the calculation, both the measured joint accelerations and torques needed to be smoothed out using Moving Avarage Filter as there were high oscillations.

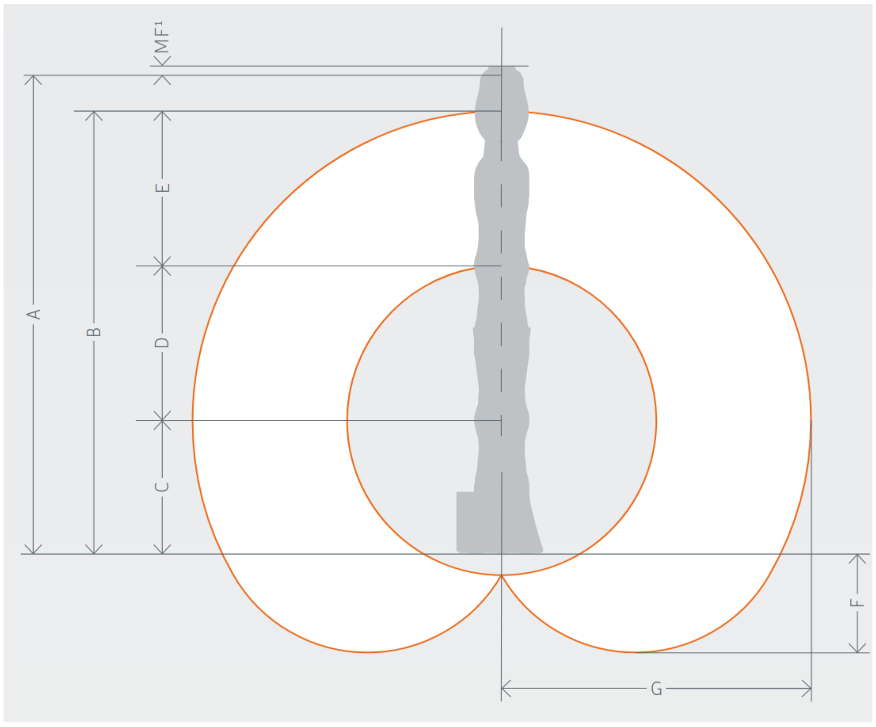


Figure 5.4: Scheme of KUKA LBR iiwa 7 r800 links [26].

Table 5.5: Dimensions of KUKA LBR iiwa 7 r800 links as depicted in figure 5.4 [26].

Dimensions	A	B	C	D	E	MF
Length [mm]	1266	1140	340	400	400	50

5.4 The results and their verification

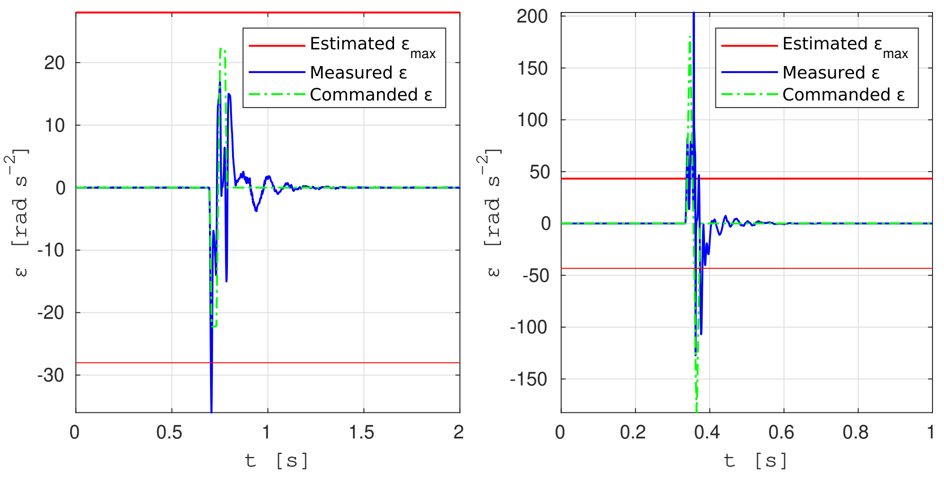
The resulting estimated acceleration bounds are shown in the first column of table 5.6. They were compared with the courses of the measured acceleration and commanded acceleration by the KUKA Sunrise Cabinet robot Controller (KRC), both provided through KUKA FRI to ROS hardware interface, as described in section 7.1. The comparison is shown in figure 5.5 for joints J_1 and J_4, as those have the biggest error; all courses can be seen in the appendix A.

It can be clearly seen that the results are fairly near to the measured courses and except for the joint J_1, all the estimated bounds are smaller than the maximal accelerations commanded by the KRC. However, when testing these values on trajectories consisting of the same rotational motion around the corresponding joint axis, only with a higher position difference

$\Delta\theta = 20^o$, the KUKA robot internal controllers kept stopping the movement, as it probably exceeded their internally set kinodynamic bounds. The same poses as those specified in table 5.3 were used.

Table 5.6: Calculated and corrected i-th joint J_i acceleration bounds, $\varepsilon max_{calculated}$ and $\varepsilon max_{corrected}$ respectively, of the KUKA robot with and without rated payload.

	Without payload		With payload 7 kg
	$\varepsilon max_{calculated}$ [rad$\cdot s^{-2}$]	$\varepsilon max_{corrected}$ [rad$\cdot s^{-2}$]	$\varepsilon max_{calculated}$ [rad$\cdot s^{-2}$]
J_1	± 28.02	± 3.65	± 3.17
J_2	± 16.38	± 3.65	± 0.80
J_3	± 60.79	± 3.85	± 1.71
J_4	± 43.27	± 6.20	± 2.56
J_5	± 133.56	± 7.00	± 5.85
J_6	± 162.29	± 12.30	± 6.58
J_7	± 177.68	± 12.30	± 11.64



(a) : Courses for joint J_1 **(b) :** Courses for joint J_4

Figure 5.5: Comparison of estimated joint acceleration bounds ε_{max}, measured joint acceleration ε and commanded joint acceleration ε by KUKA robot controller for joints J_i, $i = 1, 4$.

The values in the second column of table 5.6 were then determined through systematically lowering the computed values of the maximal accelerations without the rated payload. They are significantly lower than the initially estimated ones, which could be caused by several factors. There might be certain violations in the KUKA inner controller, which remain unknown to us, as the KUKA low-level control remains a black box. Secondly, there could be certain issues with the MoveIt `joint_trajectory_controller` and trajectory execution in general as the position commands are not passed correctly and cause violations in the KUKA internal controller. The trajectory was computed by the TOPP-RA algorithm; however, the possibility that the time parameterization was at fault was excluded as the generated trajectory

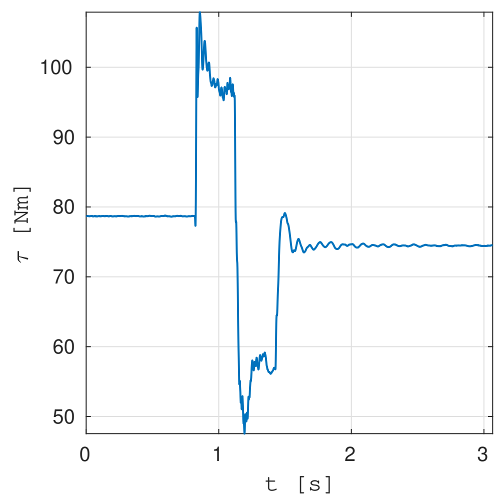


Figure 5.6: Torque course of joint J_2 when executing a joint rotation of $\Delta\theta = 20^o$. The trajectory was generated by the TOPP-RA algorithm.

had correct joint courses. A violation of torque bounds was also excluded as the torque courses of all joints were significantly below the maximal torques allowed while executing the same rotation movement of $\Delta\theta = 20^o$ with the estimated maximal joint acceleration. As an example, a course of the joint J_2 is shown in figure 5.6, which, even at the maximal value around 110 Nm, was way below the maximum torques allowed, i.e. 176 Nm for joint J_2.

Subsequently, the worst-case scenario joint acceleration bounds for a loaded manipulator specified in the third column of table 5.6 were computed from the corrected acceleration bounds ε_{max}, specified in the second column, using equations 5.4 and 5.5. These values were not verified due to time constraints and safety aspects of such experiments.

Chapter 6

Implementation of TOPP-RA as a Service in to ROS

There is already a package available for connecting TOPP-RA to ROS called `topp_ros` [28], which provides a ROS Service for calculating the path time parameterization. However, since the TOPP-RA algorithm was introduced quite recently, it has been further developed and improved. Consequently, `top_ros` is compatible with an obsolete version and does not provide all the features available in the newer versions.

Therefore, a similar ROS package called `toppra_srv` has been written, which provides the additional features. It implements a ROS service, which is described by the description file `GenerateToppraTrajectory.srv` shown below.

```
trajectory_msgs/JointTrajectory     waypoints
float64                             sampling_frequency
uint32                              n_gridpoints
float64                             trajectory_duration
bool                                splineParametrizer
bool                                plot
bool                                debug
---
trajectory_msgs/JointTrajectory     trajectory
float64                             computing_time
bool                                success
```

A ROS service request for TOPP-RA parameterization consists of several parameters. First, a list of path points `waypoints` is given. In addition to the robot's configuration, at least the first path point has to contain the information about velocity and acceleration bounds of the manipulator. Next, the number of grid points `n_gridpoints` for the calculation of the controllable sets has to be defined, as described in section 4.2. If this parameter is set to zero or if it is not specified, the TOPP-RA algorithm determines the number of grid points automatically.

As the TOPP-RA algorithm is also able to calculate a trajectory of a given duration, the parameter `trajectory_duration` realizes this feature. Where it is not specified or set to zero, the time optimal trajectory is computed.

In the newer versions, the option of choosing a different parameterizer was added, allowing to select the highest possible controls, i.e. the joint accelerations, from the controllable sets in various manners. A spline parametrizer and a parametrizer ensuring a constant acceleration between the waypoints can be selected. However, due to the issues with ROS 1 compatibility described in the section 6.1, the final trajectory is computed with the spline parametrizer even if the second option has been chosen. The parameter `splineParametrizer` chooses the spline parametrizer by default or if set to `true`; otherwise the constant acceleration parametrizer should be used. This was implemented for further use, once the compatibility issues are resolved.

Finally, the `sampling_frequency` determines the time sampling frequency of the final trajectory. The `plot` and `debug` parameters are used for plotting and printing the information about the computation process of the TOPP-RA algorithm.

The service returns the computed trajectory as a list of trajectory points with the corresponding positions, velocities and accelerations sampled with the sampling frequency. In addition `computing_time` and `success` values are returned.

6.1 TOPP-RA and ROS 1 compatibility

We have identified a problem with the compatibility of the newest version of TOPP-RA library. TOPP-RA is currently being further developed using the latest Python 3. However, ROS 1 Melodic distribution is only compatible with Python 2.7, which became obsolete in January 2020 and causes conflicts when the more recent TOPP-RA version are used. While the `toppra 0.4.0` package is the most recetn release, we implement the service for the `toppra 0.3.1` version to ensure compatibility with Python 2.7. Certain features, such as the constant velocity parameterizer, thus do not work properly, but they are prepared for the possible upgrade of the system to the newer ROS distributions. As the whole `Capek` project, under which the KUKA LBR iiwa 7 r800 robot cell was designed, uses ROS 1, the replacement path time parameterization must be compatible with it.

Moreover, a TOPP-RA C++ API in currently being developed, which could allow to overcome the issue with the Python version compatibility; however, only the basic algorithm has been implemented to date, so it would not help to resolve the missing features, such as the second parameterizer.

Chapter 7

Experiments

7.1 Experimental KUKA robot cell

All the following tests and examples have been carried out on a KUKA LBR iiwa 7 r800 manipulator mounted in a robotic cell, as shown in figure 5.1. A *KUKA Sunrise Cabinet* robot controller (KRC) is used for the robot control, safety and process control of the manipulator. It is connected via *KUKA Fast Robotic Interface* (FRI) to a real time control unit `CapekPC0` (RTCU) running Arch linux and ROS Melodic. The FRI allows the RTCU to obtain information about the robot from the KUKA Sunrise Cabinet and to perform the external control of the manipulator. As the RTCU also provides a connection between KUKA FRI and ROS, a real-time control loop, implemented with the ROS, is able to send the desired robot positions with a frequency of 1 kHz through the FRI to the KUKA Sunrise Cabinet, which then executes the low-level control of the manipulator's movement.

The RTCU is also connected via Ethernet to a Local Area Network providing access to an external client computer which can in turn communicate with the RTCU through the ROS communication services and command the desired robot movement. For this purpose, a PC with Ubuntu 18.04 operating system that runs ROS Melodic with MoveIt 1 was used. It computes and provides the desired trajectory, which is subsequently forwarded to the RTCU. The whole setup, together with an additional second KUKA manipulator, is shown in figure 7.1.

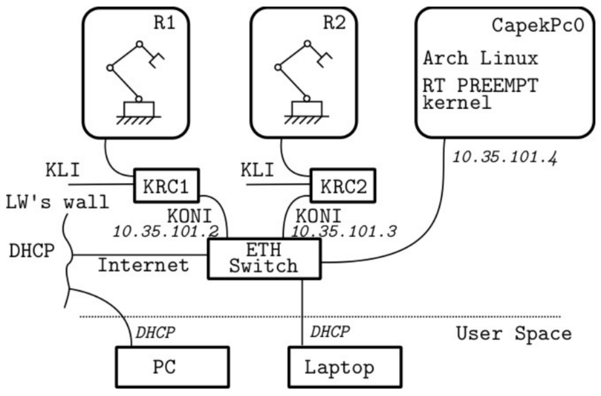


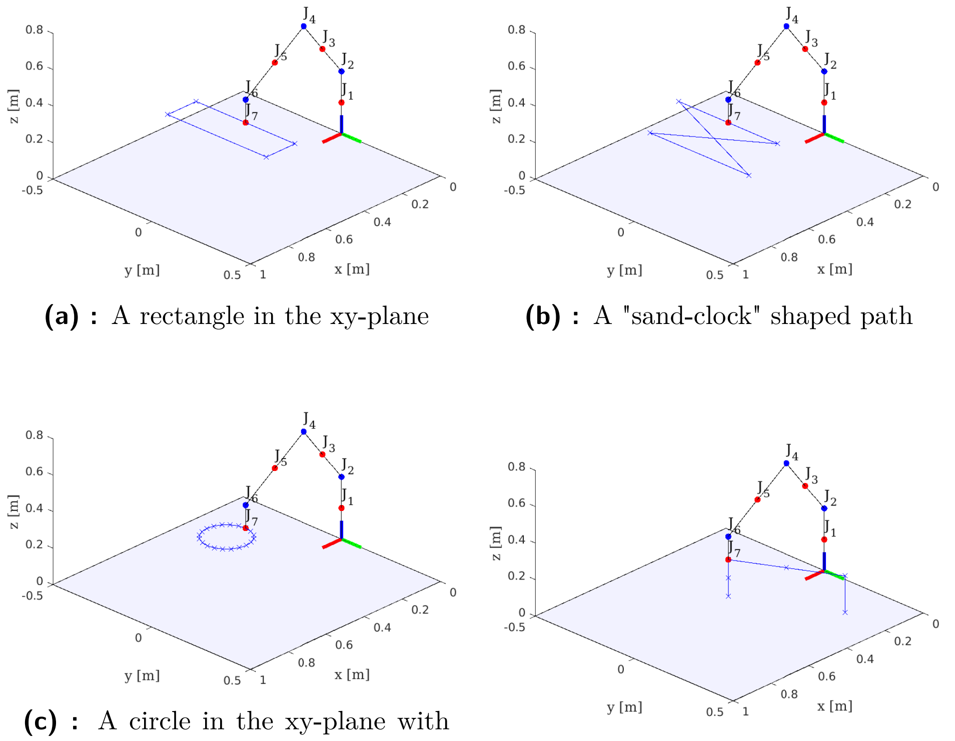
Figure 7.1: A robot cell network including two KUKA LBR iiwa 7 r800 robots, each controlled by the corresponding KUKA Sunrise Cabinet controller (KRC) which is connected via Ethernet switch to an external real time control unit `CapekPc0` and client computers. Only a single robot was used for our experiments. Author [Ing. Vladimir Petrik, Ph.D.].

7.2 Paths for testing

Four trajectories were chosen to test the path time parameterization algorithms.

- A rectangle in the xy-plane shown in figure 7.2a. Basic straight line segments should show how the time parameterization algorithm handle simple moves in the space. Next, the different deviations from the original path should be visible at the edges of the rectangle, as the manipulator would have to change the direction of movement instantaneously if the original path were to be executed.
- A "sand clock" shaped path shown in figure 7.2b is slightly more complex as the manipulator crosses the center part of its operation space several times, which will force the manipulator to slow down because all joints have to move simultaneously.
- A "circle" in the xy-plane path shown in figure 7.2c is the most complex of the four paths as it has quite dense distribution of waypoints and the manipulator moves in the center of its operation space. In reality, the "circle" path is a polygon with 20 edge points since the path is defined by a list of discrete path points.
- "Pick and place" motion path shown in figure 7.2d serves to verify the most common type of trajectories a manipulator of this type is usually tasked to do..

The coordinates of the path points for each path are listed in appendix C, where the world coordinate system is placed in the root of the manipulator.



(a) : A rectangle in the xy-plane

(b) : A "sand-clock" shaped path

(c) : A circle in the xy-plane with the radius 10 cm.

(d) : "Pick and place" motion

Figure 7.2: Paths for testing the time parameterization algorithms with highlighted path points with blue crosses.

7.3 Methods for trajectory error computation

Each path segment in the form of a line connecting two path points is transformed in to a subspace of a one-dimensional affine space $\mathbf{X_i}$ defined by the equation

$$\mathbf{X_i} = \mathbf{p_i} + r \cdot (\mathbf{p_{i+1}} - \mathbf{p_i}) = \mathbf{p_i} + r\mathbf{v},\ i = 1, .., n \tag{7.1}$$

where $\mathbf{p_i} \in \mathbb{R}^3$ and $\mathbf{p_{i+1}} \in \mathbb{R}^3$ are position vectors of the corresponding path points from n path points in total, $r \in \mathbb{R}$ is a scalar and $\mathbf{v} \in \mathbb{R}^3$ is a vector from $\mathbf{p_i}$ to $\mathbf{p_{i+1}}$.

The error of a sampled cartesian trajectory point $\mathbf{z}$ is then computed as its distance from the closest cartesian path segment. This is done by shifting the affine space $\mathbf{X_i}$ representing each path segment to the origin, thus creating a linear subspace $\mathbf{Y_i} = \mathbf{X_i} - \mathbf{p_i}$. This is followed by finding the orthogonal projection $\mathbf{x_{z_s}}$ of a shifted trajectory point position vector $\mathbf{z_s} = \mathbf{z} - \mathbf{p_i}$ on the newly computed linear subspace $\mathbf{Y_i}$ with a normalized basis vector $\mathbf{v_0} = \dfrac{\mathbf{v}}{|\mathbf{v}|}$ with the equation

$$\mathbf{x_{z_s}} = \mathbf{v_0}\mathbf{v_0}^T\mathbf{z_s}. \tag{7.2}$$

The minimal distance of the trajectory point $\mathbf{z}$ from the affine space $\mathbf{X_i}$ equals to the norm of a vector $\mathbf{x_{z_s}} - \mathbf{z_s}$ pointing from the shifted trajectory point $\mathbf{z_s}$to its orthogonal projection $\mathbf{x_{z_s}}$, which is orthogonal to the linear subspace and represents thus the minimal distance.

As the i-th path segment is not represented by the whole affine space $\mathbf{X_i}$ but rather its subspace limited by the edge path points, the coordinates t of the projection $\mathbf{x_{z_s}}$ with respect to the basis vector $\mathbf{v_0}$ are computed with the equation

$$t = \mathbf{v_0}^T\mathbf{z_s}. \tag{7.3}$$

If the projection lies within the path segment, i.e. $t \geq 0$ and $t \leq |\mathbf{v}|$, the minimal distance of the trajectory point $\mathbf{z}$ to the i-th path segment is then computed as

$$d_{z,i} = |\mathbf{x_{z_s}} - \mathbf{z_s}|. \tag{7.4}$$

Else, the minimal distance $d_{z,i}$ is equal to the distance between the trajectory point $\mathbf{z}$ and the closest edge point

$$d_{z,i} = \min\left\{|\mathbf{p_i} - \mathbf{z}|, |\mathbf{p_{i+1}} - \mathbf{z}|\right\}. \tag{7.5}$$

For every trajectory point $\mathbf{z}$, the distances $d_{z,i}$ from all path segments are computed and the final error of the corresponding trajectory point is then computed as

$$\mathrm{err}_{\mathbf{z}} = \min_i \left\{d_{z,i}\right\}. \tag{7.6}$$

7.4 Simulating the fault movement

The main objective of this thesis was to find a solution to remedy the jerky movement of the KUKA manipulator at higher speeds. Initially, both the path planner and the path time parameterization algorithms in MoveIt were set to the default options, i.e. the OMPL planner and IPTP algorithm with parameters mentioned as default in the following section 7.5. We will refer to these options as the "initial setup". Even though the initial setup has been used for some time, no significant issues with the movement have been encountered until a recent task, which required a quick, precise and the smoothest possible movement. This turned out to be unfeasible with the initial setup.

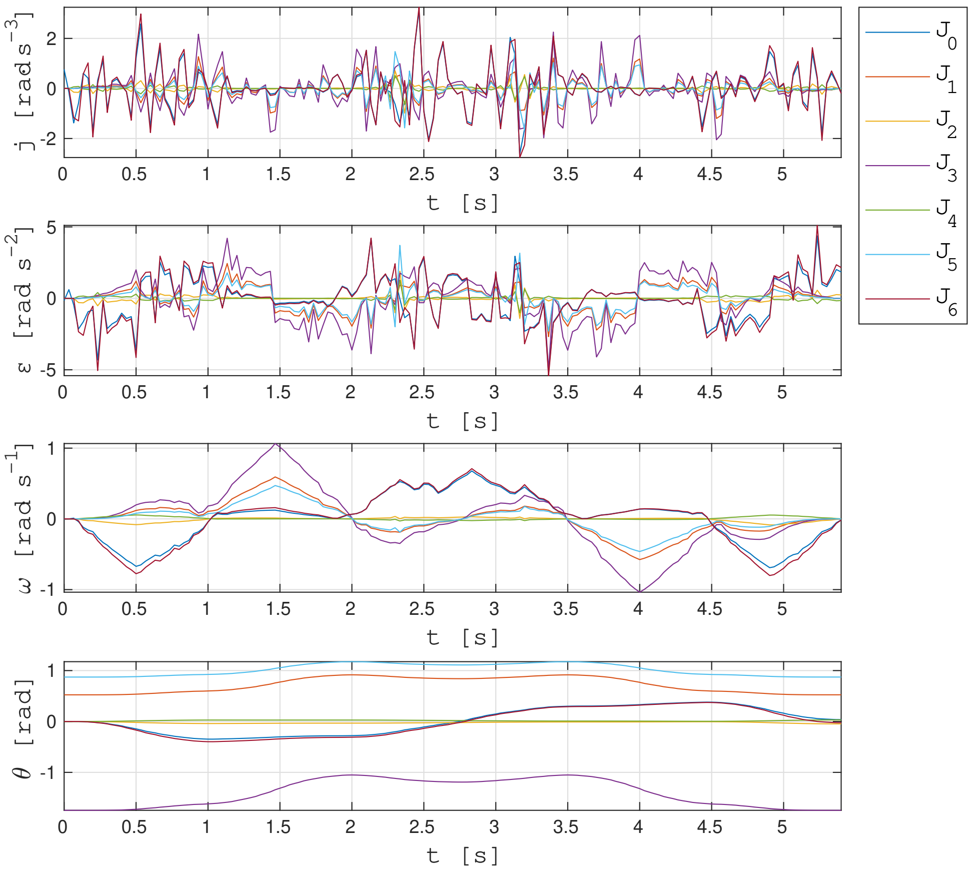


Figure 7.3: Courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$ of a rectangular trajectory computed by the IPTP.

The initial setup was tested on a rectangular path, described in section 7.2, and the results of the IPTP algorithm generated trajectory are shown in figure 7.3. It is clear, that there are quite high oscillations in joint acceleration and jerk curves, which cause rough courses of the joint velocities throughout the movement. This led to the conclusion that the time parameterization of the path was the cause of the problem as no significant issues with the path planner were identified. This will be further elaborated in following sections.

7.5 Parameter tuning

The initially selected IPTP algorithm was then tested in comparison with the second algorithm already included in MoveIt, namely the TOTG, and a recently introduced alternative time parameterization algorithm TOPP-RA, in order to further explore the possibilities of improving the path time parameterization in our setup.

The parameters of the algorithms used for the trajectory generation should be tuned prior to commencement of the experiments on different trajectories to test the algorithms' full potential on our KUKA robot cell. All three path time parameterization algorithms and their settings were tested on a rectangular and circular shaped paths, described in section 7.2. In addition, different parameter settings of the OMPL planner were tested.

7.5.1 IPTP

The IPTP algorithm described in section 3.3.1 is implemented in MoveIt as a class called `IterativeParabolicTimeParameterization`, and its constructor has two arguments:

- `max_iterations`, which determines the maximum number $n_{it_{max}}$ of forward and backward passes of the iterative algorithm.
- `max_time_change_per_it`, which limits the path segment's time extension $\Delta t_{max_{it}}$ when searching for feasible trajectory.

In addition, the acceleration and velocity constraints can be scaled with a constant factor, which can be passed by the parameters when calling the path parameterization function. However, in our experiments, we will always consider the maximal possible acceleration and velocity bounds as specified in section 5.1.

Different settings were tested on the aforementioned paths, with the results given in table 7.1. The default values are on the third line of the table. It is clear that the $n_{it_{max}}$ is at a sufficient value for paths of this length, as the increasing number of iterations does not have effect on the execution time of the generated trajectory and usually increases the computation time of the algorithm.

Table 7.1: The computation time t_{comp} and generated trajectory execution time t_{exec} of the IPTP algorithm for different settings. The maximal number of iterations n_{it} and maximal time change per iteration $\Delta t_{max_{it}}$ parameters were tested.

		Rectangular Path		Circular Path	
$n_{it_{max}}$ [-]	$\Delta t_{max_{it}}$ [s]	t_{comp} [s]	t_{exec} [s]	t_{comp} [s]	t_{exec} [s]
100	0.005	0.0025	3.8053	0.0013	2.9229
150	0.005	0.0026	3.8053	0.0013	2.9229
100	0.010	0.0014	3.8226	7.7568e-04	2.8828
150	0.010	0.0015	3.8226	6.4479e-04	2.8828
100	0.050	7.0210e-04	3.7846	6.3696e-04	4.3644
150	0.050	6.4852e-04	3.7846	5.3068e-04	4.3644

The decreasing value of the $\Delta t_{max_{it}}$ does lead to a faster trajectory in more complex cases, such as the circle shaped path. However, surprisingly, the execution time of the circular shaped trajectory increases with a smaller time change per iteration, while the execution time of the rectangular path decreases with a larger time change per iteration. Based on this observation, we have decided to use the following default settings:

$$\begin{aligned} n_{it_{max}} &= 100 \\ \Delta t_{max_{it}} &= 0.01 \text{ s.} \end{aligned} \tag{7.7}$$

7.5.2 TOTG

This algorithm described in section 3.3.2 is implemented in MoveIt as a class `TimeOptimalTrajectoryGeneration`, which requires the following arguments:

- `path_tolerance`, which determines the maximal distance δ of the computed joint trajectory from the original path configurations. As the configuration space is a multi-dimensional space of joint positions, the path tolerance δ is given in radians.
- `resample_dt`, which determines the sampling period of the computed trajectory in seconds. As the position control frequency of our KUKA robot cell is set to 1 kHz, the value 0.001 s has been chosen for this parameter.

- `min_angle_change` which determines the threshold of minimal joint position difference in radians between two adjacent path points to remove repeated points. There was no issue with redundant path points or points that very close proximity, this parameter was left at the default value 0.001 rad.

Even though parameter δ is called path tolerance, it does not specify the maximal deviation of the final cartesian trajectory from the original cartesian path (designated as trajectory error d and computed as stated in section 7.3). In fact, the path tolerance only limits the change in the manipulator's configuration defined by its joint position values i.e. gives the norm of vector $\delta = |\mathbf{q_2} - \mathbf{q_1}|$ representing the difference between two configuration vectors $\mathbf{q_1}$ and $\mathbf{q_2}$ in the configuration space. The degree to which it is reflected in the trajectory error d is depends on the robot design (joint types, link dimensions ect.) and the specific configuration (the difference in the end effectors position will increase with increasing distance from the axis of rotation). However, we have tested the trajectory generation using different path tolerances δ and noted the maximal trajectory error d_{max} occurred. This can serve to estimate the resulting trajectory error when executing trajectories around the center of the robot's working space. The results are given in table 7.2.

As expected, the smaller the path tolerance δ, the longer the trajectory execution time t_{exec} and the smaller the maximal trajectory error d_{max}. This can be clearly illustrated on both the rectangle and circular shaped path, where increasing path tolerances enable creating shortcuts that lead to quicker trajectories. The default value for the path tolerance was $\delta = 0.1$ rad which is quite high as the resulting maximal trajectory error was around $1.4 \cdot 10^{-3}$ m.

Table 7.2: Computation time t_{comp} and generated joint trajectory execution time t_{exec} of the TOTG algorithm subjected to different values of the maximal deviation δ from the original joint path. The maximal error d_{max} of the generated trajectories compared to the original paths was also noted.

		Rectangular Path		Circular Path	
δ [rad]	d_{max} [m]	t_{comp} [s]	t_{exec} [s]	t_{comp} [s]	t_{exec} [s]
0.0005	$2.1 \cdot 10^{-4}$	0.0357	3.5005	0.0208	2.9287
0.0008	$3.4 \cdot 10^{-4}$	0.0308	3.4963	0.0131	2.6121
0.0010	$4.2 \cdot 10^{-4}$	0.0288	3.4936	0.0126	2.4849
0.0030	$8.0 \cdot 10^{-4}$	0.0297	3.4720	0.0140	2.3565
0.0050	$1.3 \cdot 10^{-3}$	0.0290	3.4546	0.0124	2.3565
0.1000	$1.4 \cdot 10^{-3}$	0.0289	3.4536	0.0141	2.3565

As the path tolerance affects only the preprocessing of the input path of the TOTG algorithm, the computation times remain similar for all settings. The computation time includes the sampling of the trajectory with the frequency of 1kHz. Subsequently, the parameter `path_tolerance` can be chosen for each task differently according to the precision needed; we will use the value of

$$\delta = 0.001 \text{ rad} \tag{7.8}$$

as the resulting error is expected to lie around $4.2 \cdot 10^{-4}$ m which is sufficient for most tasks and is in some cases similar to the error of the trajectories generated by TOPP-RA algorithm, which will be shown in section 7.6 below.

7.5.3 TOPP-RA

The parameters used by the TOPP-RA algorithm are specified in section 6, the ROS Service implementation is described.

The results of the different settings for the parameter `n_gridpoints` which determines the number of points for discretization n_{gridp} of the TOPP-RA algorithm are shown in table 7.3. As the trajectories of different lengths have different number of path points n_{pathp}, it is recommended in the TOPP-RA documentation that the number of grid points be derived from the number of path points of the input path.

It is evident that the increasing number of grid points where the path is discretizied and the controllable sets are computed prolongs the computation time t_{comp} but leads to faster trajectories. The more grid points, the more controllable sets that need to be computed by solving small linear programs. However, a higher density of known controllable sets results in choosing the controls with higher precision, which leads to faster trajectories. The automatic selection of the number of grid points is set by default, which results in the fastest computation time but generates the slowest trajectories. The computation time of the TOPP-RA algorithm includes the time needed for sampling the trajectory with the frequency of 1 kHz.

As the TOPP-RA manual recommends to use more than twice the number of grid points and we have a quite large distance between the adjacent path points, as it will be shown in the next section. The number of grid points will be computed using the following equation:

$$n_{gridp} = 3 \cdot n_{pathp}. \tag{7.9}$$

Table 7.3: The computation time t_{comp} and generated trajectory execution time t_{exec} of the TOPP-RA algorithm subjected to different settings. The effect of the changing number of grid points n_{gridp} derived from the number of original path points n_{pathp} on the resulting trajectory was tested. The automatic selection of n_{gridp} by the TOPP-RA algorithm was also tested.

	Rectangular Path		Circular Path	
n_{gridp} [-]	t_{comp} [s]	t_{exec} [s]	t_{comp} [s]	t_{exec} [s]
Automatically selected	0.0121	3.7620	0.0069	2.8830
$2 \cdot n_{pathp}$	0.0164	3.6880	0.0124	2.8710
$3 \cdot n_{pathp}$	0.0252	3.7090	0.0169	2.8470
$4 \cdot n_{pathp}$	0.0352	3.6820	0.0200	2.8370
$6 \cdot n_{pathp}$	0.0911	3.6520	0.0311	2.8270
$8 \cdot n_{pathp}$	0.1083	3.6240	0.0448	2.8110

7.5.4 Effect of path discretization settings on the time parameterization algorithms

There are three main parameters in MoveIt for specifying yhe manner how the by OMPL planned path will be sampled in to path points, namely:

- `cartesian_path_max_step_translation`, which determines the maximal euclidean distance d_{max} between two adjacent path points,
- `cartesian_path_max_step_orientation`, which determines the maximal change in the orientation between two adjacent path points; however, since there is no need to further limit the orientation of the end effector in our applications when the next parameter is set, we will leave this its value unchanged,
- `cartesian_path_max_joint_distance`, which limits the maximal difference in joint position $\Delta\theta_{max}$ between two adjacent path points.

To ascertain whether the trajectory generation will improve these parameters with different settings, a grid search was executed, where the parameters mentioned in table 7.4 were tested for all three path time parameterization algorithms on the rectangular path described in section 7.2. The default settings for the time parameterization algorithms were used.

The table 7.5 shows the results, where the computation time of the corresponding time parameterization algorithms and the execution time of the generated trajectory were measured to observe the effects of the different

Table 7.4: MoveIt cartesian planner path discretization parameter settings for testing their effects on time parameterization algorithms, namely the maximal step in translation d_{max} between two adjacent path points and maximal joint distance $\Delta\theta_{max}$ between two configurations corresponding to the adjacent path points.

Setting Nr.	d_{max} [m]	$\Delta\theta_{max}$ [rad]
1	0.008	0.04
2	0.01	0.05
3	0.012	0.06
4	0.015	0.075

Table 7.5: The resulting time parameterization computation time t_{comp} and trajectory execution time t_{exec} of the time parameterization algorithms based on different parameter settings of the path planner specified in table 7.4. The rectangular path described in section 7.2 was used.

	IPTP		TOTG		TOPP-RA	
Setting Nr.	t_{comp} [s]	t_{exec} [s]	t_{comp} [s]	t_{exec} [s]	t_{comp} [s]	t_{exec} [s]
1	0.0229	3.7243	0.0497	3.4334	0.0427	3.5990
2	0.0185	3.8226	0.0344	3.4406	0.0316	3.6820
3	0.0169	3.7531	0.0280	3.4536	0.0269	3.6790
4	0.0123	3.7131	0.0264	3.4051	0.0217	3.5570

parameter settings. It is clear that the increasing step size in both the cartesian translation and joint rotation affects the generated trajectories as the trajectory execution times decrease with increasing values. This is because the OMPL generates rough trajectories with small step sizes; due to the numeric inverse kinematics, different configurations are found for path points that lie in a proximity. Even slightly different joint position values of adjacent configurations can result in end effector positions with larger Euclidean distance between them. Consequently, the trajectories are forced to follow a more complex path shape. A larger gap between the points creates more room for smooth transitions between the path points which leads to faster trajectories.

The computation times of the time parameterization algorithms also vary with the different step sizes of translation and joint position. The computation time tends to decrease with the larger step sizes for all three algorithms, which can be explained by fewer path segments needing to be transformed in to a feasible trajectory. Indeed, all three algorithms divide the trajectory computation task in connecting the adjacent path segments, whether with circular blends, as in the TOTG algorithm, through computing controllable sets, as in the TOPP-RA algorithm, or with systematic adjustment of the time difference between two adjacent trajectory points in IPTP.

However, a larger step size can cause problems as the trajectory between the path points usually deviates from the originally intended path, and the bigger the gaps between the path points are, the more room there is for deviation from the planned path. The trajectory segments between the given path points are in principle undefined and can be generated differently with each algorithm. Having regard to the above, we will use a slightly bigger step size than the default settings number 2 in table 7.4 to prevent rough courses of the planned path, specifically

$$\begin{aligned} d_{max} &= 0.012 \text{ m} \\ \Delta\theta_{max} &= 0.06 \text{ rad.} \end{aligned} \tag{7.10}$$

7.6 Comparison of trajectories generated by the individual time parameterization algorithms

Trajectories have been generated for all the paths described in section 7.2, which were planned by the OMPL planner. The visualization and joint trajectory courses of all the generated trajectories can be found in appendix B for detailed study. However, for the sake of clarity, only the circular and rectangular paths were used in this section to illustrate the behavior of each algorithm. The parameter settings of all three time parameterization algorithms are specified in section 7.5 and the applied acceleration and velocity bounds are given in table 5.6 for an unweighted KUKA robot. The errors of the computed trajectories were determined using the methods introduced in section 7.3,and the error of the planned path compared to the initial intended path was computed similarly.

7.6.1 Computation and trajectory execution time

The resulting computation times of each time parameterization algorithm for all paths can be found in table 7.6. It is clear that the IPTP algorithm is significantly faster in all cases, which is also the reason why it is still used by default in MoveIt. The TOPP-RA algorithm had shorter computation times than TOTG in all cases and was approximately 40% faster. Moreover, where necessary, the algorithm is capable of computing the trajectories even faster, as shown in table 7.3. The computation time is directly dependent on the number of grid points of the TOPP-RA algorithm. This, however, also negatively affects the execution time of the computed trajectory as slower trajectories have been found.

Table 7.6: Computation times of the time parameterization algorithms.

	t_{IPTP} [s]	t_{TOTG} [s]	$t_{\text{TOPP-RA}}$ [s]
Rectangle	0.0013	0.0352	0.0218
Circle	0.0006	0.0163	0.0118
Sand Clock	0.0027	0.0858	0.0316
Pick and place	0.0024	0.0484	0.0293

Table 7.7 contains the execution times of the generated trajectories. The IPTP algorithm computes the slowest trajectories, which can be expected, as it does not claim to be a time optimal time parameterization. The TOPP-RA algorithm computes trajectories that are slower than the ones computed by the TOTG algorithm even though it claims to find the time optimal trajectory. This is due to the applied method of computing a continuous path from the input path points generated by the path planner, which is then translated in to the final trajectory. Paths of different lengths and shapes are generated by the two algorithms, and thus both can generate time optimal, yet different trajectories. As we will show in section 7.6.3, TOPP-RA generates longer trajectories for several reasons, which supports these findings.

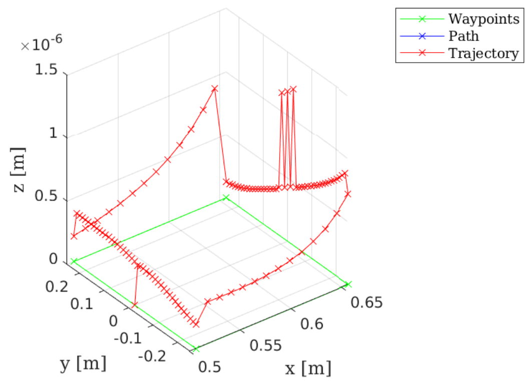
Table 7.7: Trajectory execution times of the time parameterization algorithms.

	t_{IPTP} [s]	t_{TOTG} [s]	$t_{\text{TOPP-RA}}$ [s]
Rectangle	3.7531	3.4936	3.7350
Circle	2.5932	2.4849	2.5940
Sand Clock	4.7461	4.2596	4.4800
Pick and Place	4.5838	4.4772	4.5610

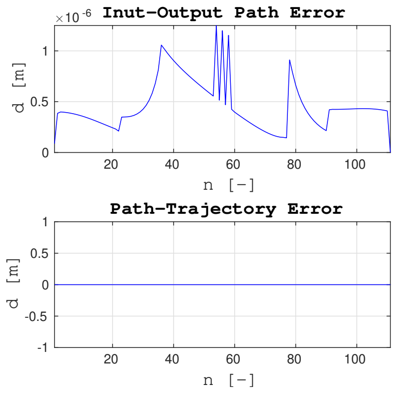
7.6.2 Trajectory sampling

The IPTP is the only one of the three algorithms that does not provide the possibility of sampling the computed trajectory with a certain period. As shown in figure 7.4 the resulting trajectory is returned as a list of the same path points that were given by the path planner, only with added timing. The Euclidean distance between the computed trajectory points and originally intended path also supports these findings, as the trajectory error is zero along the whole path, as shown in figure 7.4b. This, however, does not correspond to reality as the segments between the trajectory points are not defined and the error thus cannot be computed in those segments.

This is probably the cause of the jittery movement of our KUKA robot. The generated trajectory has to be fitted with a quintic spline by the `FollowJointTrajectory` in order to provide the KUKA robot with a position command every millisecond, as described in section 3.4. Subsequently, the trajectory passed for execution to the KUKA robot controller may differ from the generated ones and can even violate the kinodynamic bounds as the fitting of the spline is not limited by the latter. Because of this, we will try to replace the IPTP with one of the remaining time parameterization algorithms.



(a) : Input waypoints, planned path and highlighted trajectory points with red crosses.



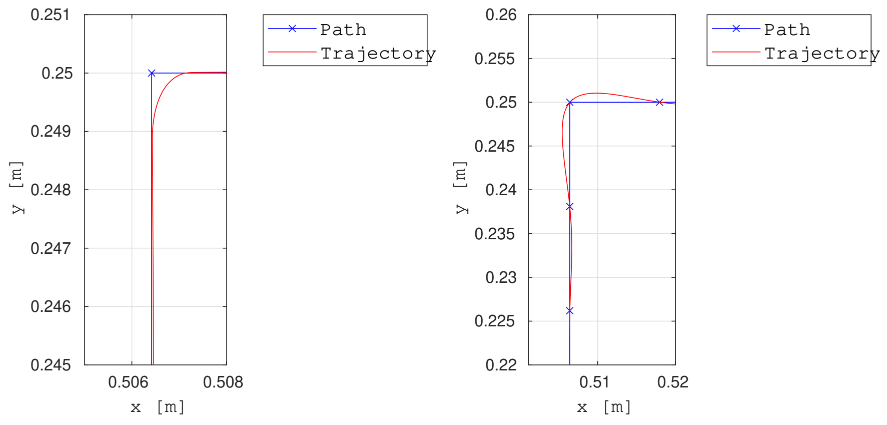
(b) : Path end trajectory error.

Figure 7.4: Figures visualizing the rectangle shaped trajectory generated by the IPTP time parameterization algorithm. The path was planned by the OMPL from the input waypoints.

7.6.3 Deviation of the trajectory from the original path

The main difference between the TOTG and TOPP-RA generated trajectories lies in the manner of handling of the input path points. The TOPP-RA algorithm tries to connect the input path points with a feasible trajectory, i.e. includes the input configurations in the computed trajectory. The TOTG, on the other hand, replaces parts of the input path with arcs that miss the original path points. This can be clearly seen in figure 7.5 on an edge detail of the generated trajectory.

Moreover, larger path segments may deviate from the original path points when using the TOTG algorithm, as shown in figure 7.6a. It is further noticeable on the path-trajectory error displayed in figure 7.6b, where a non-zero error is visible throughout the whole trajectory, with spikes in the edge sections of the rectangular trajectory, as there is the biggest deviation from the original edge path points.

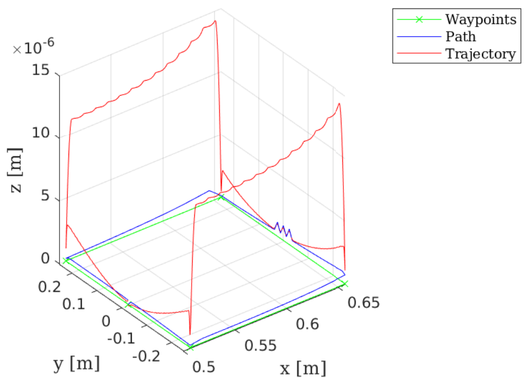


(a) : TOTG. **(b) :** TOPP-RA.

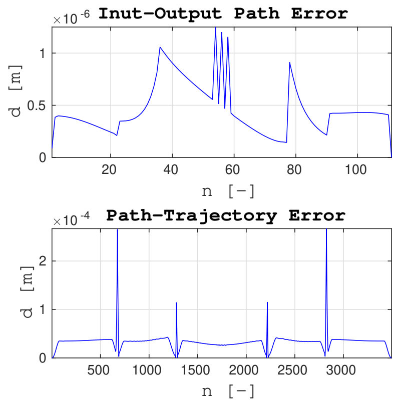
Figure 7.5: The detail of a rectangle shaped trajectories generated bye TOPP-RA and TOTG. The path points generated by the path planner are marked with blue crosses.

The TOPP-RA algorithm generates more oscillating trajectory as it connects the path points. This is shown in figure 7.6d where the spikes in the trajectory error are more frequent, but the deviation always returns to values around 10^{-8} m as the trajectory crosses the corresponding path points. However, it is disputable whether this can be qualified as a trajectory error because only the path points were given to the TOPP-RA algorithm, without specification of the path between them. Therefore, we consider the trajectory error in the sense of the deviation from the originally intended path given by the waypoints.

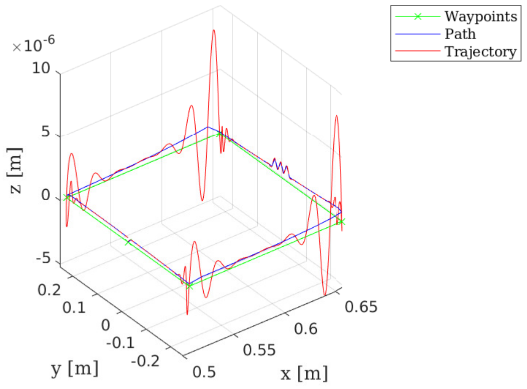
Nonetheless, it is unclear to what extent the omitting of path points in the TOTG generated trajectory causes problems in real tasks. The magnitude of the cartesian trajectory error can be limited by an input parameter. The exact maximal error cannot be set directly because the effect of the joint path tolerance on the cartesian trajectory error depends on the manipulator's configurations. However, the expected values can be estimated, as shown in table 7.2. In our experiments, the trajectory error spikes are usually much smaller than those of the TOPP-RA generated trajectory between the path points, specifically around $5 \cdot 10^{-4}$ m, compared to 10^{-3} m, (i.e. the typical value for the maximal trajectory error of the TOPP-RA algorithm given the selected path discretization step). This applies to all tested path shapes except for the circular path, where the TOTG trajectory error is larger. Moreover, the path-trajectory error is even smaller in most parts of the trajectory, approximately $5 \cdot 10^{-5}$ m. Considering the trajectory as a whole, the TOPP-RA generated trajectory has error throughout the trajectory due to the frequent oscillations. This clearly follows from a comparison of figures 7.6b and 7.6d. The TOTG generated trajectory also respects the start and goal path points as the path-trajectory error is nearly zero at these points; this can be seen in both figures 7.6 and 7.7.



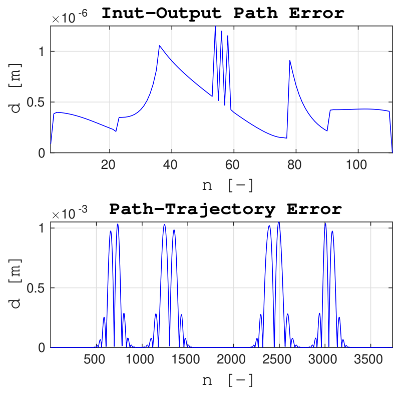
(a) : Visualization of input waypoints, planned path and generated trajectory by TOTG algorithm.



(b) : Path and TOTG generated trajectory error of n path/trajectory poitns.



(c) : Visualization of input waypoints, planned path and generated trajectory by TOPP-RA algorithm.



(d) : Path and TOPP-RA generated trajectory error of n path/trajectory poitns.

Figure 7.6: Figures visualizing the generated rectangular trajectory by the TOPP-RA and TOTG time parameterization algorithms.

Issues with the omitting of path points may, however, occur in high precision tasks where the manipulator needs to pass through the predefined path points. A collision with other objects could also be a problem since only the path points generated by the path planner are checked for collisions. Consequently, if the input path points are omitted, additional collision checking might be necessary because the exact value of the maximal path-trajectory error cannot be predicted with certainty.

7.6.4 Impact of the cartesian path planner on the generated trajectory

The OMPL cartesian planner uses a numeric inverse kinematic task to compute a joint path from the cartesian path as described in section 3.2. This results in rough path courses which can be seen at the input-output path error in figure 7.7b and its spatial visualization in figure 7.7c. We have tried to suppress this behavior with longer path segments (steps), but some oscillations are still apparent in the planned path.

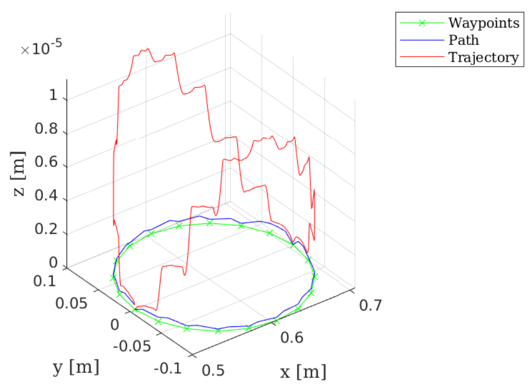
This also causes high oscillations in the TOPP-RA generated path as it follows all the path points of the rough path and connects them. This results in a longer path, which is slower than the TOTG generated path. The TOTG generates smoother trajectories since the replacement of the original path bypasses the rough path courses. This fact is also evident when comparing the trajectories in figures 7.7a and 7.7c.

7.6.5 Joint trajectory courses

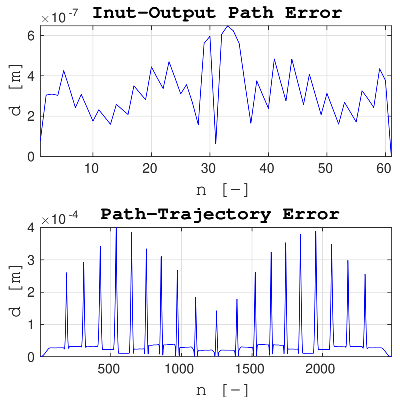
Joint trajectory courses of the circular trajectory were visualized in figure 7.8 in order to compare the smoothness of the generated trajectories. The courses are as can be expected since the "circle" path is actually a polygon; there are spikes in the velocity in the straight segments and lows when the direction of the end effector's movement has to change.

The main differences between the TOTG and TOPP-RA generated joint trajectories, respectively, lie in the acceleration and jerk courses. The TOTG algorithm always chooses the maximal acceleration possible, which can be illustrated by the rectangular courses of the joint acceleration in figure 7.8a. This causes very high joint jerk spikes with the magnitude of around 10^4 rad $\cdot$ s^{-3}. Certain spikes have also occurred in the acceleration courses, which are noticeable in figure 7.9a. There are high oscillations in the acceleration courses at the start of nearly all trajectories generated by TOTG, which cannot be easily explained.

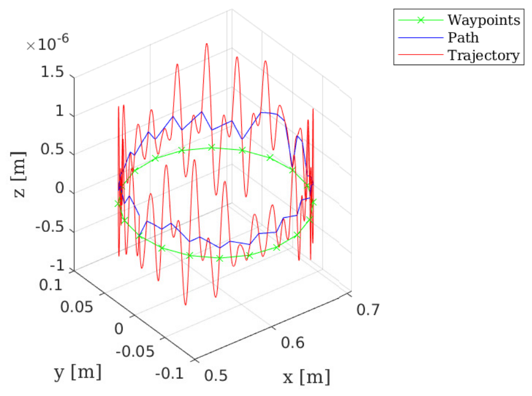
The TOPP-RA algorithm, on the other hand, uses a spline parameterizer for choosing the controls, i.e. the accelerations specified in section 4.2 and further shown in figure 7.8b. The spline parameterization results in much smaller joint jerk spikes, with the magnitude of around $9 \cdot 10^2$ rad $\cdot$ s^{-3}, which are nearly ten times smaller than the ones in the TOTG trajectory.



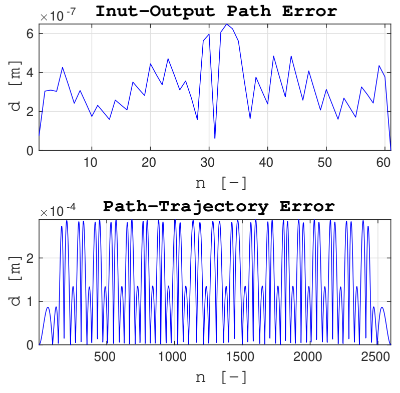
(a) : Visualization of input waypoints, planned path and generated trajectory by TOTG algorithm.



(b) : Path and TOTG generated trajaectory error of n path/trajectory poitns.



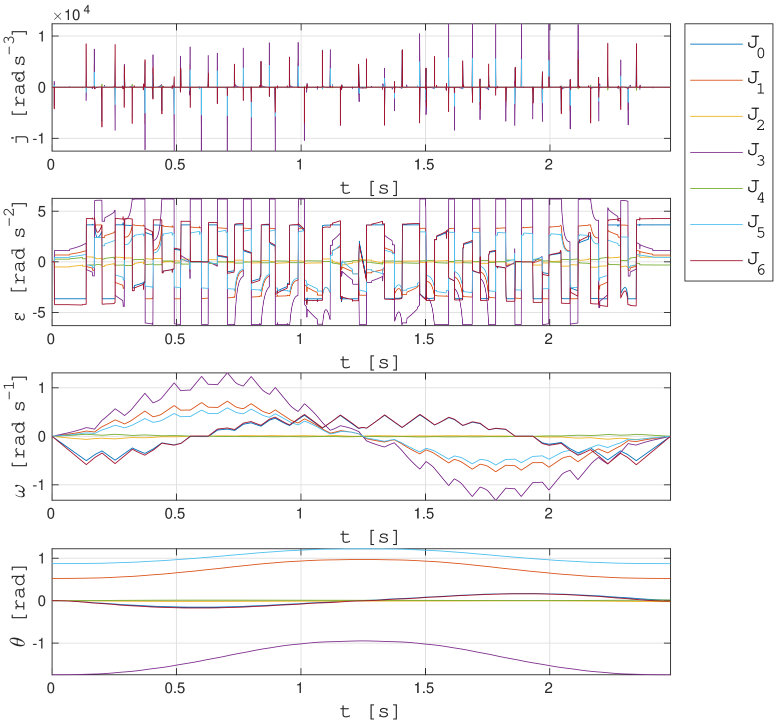
(c) : Visualization of input waypoints, planned path and generated trajectory by TOPP-RA algorithm.



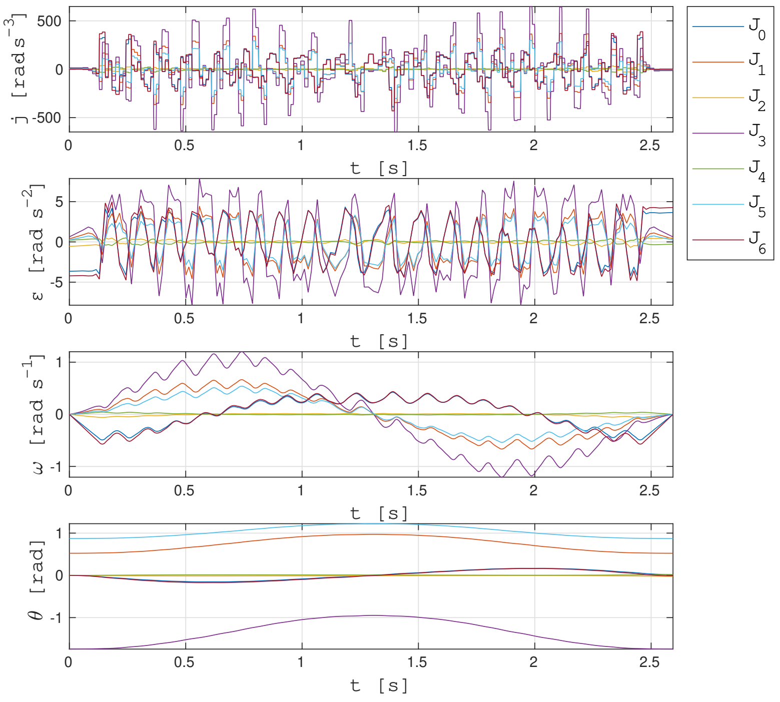
(d) : Path and TOPP-RA generated trajectory error of n path/trajectory poitns.

Figure 7.7: Figures visualizing the generated circular trajectory by the TOPP-RA and TOTG time parameterization algorithms.

Smaller jerk spikes of the TOPP-RA generated trajectory should result in a smoother motion of the manipulator. Since the KUKA robot is position controlled and the low-level movement is done by an embedded controller, the final commanded acceleration by the KUKA robot controller should have much smoother courses then those computed by TOTG.



(a) : Joint courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$ computed by TOTG.



(b) : Joint courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$ computed by TOPP-RA.

Figure 7.8: Figures visualizing the generated and executed circle shaped trajectory by the TOPP-RA and TOTG time parameterization algorithms.

7.7 Trajectory execution by the KUKA robot

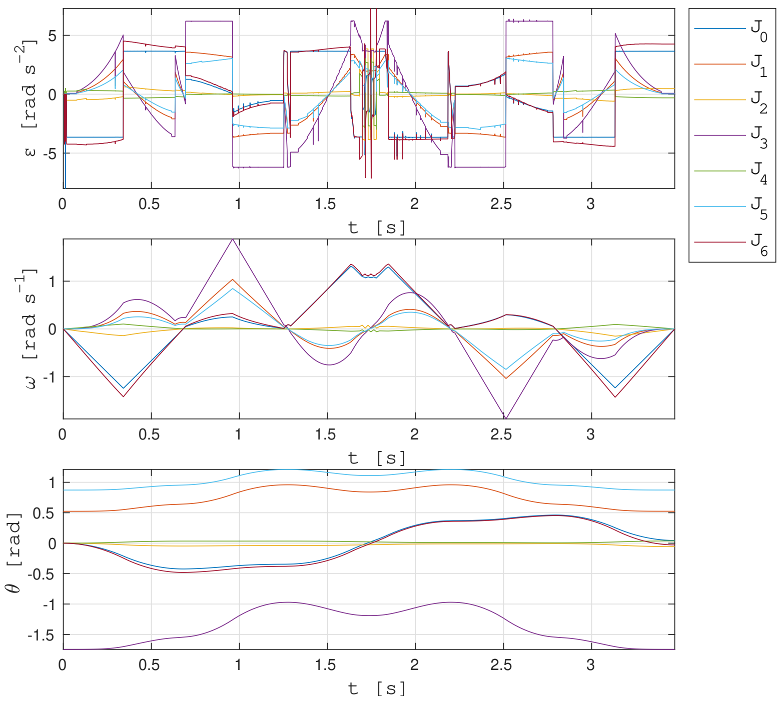
Following the comparison of the TOPP-RA and TOTG time parameterization algorithms, the generated rectangular trajectories were executed by the KUKA LBR iiwa 7 r800 robot to test the behavior on the real setup. The resulting courses are shown in figures 7.9 and 7.10, where the measured joint positions and the velocities and accelerations, both derived from the joint positions, are compared with the computed ones. The joint acceleration courses had to be smoothed out for clarity using a moving average filter with the span of 20 samples, i.e. milliseconds, as the data about the KUKA robot is forwarded to ROS every millisecond.

The joint courses of the executed trajectory correspond to the theoretical ones. The execution times approximately correspond to the computed times as shown in table 7.8. The TOTG and TOPP-RA joint acceleration courses, respectively, are quite similar. Due to the noise of the acceleration courses, it is hard to estimate to what degree the high jerk and smaller acceleration peaks in the courses of the TOTG generated trajectory were eliminated by the KUKA robot controller. Considering the similarity to the TOPP-RA courses and the observation of the executed movement, we believe that the final trajectory is in this case sufficiently smooth for our purposes, however, it does not always have to be the case.

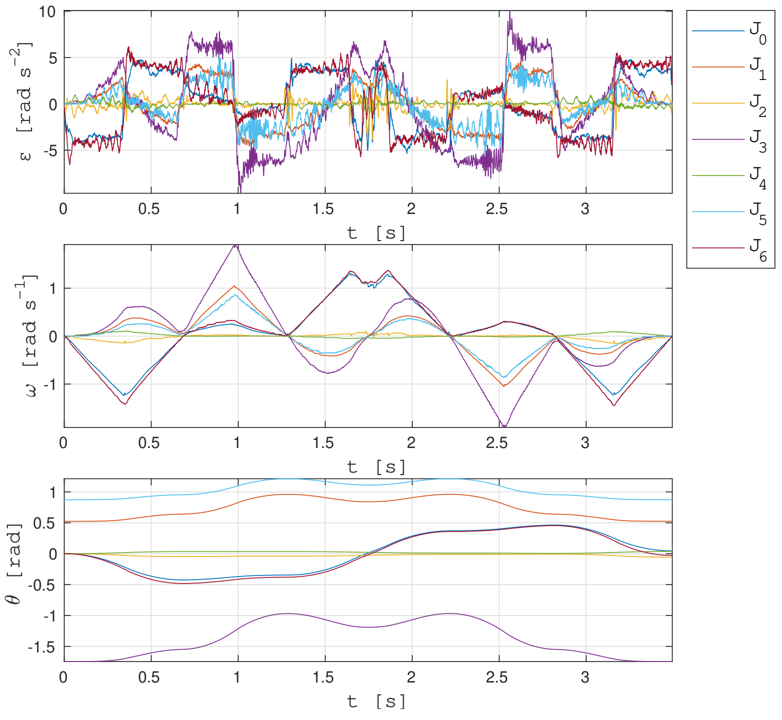
Table 7.8: The computed and actual trajectory execution times of the TOTG and TOPP-RA algorithms on a rectangular path.

	t_{TOTG} [s]	$t_{\text{TOPP-RA}}$ [s]
Computed	3.4936	3.7350
Executed	3.4950	3.7360

The courses of the executed TOPP-RA trajectory show small oscillations in joint accelerations and velocity, which are caused by rough path courses and the design of the algorithm, as mentioned above, and are consequently forwarded for execution.

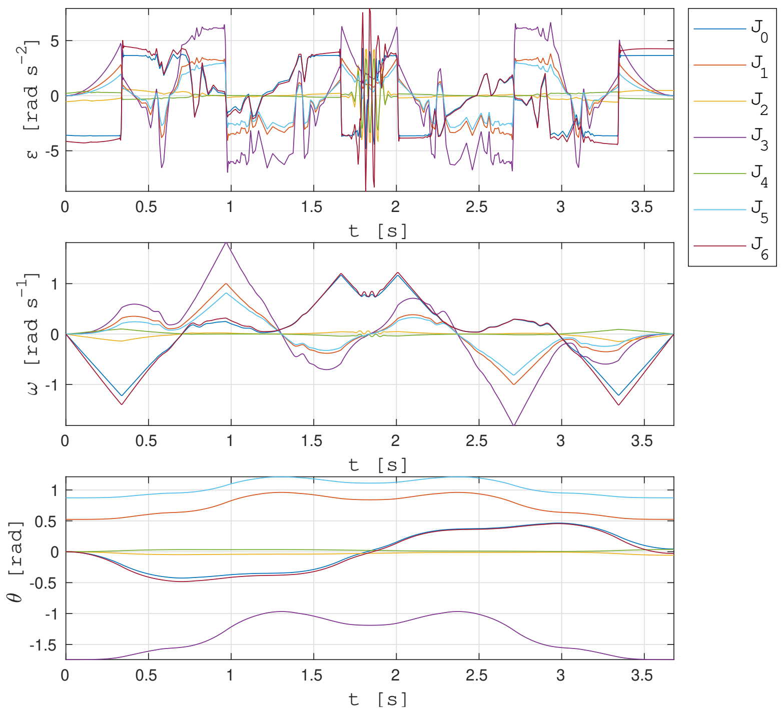


(a) : Computed joint courses of joint position θ, velocity ω, and acceleration ε for joints J_i, $i = 1, ..., 7$.

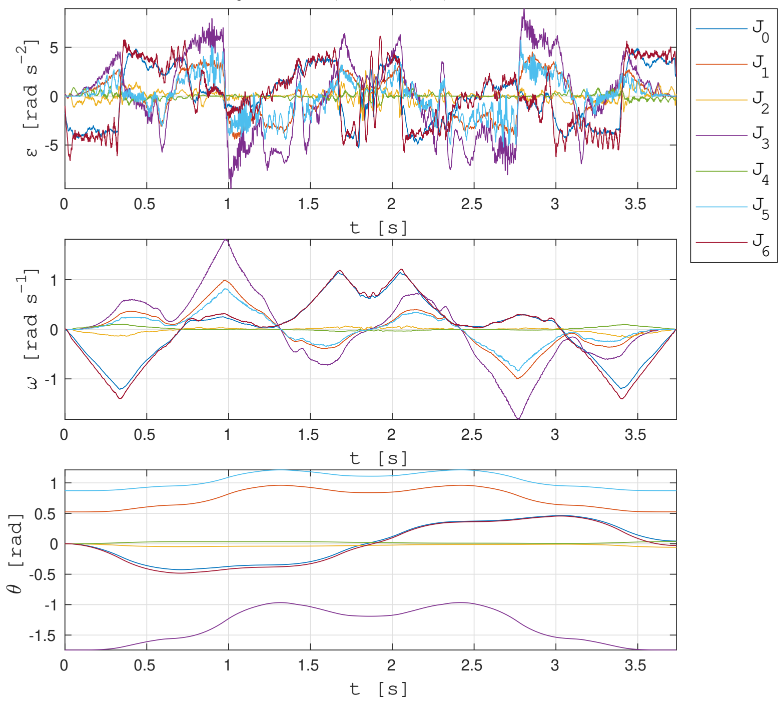


(b) : Joint courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$ executed by the KUKA robot.

Figure 7.9: Comparison of the joint courses of a rectangle shaped trajectory computed by the TOTG time parameterization algorithms and executed by the KUKA robot.



(a) : Computed joint courses of joint position θ, velocity ω, and acceleration ε for joints J_i, $i = 1, ..., 7$.



(b) : Joint courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$ executed by the KUKA robot.

Figure 7.10: Comparison of the joint courses of a rectangle shaped trajectory computed by the TOPP-RA time parameterization algorithm and executed by the KUKA robot.

7.8 Discussion of the results

The above-described characteristics of the respective path time parameterization algorithms follow from all the generated trajectories listed in appendix B. The IPTP time parameterization has been declared unfit for our setup as it was probably responsible for the jittery movement of the manipulator. Nonetheless, we can see the reason why it is still widely used by default in MoveIt as the computation times are nearly ten times shorter in comparison to the other algorithms and its faults were less noticeable at lower motion speeds.

The execution of the trajectories did not reveal any other aspects where the TOPP-RA and TOTG algorithms would deviate. Both trajectories were executed as expected, and the biggest difference between them remains the manner how they handle the path points and, consequently and the shapes of the resulting trajectories. The difference in the speed of the algorithms is also an important aspect.

The TOTG algorithm's disadvantages include longer computation time; the omitting of the input path points, and the resulting deviation of the generated trajectory from the original path in larger segments. The error is mostly small but not insignificant in view of possible collisions. Moreover, the maximal deviation of the trajectory from the original path, i.e. the path-trajectory error, cannot be set directly because the replacing of the initial path is done in the configuration space of the robot. The small spikes in the acceleration courses could also cause potential jitters, mostly at the start of the trajectories. Nonetheless, the algorithm handles well the rough input paths for path tolerances $\delta > 10^{-3}$ and, consequently, generates faster trajectories than TOPP-RA. For smaller path tolerances, the TOTG generated algorithms also start to oscillate due to the path shape.

The TOPP-RA algorithm's drawback, on the other hand, lies in oscillating and thus slower trajectories caused by rough courses of the planned path. The oscillations cause even larger deviations from the originally intended path than those resulting from the TOTG algorithm, which is a toll for respecting the input path points and creating a trajectory that connects them. This behavior could be suppressed by choosing even larger step between path points; this would, however, lead to a loss of precision as there would be fewer points from the original path to follow and deviations could arise in larger segments. A more densely sampled path would be a solution for larger deviations between the trajectory points provided that it is sufficiently smooth. This was not the case of the paths we have used, though. The

TOPP-RA generated trajectories are thus highly dependent on the shape and density of the planned path. Nonetheless, the TOPP-RA algorithm presents better results in terms of the computation time. Our experimetns revealed that TOPP-RA is notably faster then the TOTG algorithm, and indeed can compute the trajectory nearly twice as fast in some cases.

It should also be noted, that the tested version of TOPP-RA is now obsolete. Newer versions are now available and further developed, a C++ API is also in progress, so there is a chance that the TOPP-RAs capabilities will improve in our setup. Moreover, TOPP-RA is equipped with other useful features, such as generating a trajectory of a defined duration or using other parameterizers, which we did not include in this thesis. We also did not cover the robustness of both algorithms, the TOPP-RA documentation claims 100% success rate of finding a feasible time parameterization by the algorithm, the statistics for the TOTG algorithm are unknown and possible issues may arise.

In summary, the TOPP-RA time parameterization does seem to be a perspective solution remedying the issues with our KUKA robot cell, despite the consequences of the inaccurate path planning, probably caused by the inverse kinematics. Nonetheless, the TOTG algorithm is also a suitable solution, it provides stable trajectories that might be more suitable for certain applications than TOPP-RA. Indeed, even though the trajectories are slightly deviated, they follow the originally intended path more accurately and without oscillations. Moreover, the path tolerance can be set according to the task that is being executed. The final path-trajectory error can, however, be only estimated because the limitations of the generated trajectory are applied in the configuration space of the manipulator. Some additional collision checking might therefore be necessary for larger path tolerances.

Chapter 8

Conclusion

In this thesis, we have described the theory of path planning and trajectory computation of a robotic manipulator with the aim to better understand the principles of motion planning of a KUKA LBR iiwa 7 r800 robotic cell and the issue of its jittery movement. The motion planning and execution pipeline of a *Robot Operating System* (ROS) framework including its MoveIt motion planning package were explained, as these software tools are used to control the KUKA robot and are crucial for understanding the movement of the manipulator.

Both parts of the MoveIt trajectory generation process were explored. First, the settings and the planned paths of the *Open Motion Planning Library* (OMPL) used for path planning were tested. Next, two algorithms for the time parameterization of the planned path were analyzed, namely the *Iterative Parabolic Time Parameterization* (IPTP) and *Time Optimal Trajectory Generation* (TOTG). Their principles were explained and their behavior tested using different parameter settings. Both algorithms were compared with the recently introduced *Time-Optimal Path Parameterization based on Reachability Analysis* (TOPP-RA) algorithm on a set of four trajectories; the resulting courses were visualized and are shown in appendix B. A ROS service has been written to connect the TOPP-RA algorithm with ROS.

We found out that the joint acceleration bounds of the KUKA robot were missing in the MoveIt configuration. Since these bounds, together with the joint velocity bounds, are necessary for computing a correct trajectory, the worst case scenario values were estimated by a series of experiments. The actual acceleration limits depend on the robots configuration and would require the identification of a dynamic model, which is out of the scope of this thesis.

Based on the results of the testing of different parameter settings for each time parameterization algorithm and the completed experiments, we made the following conclusions. Firstly, the IPTP algorithm is problematic in several aspects: the implementation does not use parabolic, but rather only linear functions for trajectory computation, and it does not offer functions for sampling of the final trajectory. The ROS joint controller is thus forced to interpolate the resulting trajectory, consequently modifying the final trajectory, which can subsequently violate the kinodynamic bounds of the manipulator. In addition, high oscillations occurred in the joint acceleration courses. Therefore, we have decided to replace the IPTP algorithm with one of the two remaining algorithms in our setup, even though IPTP is significantly faster in computing the trajectories than the other algorithms.

Secondly, the TOTG and TOPP-RA algorithms differ in the manner of handling the planned path passed to them in the form of a list of path points and their time parameterization computation time. The TOTG algorithm replaces the path segments with circular blends, which results in a trajectory that deviates from the passed path points. The deviation is non-zero throughout the whole trajectory, except for the start and goal points, which is the primary drawback of this algorithm. There are also spikes in the acceleration courses of the computed trajectories, mostly at the beginning, which could potentially cause jittery movement. Nonetheless, this is compensated to a certain degree by the KUKA robot controller. TOTG also generates faster trajectories then TOPP-RA as the path preprocessing crates smoother trajectories with shortcuts in the form of the circular blends.

The TOPP-RA algorithm, on the other hand, connects the path points with a trajectory. This, however, results in an oscillating movement as the trajectory follows a path with rough courses, which are probably caused by the numeric inverse kinematics used by the Open Motion Library planner. At the peaks, the resulting deviations from the path exceed the maximal deviations of the TOTG generated trajectories. There is also an issue with the compatibility of the TOPP-RA software with the ROS 1 distribution. For this reason, an obsolete version of TOPP-RA was used in this thesis, which is nonetheless capable of computing the trajectories much faster than the TOTG parameterization.

Finally, both the TOTG and TOPP-RA algorithms can be a suitable solution for our setup, albeit in different situations. The TOTG generated trajectories have only a small, yet not insignificant error from the original path, which can, to a certain degree, be limited by the path tolerance parameter. The rough courses of the planned path are handled well because the deviation of the trajectory eliminates small the serrated path. This is TOTG's advantage compared to TOPP-RA and TOTG is thus a suitable solution for situations where a rough path causes high oscillations in the TOPP-RA generated trajectories and the deviation from the original path is not a problem

The TOPP-RA algorithm, on the other hand, respects the planned path and is notably faster in computing the time parameterization then the TOTG algorithm even though it strongly dependent on the smoothness and density of the discrete input path. The TOPP-RA would also be a clear choice thanks to its smoother acceleration courses, were it not for the rough paths from which the trajectory is computed. It also offers some additional features and could present a more robust solution. Given that the algorithm is being further developed, its capabilities might possibly improve in our setup.

The above summarized work covers all the guidelines given for this thesis. The main contribution of this thesis lies in the description and explanation of a frequently encountered issue with the time parameterization algorithm selected by default in MoveIt, and in description and comparison of other time parameterization algorithm which can be used instead. As ROS and MoveIt software are widely used and there are many occurrences of people reporting the same issue with the time parameterization, we present them with different options for replacing the IPTP algorithm and describe their advantages and drawbacks. There is also a discussion among the MoveIt developers about including TOPP-RA into the MoveIt package, so this thesis can also be useful for anyone who is considering this option.

For further work, we recommend inspecting the behavior of the OMPL planner and the rough paths as these cause problems with the TOPP-RA algorithm. Moreover, the execution of the generated trajectories could be further explored to find the cause of the significant difference between the estimated and verified joint acceleration bounds, respectively, as presented in this thesis. It would be interesting to use other methods of estimation of the joint acceleration bounds to verify or improve our results.

Bibliography

[1] Reza N. Jazar. *Theory of applied robotics. kinematics, dynamics, and control.* 2nd ed. New York: Springer, c2010. ISBN: 978-1441917492.

[2] Peter Corke. *Robotics, vision and control. fundamental algorithms in MATLAB.* 2nd ed. Berlin: Springer, 2013. ISBN: 978-3-642-20143-1.

[3] Bruce Donald et al. "Kinodynamic motion planning". In: *Journal of the ACM* 40.5 (1993), pp. 1048–1066. ISSN: 0004-5411. DOI: `10.1145/174147.174150`.

[4] *ROS Introduction.* URL: `http://wiki.ros.org/ROS/Introduction` (visited on 04/05/2021).

[5] *ROS Concepts.* URL: `http://wiki.ros.org/ROS/Concepts` (visited on 04/05/2021).

[6] *MoveIt Documentation. Planners.* 2016. URL: `https://moveit.ros.org/documentation/planners/` (visited on 04/01/2021).

[7] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82. ISSN: 1070-9932. DOI: `10.1109/MRA.2012.2205651`.

[8] Rice University Kavraki Lab. *Open Motion Planning Library: A Primer.* Huston, Texas, 2021. URL: `https://ompl.kavrakilab.org/OMPL_Primer.pdf` (visited on 04/01/2021).

[9] Jia Pan, Sachin Chitta, and Dinesh Manocha. "FCL. A general purpose library for collision and proximity queries". In: *2012 IEEE International Conference on Robotics and Automation.* IEEE, 2012, pp. 3859–3866. ISBN: 978-1-4673-1405-3. DOI: `10.1109/ICRA.2012.6225337`.

[10] R. Smits. *KDL: Kinematics and Dynamics Library.* URL: `http://www.orocos.org/kdl` (visited on 04/01/2021).

[11] *MoveIt Time Parameterization Algorithms.* URL: `http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/time_parameterization/time_parameterization_tutorial.html#time-parameterization-algorithms` (visited on 04/05/2021).

[12] *MoveIt Iterative Spline Parameterization issues.* URL: `https://github.com/ros-planning/moveit/issues/2183%5C#issue-650150134` (visited on 04/10/2021).

[13] Bruno Siciliano. *Robotics. modelling, planning and control.* 1st ed. London: Springer, c2010. ISBN: 978-1-84628-641-4.

[14] Friedrich Lange and Alin Albu-Schäffer. "Iterative path-accurate trajectory generation for fast sensor-based motion of robot arms". In: *Advanced Robotics* 30.21 (2016-08-29), pp. 1380–1394. ISSN: 0169-1864. DOI: `10.1080/01691864.2016.1222307`.

[15] Tobias Kunz and Mike Stilman. "Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity". In: *Robotics: Science and Systems VIII.* Robotics: Science and Systems Foundation, 2012-07-09, pp. -. ISBN: 9780262519687. DOI: `10.15607/RSS.2012.VIII.027`.

[16] *MoveIt implementation of Time Optimal Trajectory Generation algoritm.* URL: `https://github.com/ros-planning/moveit/blob/melodic-devel/moveit_core/trajectory_processing/src/time_optimal_trajectory_generation.cpp` (visited on 04/10/2021).

[17] *ROS joint_trajectory_controller package.* URL: `http://wiki.ros.org/joint_trajectory_controller` (visited on 05/06/2021).

[18] Dave Coleman. *ROS Control documentation.* URL: `https://github.com/ros-controls/ros_control/blob/noetic-devel/ros_control/documentation/gazebo_ros_control.pdf` (visited on 05/06/2021).

[19] Quang-Cuong Pham. "A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm". In: *IEEE Transactions on Robotics* 30.6 (2014), pp. 1533–1540. ISSN: 1552-3098. DOI: `10.1109/TRO.2014.2351113`.

[20] Hung Pham and Quang-Cuong Pham. "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis". In: *IEEE Transactions on Robotics* 34.3 (2018), pp. 645–659. ISSN: 1552-3098. DOI: `10.1109/TRO.2018.2819195`.

[21] *Suggestion to include TOPP-RA in to MoveIt.* URL: `https://github.com/ros-planning/moveit/issues/2110%5C#issue-624468617` (visited on 04/10/2021).

[22] Quang-Cuong Pham and Olivier Stasse. "Time-Optimal Path Parameterization for Redundantly Actuated Robots. A Numerical Integration Approach". In: *IEEE/ASME Transactions on Mechatronics* 20.6 (2015), pp. 3257–3263. ISSN: 1083-4435. DOI: `10.1109/TMECH.2015.2409479`.

[23] *TOPP-RA Github repository.* URL: https://github.com/hungpham2511/toppra (visited on 04/12/2021).

[24] *KUKA LBR iiwa 7 r800.* URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa (visited on 04/17/2021).

[25] *Pub Spez LBR iiwa en.* 5th ed. Augsburg,Germany: KUKA Roboter GmbH, 2015. URL: http://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Spez_LBR_iiwa_en.pdf (visited on 04/17/2021).

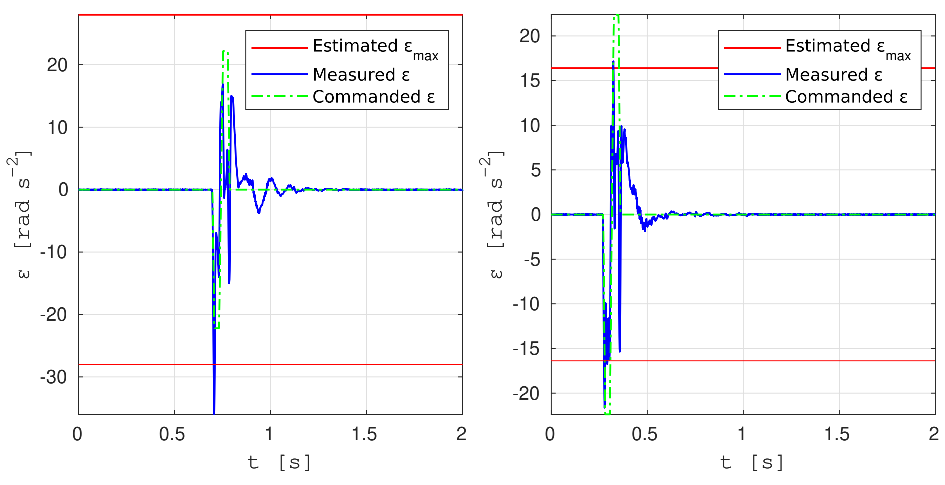
[26] *KUKA Sensitive robotics LBR iiwa.* Augsburg,Germany: KUKA Roboter GmbH, 2017. URL: https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_lbr_iiwa_brochure_en.pdf?rev=5a25f7eac825492e92af6343dbf5bc6b (visited on 04/17/2021).

[27] Ken Anderson. *IterativeParabolicTimeParameterization.* URL: https://github.com/ros-planning/moveit/blob/melodic-devel/moveit_core/trajectory_processing/src/iterative_time_parameterization.cpp (visited on 04/05/2021).

[28] *Topp_ros package repository.* URL: https://github.com/larics/topp_ros (visited on 04/23/2021).

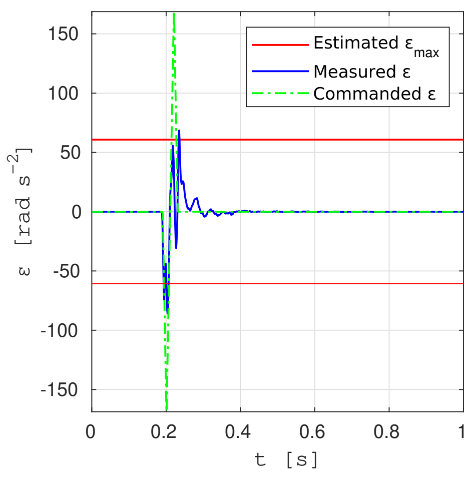
Appendix A

Comparison of estimated joint acceleration bounds of KUKA LBR iiwa 7 r800

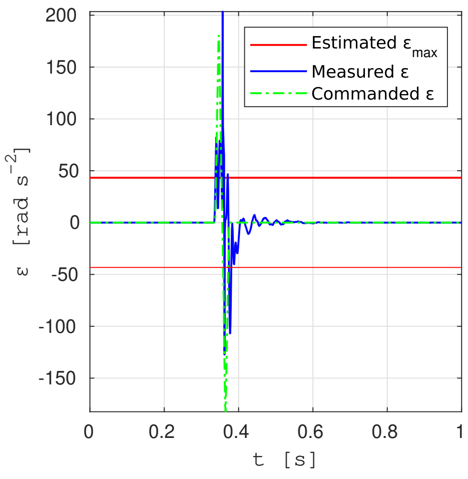


(a) : Courses for joint J_1 **(b) :** Courses for joint J_2

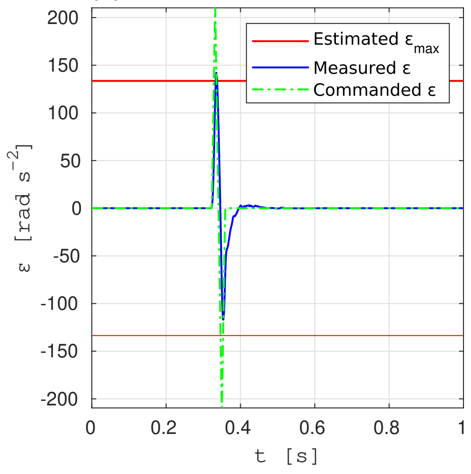
Figure A.1: Comparison of estimated joint acceleration bounds ε_{max}, measured joint acceleration ε and commanded joint acceleration ε by KUKA robot controller for joints J_i, $i = 1, 2$ in response to a position step input of the magnitude $\Delta\theta_i = 2^o$.



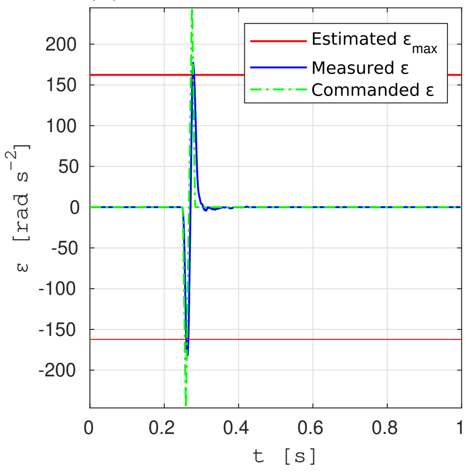
(a) : Courses for joint J_3



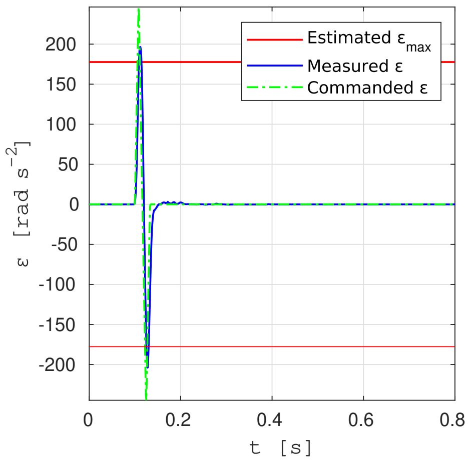
(b) : Courses for joint J_4



(c) : Courses for joint J_5



(d) : Courses for joint J_6

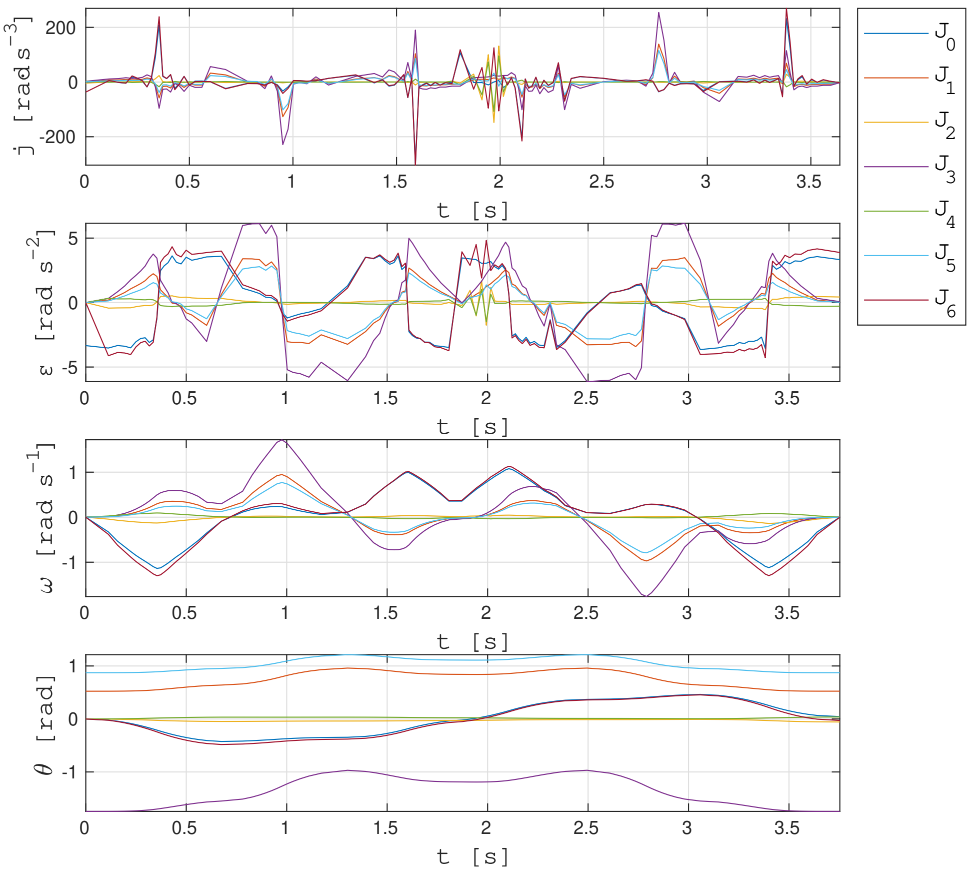


(e) : Courses for joint J_7

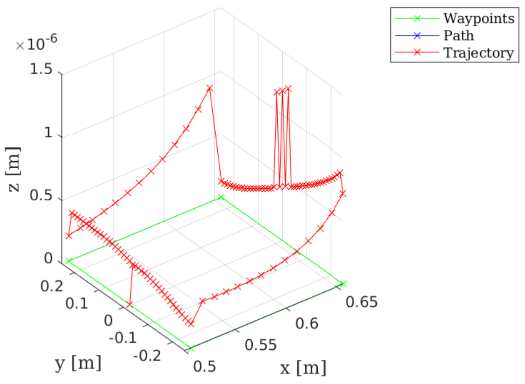
Figure A.2: Comparison of estimated joint acceleration bounds ε_{max}, measured joint acceleration ε and commanded joint acceleration ε by KUKA robot controller for joints J_i, $i = 2, ..., 7$ in response to a position step input of the magnitude $\Delta\theta_i = 2^o$.

Appendix B

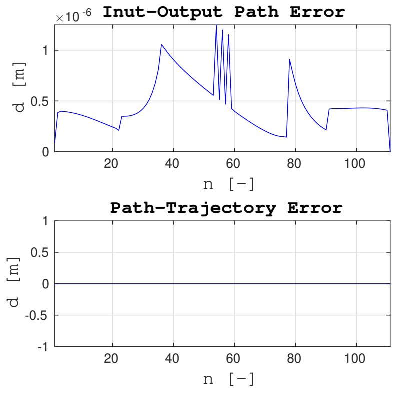
Trajectories generated by the IPTP, TOTG and TOPP-RA time parametrization algorithms



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

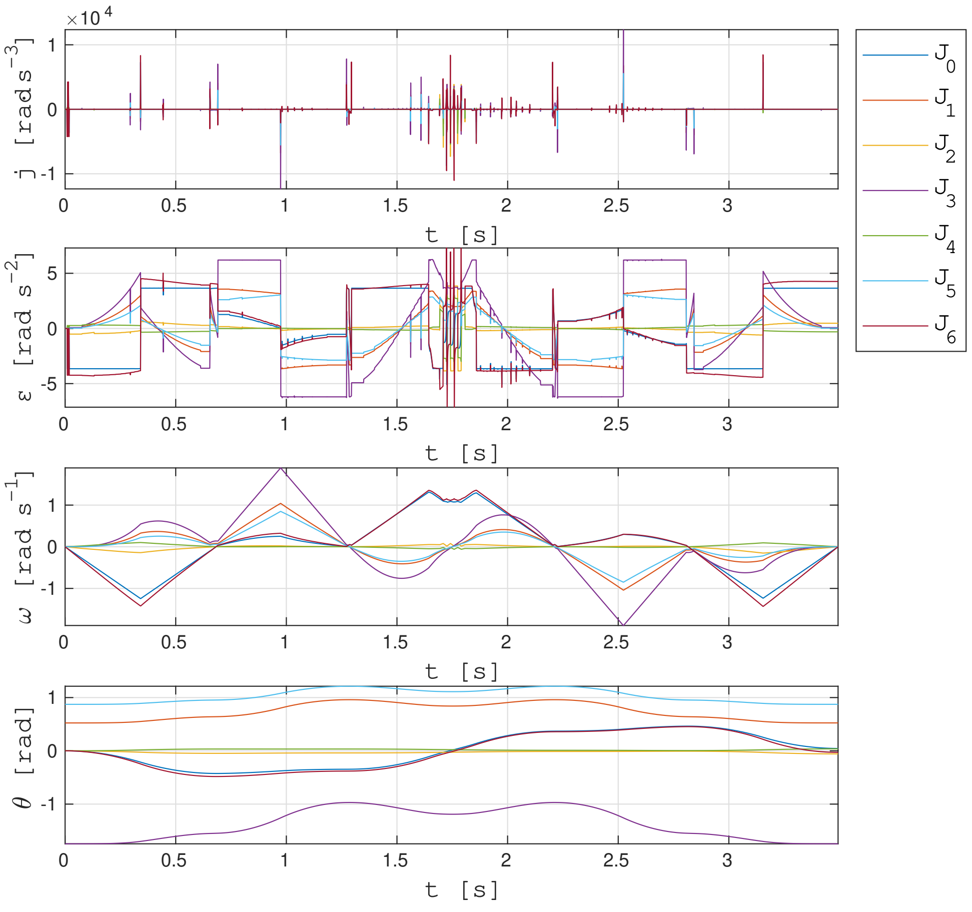


(b) : Visualization of the input waypoints, planned path and generated trajectory.

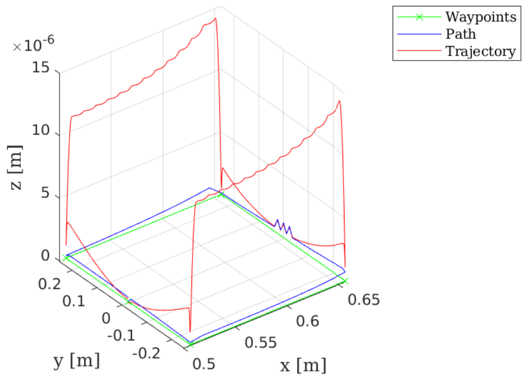


(c) : Path and trajectory error.

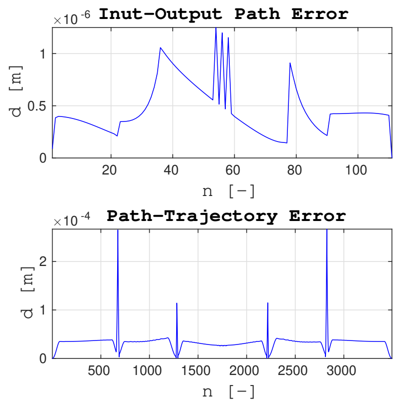
Figure B.1: Figures visualizing a rectangle shaped trajectory generated by the IPTP time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

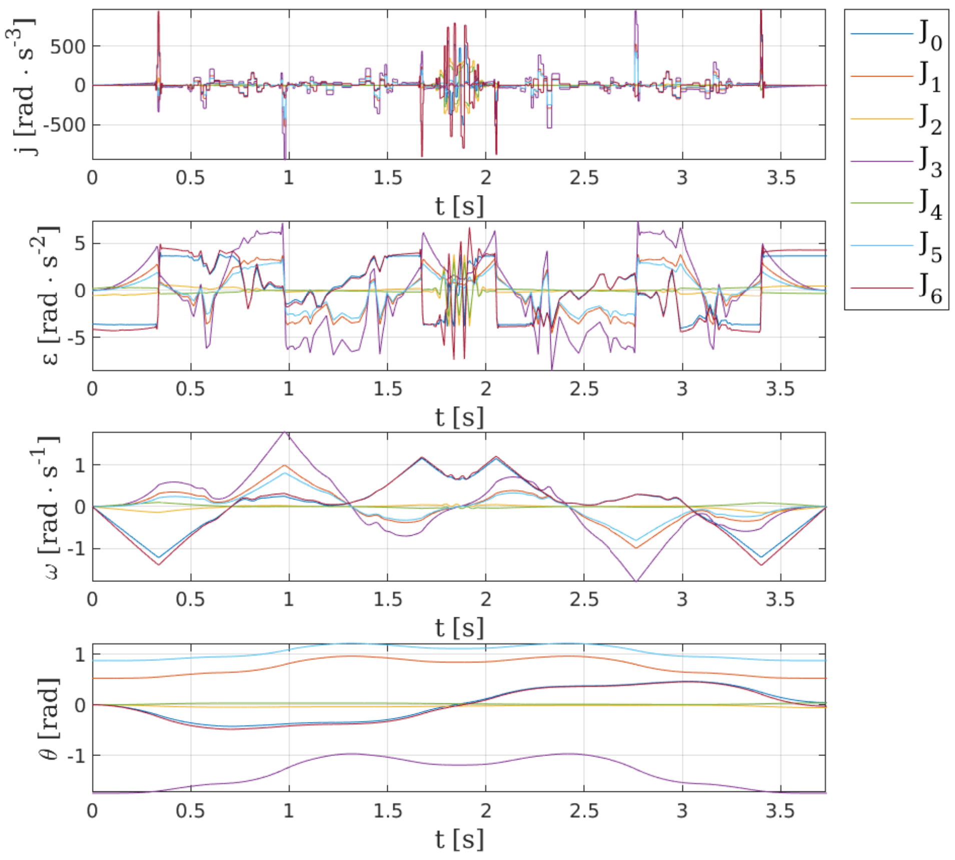


(b) : Visualization of the input waypoints, planned path and generated trajectory.

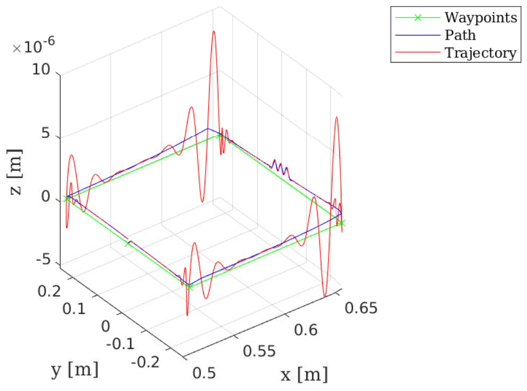


(c) : Path and trajectory error.

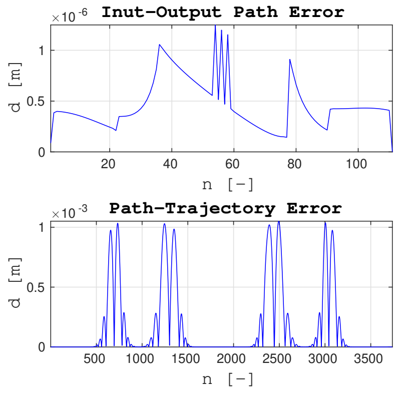
Figure B.2: Figures visualizing a rectangle shaped trajectory generated by the TOTG time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

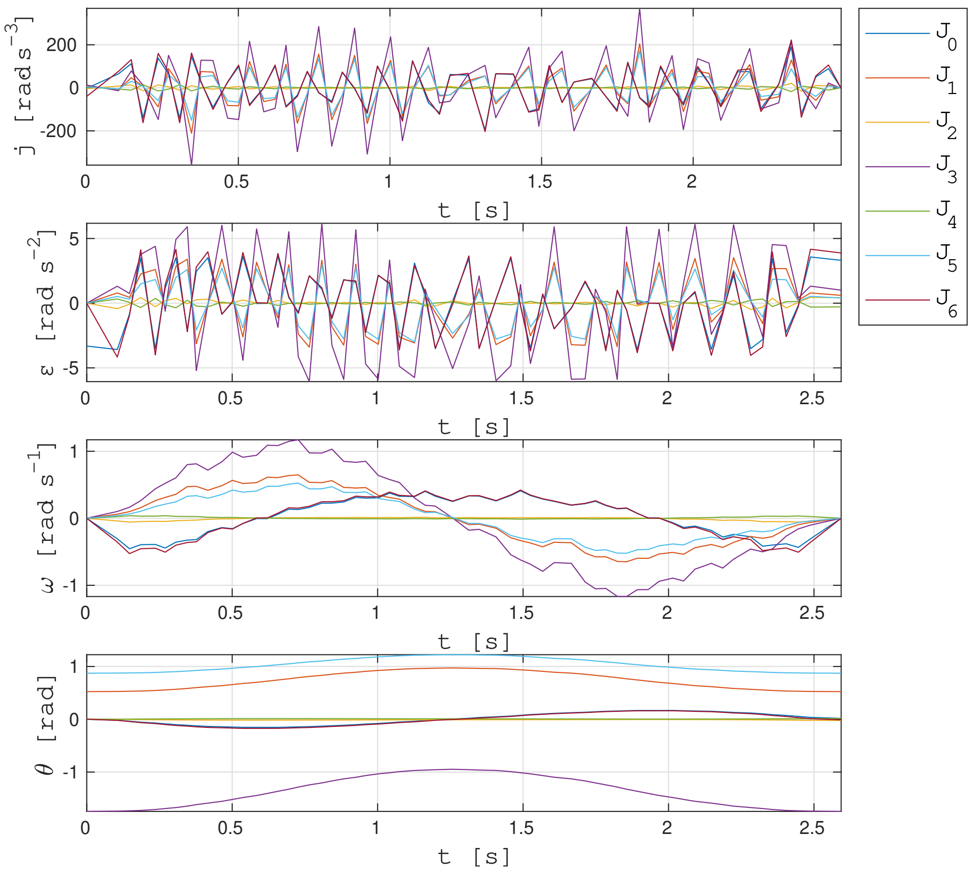


(b) : Visualization of the input waypoints, planned path and generated trajectory.

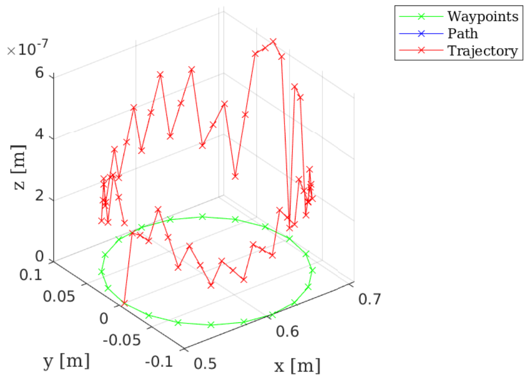


(c) : Path and trajectory error.

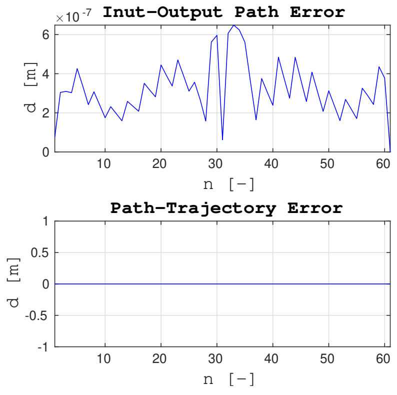
Figure B.3: Figures visualizing a rectangle shaped trajectory generated by the TOPP-RA time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

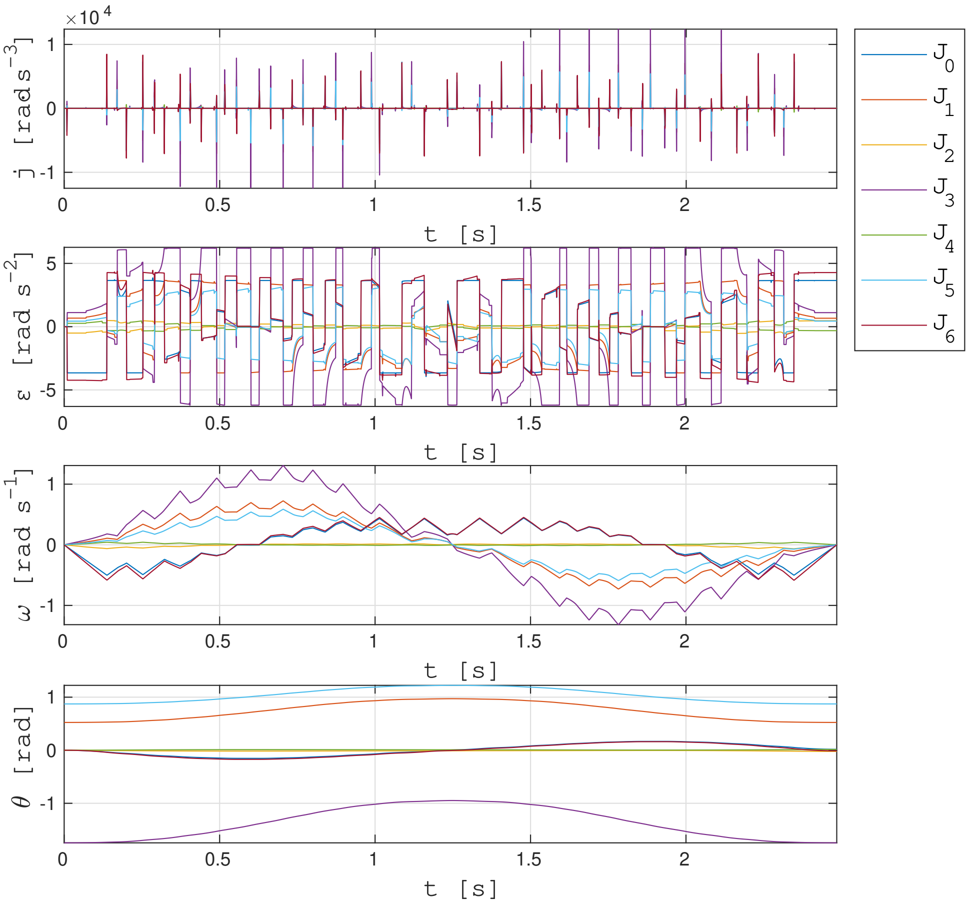


(b) : Visualization of the input waypoints, planned path and generated trajectory.

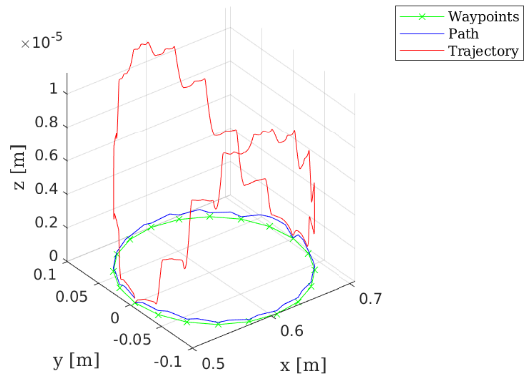


(c) : Path and trajectory error.

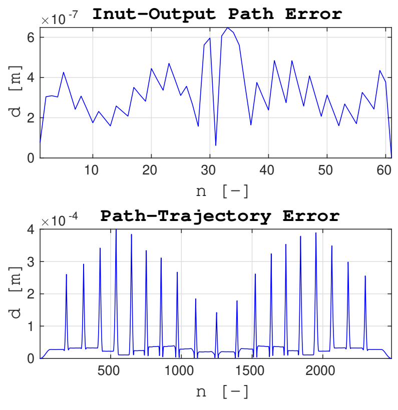
Figure B.4: Figures visualizing a rectangle shaped trajectory generated by the IPTP time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

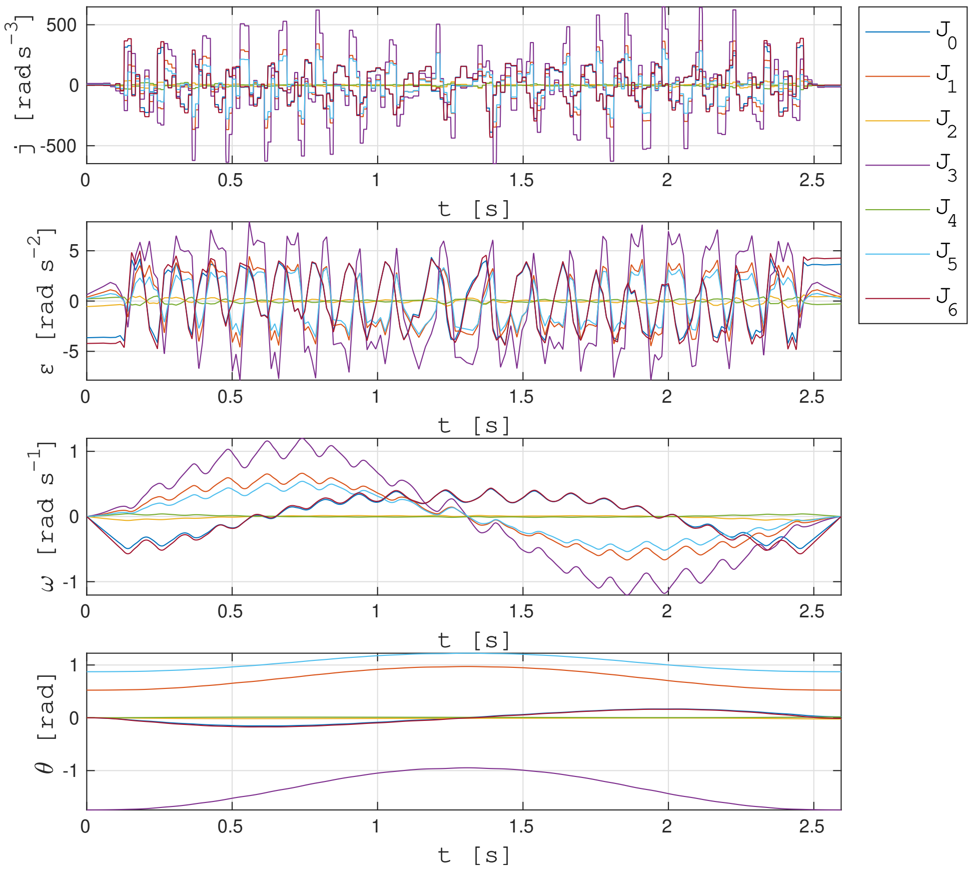


(b) : Visualization of the input waypoints, planned path and generated trajectory.

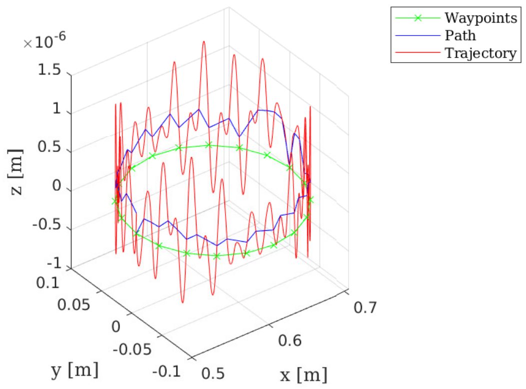


(c) : Path and trajectory error.

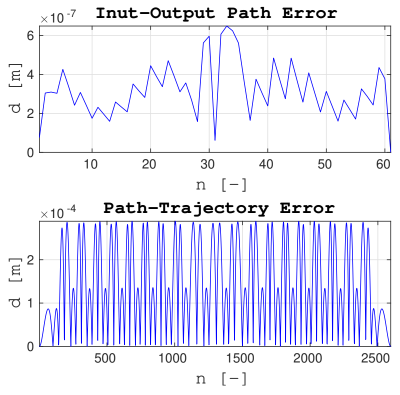
Figure B.5: Figures visualizing a rectangle shaped trajectory generated by the TOTG time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

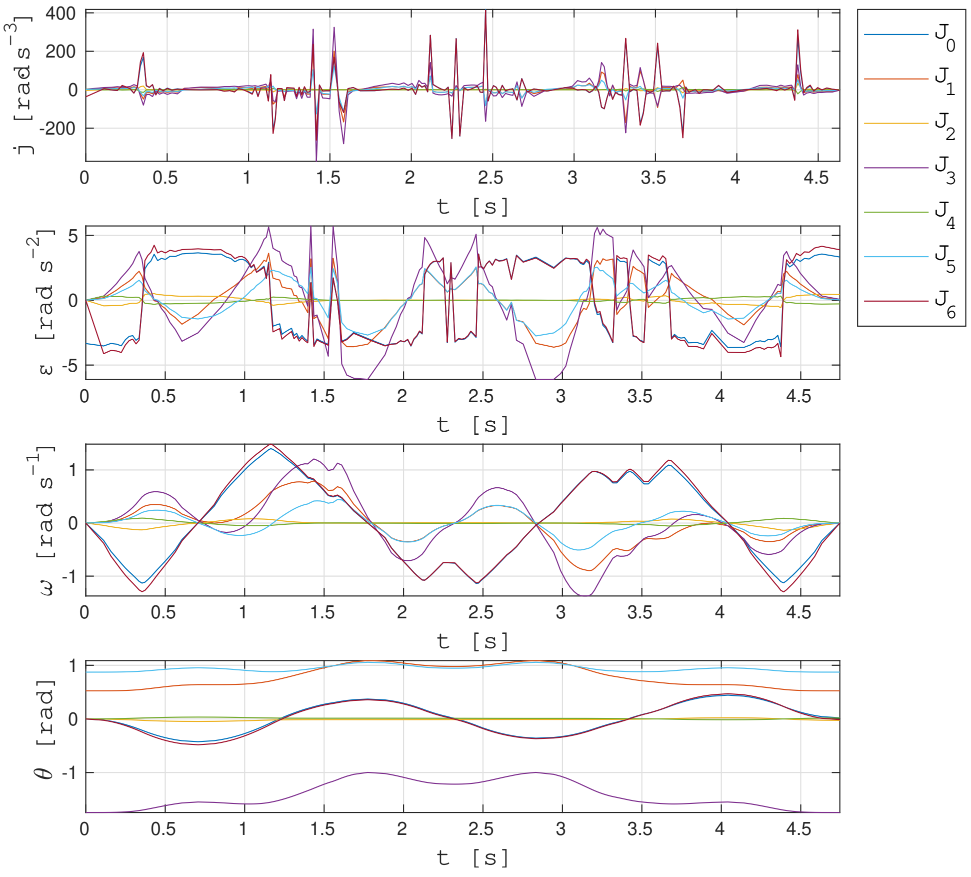


(b) : Visualization of the input waypoints, planned path and generated trajectory.

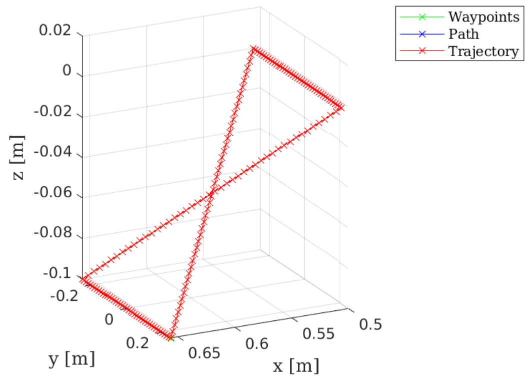


(c) : Path and trajectory error.

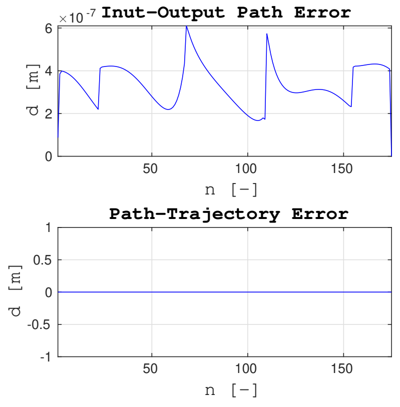
Figure B.6: Figures visualizing a rectangle shaped trajectory generated by the TOPP-RA time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

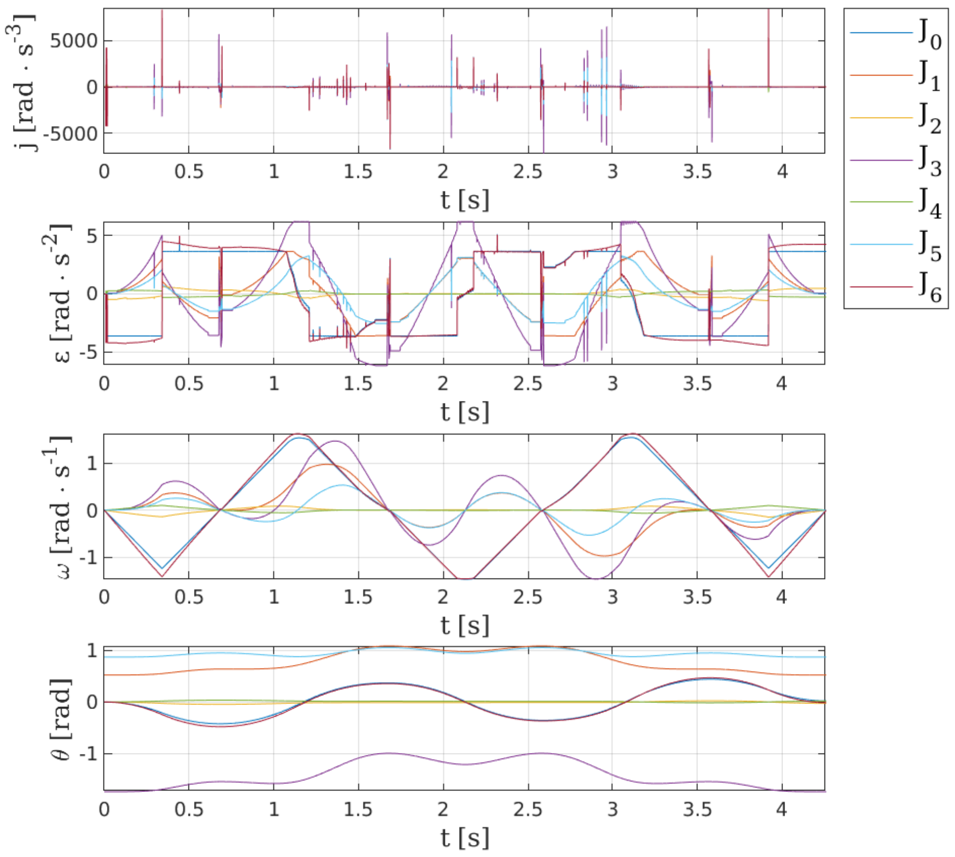


(b) : Visualization of the input waypoints, planned path and generated trajectory.

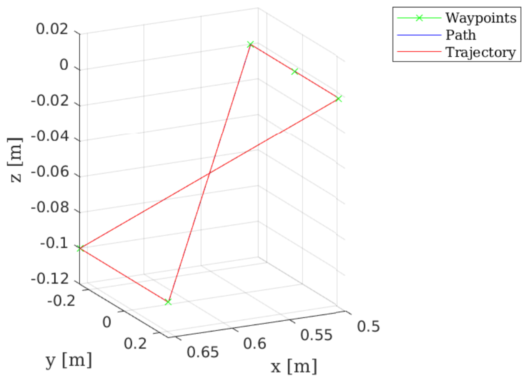


(c) : Path and trajectory error.

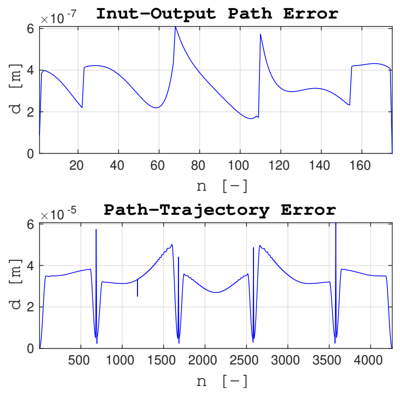
Figure B.7: Figures visualizing a rectangle shaped trajectory generated by the IPTP time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

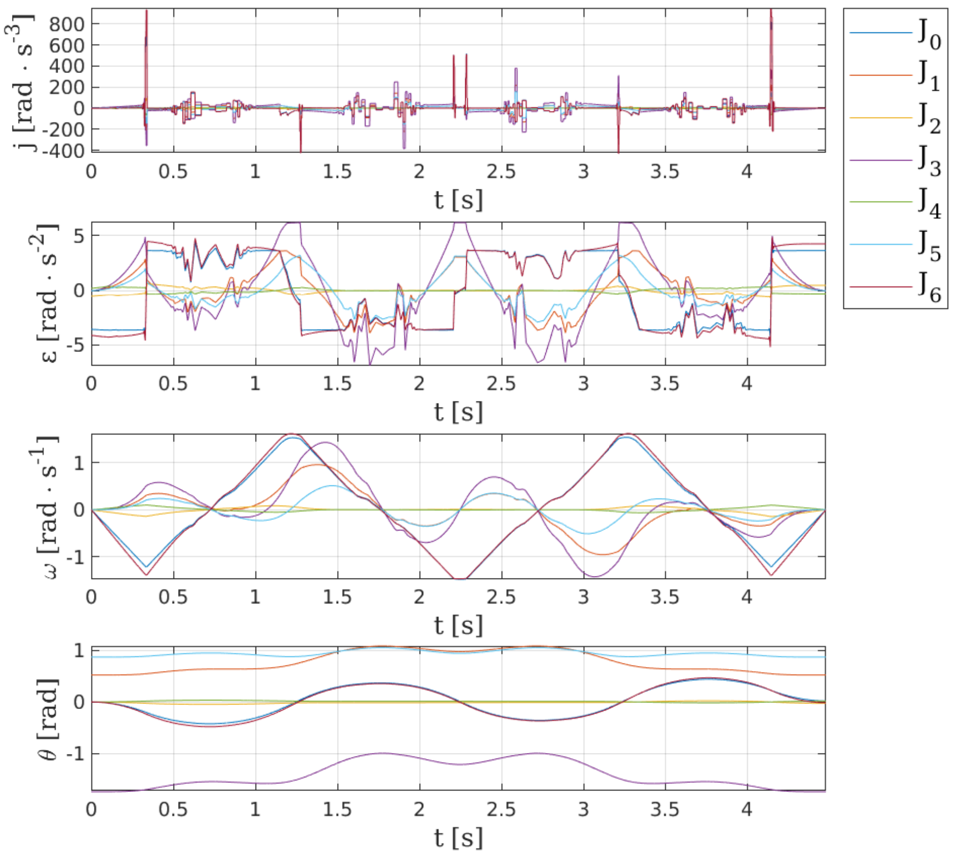


(b) : Visualization of the input waypoints, planned path and generated trajectory.

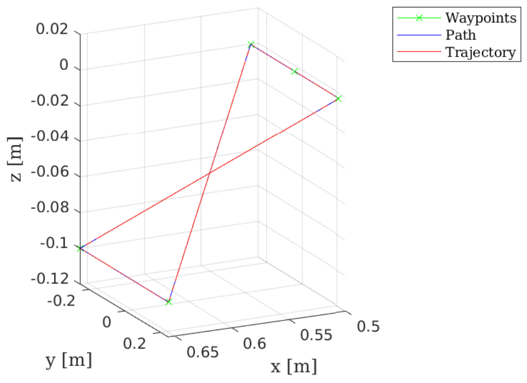


(c) : Path and trajectory error.

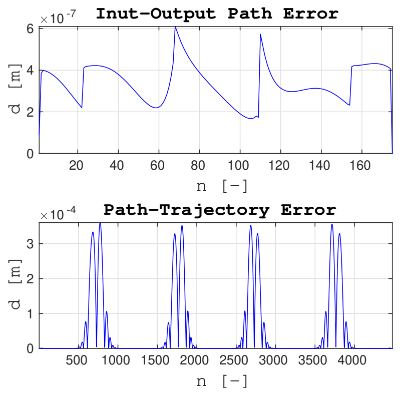
Figure B.8: Figures visualizing a rectangle shaped trajectory generated by by the TOTG time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

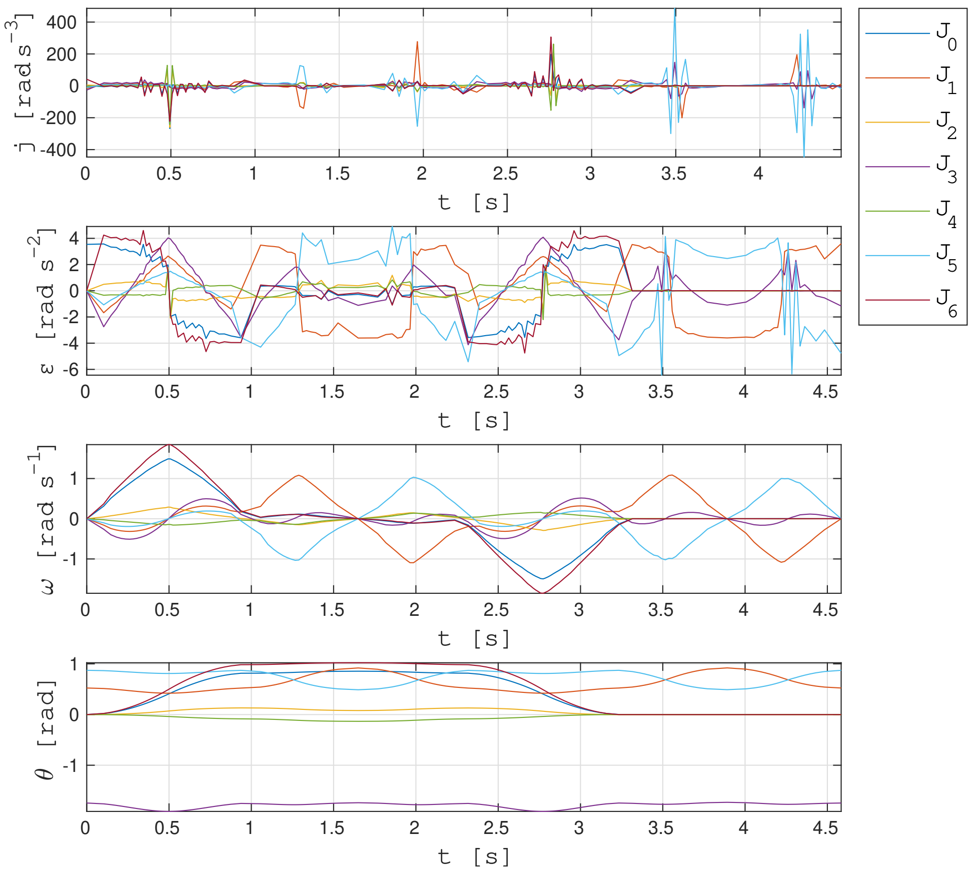


(b) : Visualization of the input waypoints, planned path and generated trajectory.

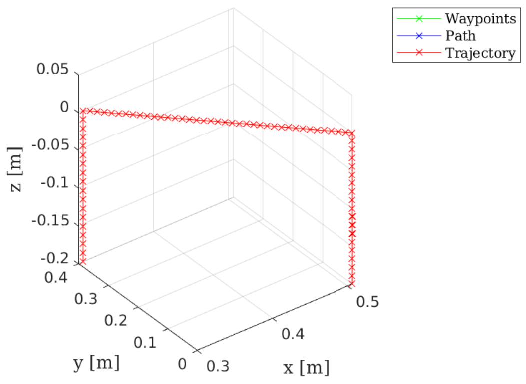


(c) : Path and trajectory error.

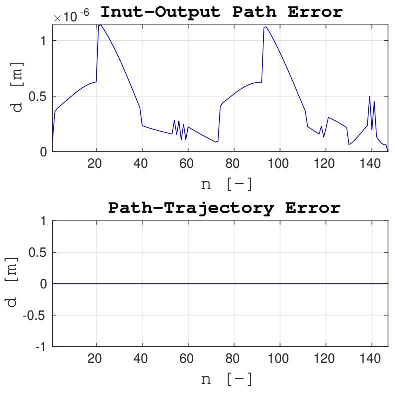
Figure B.9: Figures visualizing a rectangle shaped trajectory generated by the TOPP-RA time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

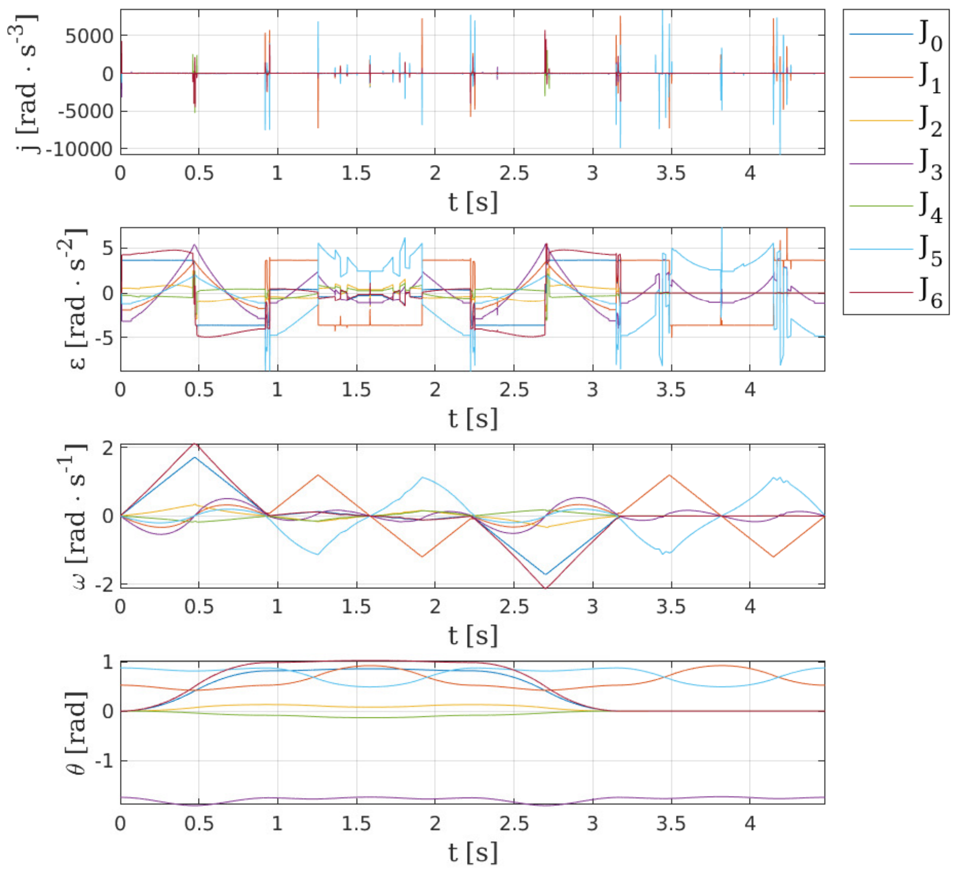


(b) : Visualization of the input waypoints, planned path and generated trajectory.

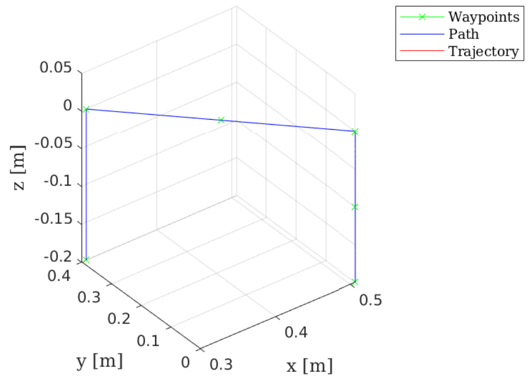


(c) : Path and trajectory error.

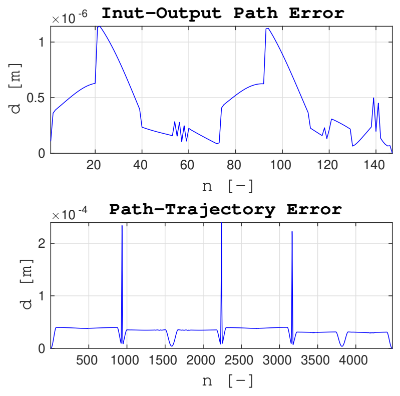
Figure B.10: Figures visualizing a rectangle shaped trajectory generated by the IPTP time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.

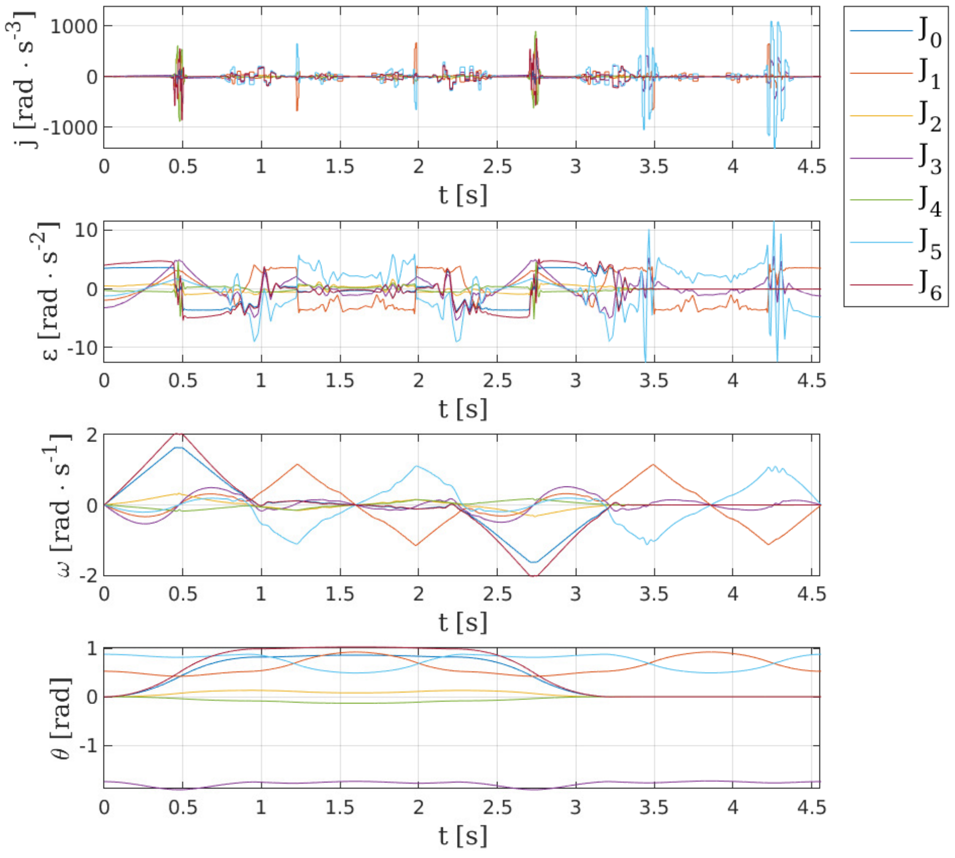


(b) : Visualization of the input waypoints, planned path and generated trajectory.

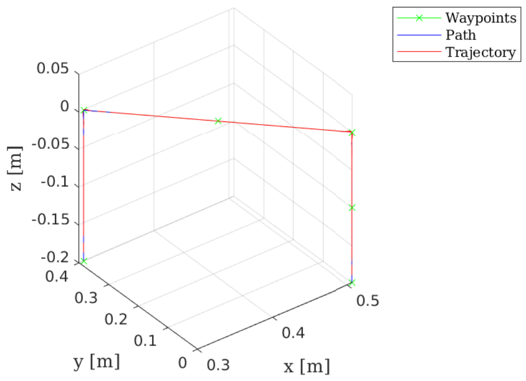


(c) : Path and trajectory error.

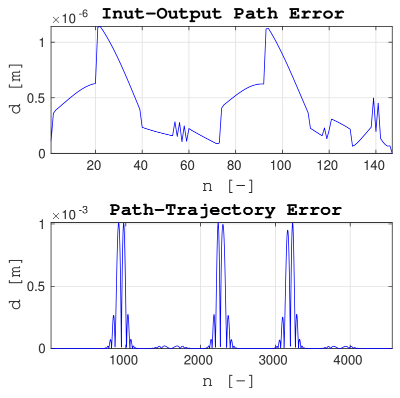
Figure B.11: Figures visualizing a rectangle shaped trajectory generated by the TOTG time parameterization algorithm. The path was planned by the OMPL from the input waypoints.



(a) : The courses of joint position θ, velocity ω, acceleration ε and jerk j for joints J_i, $i = 1, ..., 7$.



(b) : Visualization of the input waypoints, planned path and generated trajectory.



(c) : Path and trajectory error.

Figure B.12: Figures visualizing a rectangle shaped trajectory generated by the TOPP-RA time parameterization algorithm. The path was planned by the OMPL from the input waypoints.

Appendix C

Description of the paths for the time parameterization testing

Table C.1: Path points of the rectangular path.

	x [m]	y [m]	z [m]
$\mathbf{p_1}$	0.5064	6.1926e-08	0.3033
$\mathbf{p_2}$	0.5064	-0.2500	0.3033
$\mathbf{p_3}$	0.6564	-0.2500	0.3033
$\mathbf{p_4}$	0.6564	0.2500	0.3033
$\mathbf{p_5}$	0.5064	0.2500	0.3033
$\mathbf{p_6}$	0.5064	6.1926e-08	0.3033

Table C.2: Path points of the "sand clock" shaped path.

	x [m]	y [m]	z [m]
$\mathbf{p_1}$	0.5064	6.1926e-08	0.3033
$\mathbf{p_2}$	0.5064	-0.2500	0.3033
$\mathbf{p_3}$	0.6564	0.2500	0.2033
$\mathbf{p_4}$	0.6564	-0.2500	0.2033
$\mathbf{p_5}$	0.5064	0.2500	0.3033
$\mathbf{p_6}$	0.5064	6.1926e-08	0.3033

Table C.3: Path points of the "pick and place" shaped path.

	x [m]	y [m]	z [m]
$\mathbf{p_1}$	0.5064	6.1926e-08	0.3033
$\mathbf{p_2}$	0.4064	0.2000	0.3033
$\mathbf{p_3}$	0.3064	0.4000	0.3033
$\mathbf{p_4}$	0.3064	0.4000	0.1033
$\mathbf{p_5}$	0.3064	0.4000	0.3033
$\mathbf{p_6}$	0.4064	0.2000	0.3033
$\mathbf{p_7}$	0.5064	6.1926e-08	0.3033
$\mathbf{p_8}$	0.5064	6.1926e-08	0.2033
$\mathbf{p_9}$	0.5064	6.1926e-08	0.1033
$\mathbf{p_{10}}$	0.5064	6.1926e-08	0.2033
$\mathbf{p_{11}}$	0.5064	6.1926e-08	0.3033

Table C.4: Path points of the circle shaped path.

	x [m]	y [m]	z [m]
$\mathbf{p_1}$	0.5064	6.1926e-08	0.3033
$\mathbf{p_2}$	0.5113	-0.0309	0.3033
$\mathbf{p_3}$	0.5255	-0.0588	0.3033
$\mathbf{p_4}$	0.5476	-0.0809	0.3033
$\mathbf{p_5}$	0.5755	-0.0951	0.3033
$\mathbf{p_6}$	0.6064	-0.100	0.3033
$\mathbf{p_7}$	0.6373	-0.0951	0.3033
$\mathbf{p_8}$	0.6652	-0.0809	0.3033
$\mathbf{p_9}$	0.6873	-0.0588	0.3033
$\mathbf{p_{10}}$	0.7015	-0.0309	0.3033
$\mathbf{p_{11}}$	0.7064	6.1926e-08	0.3033
$\mathbf{p_{12}}$	0.7015	0.0309	0.3033
$\mathbf{p_{13}}$	0.6873	0.0588	0.3033
$\mathbf{p_{14}}$	0.6652	0.0809	0.3033
$\mathbf{p_{15}}$	0.6373	0.0951	0.3033
$\mathbf{p_{16}}$	0.6064	0.1000	0.3033
$\mathbf{p_{17}}$	0.5755	0.0951	0.3033
$\mathbf{p_{18}}$	0.5476	0.0809	0.3033
$\mathbf{p_{19}}$	0.5255	0.0588	0.3033
$\mathbf{p_{20}}$	0.5113	0.0309	0.3033
$\mathbf{p_{21}}$	0.5064	6.1926e-08	0.3033