Karlsruhe Institute of Technology

Dynamic Online Trajectory Generation

Acceleration Capabilities Considered for Real-Time Path Planning

Diploma Thesis
of

Robert Katzschmann

At the Department of Informatics, Karlsruhe Institute of Technology
Institute for Anthropomatics (IFA)
High Performance Humanoid Technologies Lab (H^2T)

and

at the Department of Computer Science, Stanford University
Artificial Intelligence Laboratory
Robotics Group

Reviewer:	**Prof. Georg Bretthauer**
Second reviewer:	**Prof. Oussama Khatib**
Third reviewer:	**Prof. Tamim Asfour**
Advisor:	**Dr. Torsten Kröger**
Second advisor:	**Prof. Tamim Asfour**

Duration: August 1st, 2012 – January 31th, 2013

Declaration:

I hereby declare that I completed the work at hand independently and only used the specified tools and resources.
Palo Alto, January 31th, 2013

Abstract

A concept of *online trajectory generation* for robot motion control systems enabling instantaneous reactions to unforeseen sensor events was introduced in former publications. This thesis extends the existing concept by allowing time-variant kinematic motion constraints being applied online to the algorithms, so that low-level trajectory parameters can now be changed abruptly, and the system can react instantaneously within the same control cycle of typically two milliseconds or less. The formerly proposed class of algorithms does not take into account dynamically changing acceleration capabilities for given kinematic and dynamic models of robot systems. This leads to the problem that the values of the motion constraints used for the online trajectory generation algorithms have to be chosen constant in its value and relatively low compared to the actual available acceleration capabilities of the robot. This assures on the one hand that the generated motion trajectory can be performed all the way through with, if at all, negligible tracking-errors. And on the other hand, this leads to a suboptimal reactiveness of the system, since it could potentially outperform more when accelerating and decelerating. This thesis extends the algorithms of the previous approach *by taking into consideration the whole system dynamics* when generating trajectories online. The extension considers the acceleration capabilities of a robot *by looking ahead in time along its future motion path*, thus allowing the generation of adaptive trajectory profiles during the motion of the robot. Real-world experimental results using a lightweight robot arm highlight the practical relevance of this extension.

Zusammenfassung

In früheren Veröffentlichungen wurde ein Konzept der *Online-Trajektorien-Generierung* für Bewegungskontrollsysteme von Robotern eingeführt, welches sofortige Reaktionen auf unvorhergesehene Sensorereignisse ermöglicht. Diese Diplomarbeit erweitert das existierende algorithmische Konzept, indem in Echtzeit zeitvariante kinematische Bewegungseinschränkungen auf die Algorithmen angewandt werden. Low-level Trajektorienparameter können dadurch abrupt geändert werden, denn das System reagiert unverzögert innerhalb des gleichen Steuerungszyklus von maximal zwei Millisekunden. Die in den früheren Veröffentlichungen vorgeschlagene Klasse von Algorithmen berücksichtigt nicht die dynamisch ändernden Beschleunigungsfähigkeiten trotz existierender kinematischer und dynamischer Modelle der Robotersysteme. Dies führt zu der Problemstellung, dass die Werte, welche für die Bewegungeinschränkungen der Echtzeit-Trajektoriengenerierung verwendet werden, konstant und relativ niedrig im Vergleich zu den tatsächlich verfügbaren Beschleunigungsfähigkeiten des Roboters ausgewählt werden müssen. Dies gewährleistet einerseits, dass die erzeugten Bewegungstrajektorien sich vollständig mit vernachlässigbarem Tracking-Fehler durchführen lassen. Andererseits führt dies zu einer suboptimalen Reaktionsfähigkeit des Systems, da beim Beschleunigen und Abbremsen erhebliche Leistungsreserven nicht genutzt werden. Die vorliegende Arbeit erweitert die bisherigen Algorithmen *durch Berücksichtigung der gesamten Systemdynamik* bei der Erzeugung von Trajektorien in Echtzeit. Die Erweiterung betrachtet die Beschleunigungsfähigkeiten eines Roboters *zeitlich vorausschauend entlang seines zukünftigen Bewegungspfades* und erlaubt dadurch die adaptive Generierung von Trajektorienprofilen während dem Verfahren des Roboters.
Der generelle Aufbau dieser Arbeit ist in Abb. 1.9 wiedergegeben. Kapitel 2 gibt einen Überblick über verwandte Arbeiten und führt in die erforderlichen Grundlagen ein. In Kapitel 3 wird das Konzept der Beschleunigungsfähigkeit in Form von *Parallelotopen* beschrieben und eine Verwendungsmöglichkeit bei der Bahnplanung vorgestellt. Kapitel 4 stellt ein neues Konzept zur besseren Beschreibung der Beschleunigungsfähigkeit vor und wie dies optimiert für Online-Trajektorien-Generierung verwendet werden kann. Kapitel 5 vereint auf verschiedene Weisen das neue Konzept aus Kapitel 4 mit dem Online-Trajektorien-Generator aus vorigen Arbeiten zu dem sogenannten *Dynamischen Online-Trajektorien-Generator*. Das Kapitel 6 beschreibt die Implementierung der neuen Algorithmen und stellt deren praktische Relevanz anhand von experimentellen Ergebnissen an einem Roboter-Leichtbauarm dar. Eine Diskussion über die neuen Algorithmen und alternative Ansätze wird in Kap. 7 geführt. Eine abschließendes Fazit und Ausblick ist in Kapitel 8 zu finden.

Acknowledgements

I would like to express my strong gratitude to Professor Oussama Khatib and Professor Tamim Asfour for their continuous advice and assistance throughout this research, without their help this research would not have been possible. I would also like to express my deep appreciation to Dr. Torsten Kröger for his patient guidance, strong encouragement and useful critiques as my research advisor. I wish to acknowledge the permanent support provided by my lab colleagues Samir Menon and Francois Conti, the many fruitful discussions we had are highly appreciated. My grateful thanks are also extended to all members of the Artificial Intelligence Laboratory at Stanford University for their constant support and for providing me with all the resources needed. I would like to thank the Dr.-Ing.-Willy-Höfler-Stiftung, Förderverein Kurt Fordan für herausragende Begabungen e.V., and the Friedrich-Naumann-Stiftung für die Freiheit for the research funding provided. Finally, I wish to thank my family for their support and encouragement throughout my research.

Contents

List of Figures

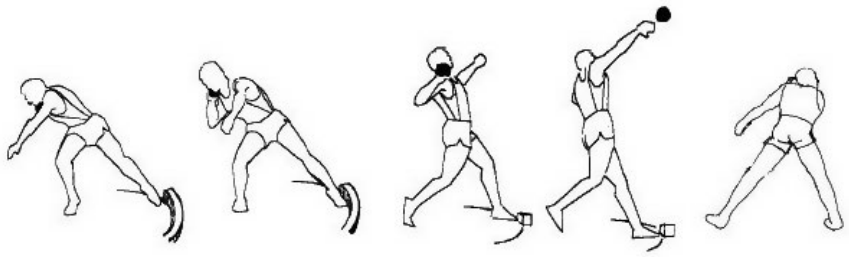
1. Introduction

The introduction begins with a general motivation for the relevance of the research task dealt with in this thesis. Following the motivation, the actual problem is described in detail. Based on this problem description, an outline of the thesis work is given.

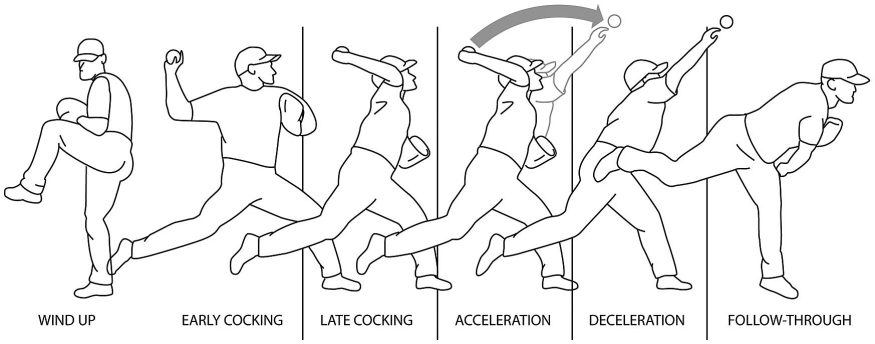
1.1. Motivation

The analysis of dynamic capabilities is an important aspect when optimizing robot designs for specific tasks. Modeling these capabilities correctly for the use with online trajectory generation is an important task when working with highly dynamic robotic systems. The mapping of actuation forces/torques to the capability to accelerate and decelerate is a solved problem, but it has not been done yet in connection with the generation of motion trajectories in real-time.

When a human throws a ball in a straight manner, he uses less acceleration capabilities compared to a more natural throwing movement above the head. We consider for example a straight throwing movement of a shot-putter compared to the more natural, over-the-head throwing movement of a baseball player. This is visualized in Fig. 1.1. Both are professionals in their discipline und certainly had a lot of training in order to develop an optimized throwing trajectory, nevertheless the baseball player is using more of his acceleration capabilities compared to the shot-putter. Why is this?



(a) Straight Throwing Movement [27].



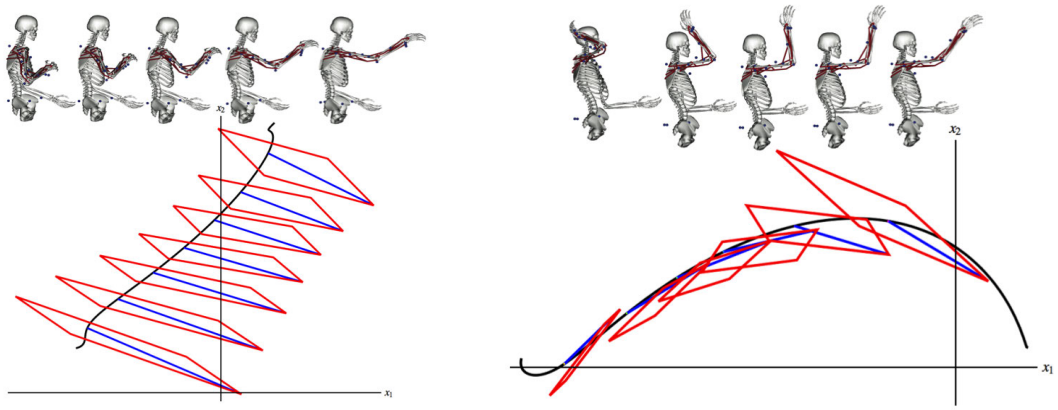
(b) Natural Throwing Movement [33].

Figure 1.1.: Throwing movement of a shot-putter and a baseball player.

The shot-putter is dealing with a relatively heavy ball. This is why his main concern is to perform a short motion as close as possible to his body in order to not drop the ball while setting it off with a strong impulse.

Athletes know that a motion straight in its path and close to the body has proven itself through constant trial-and-error as most optimal for throwing a heavy object. Nevertheless, the motion is not along the path of highest acceleration capabilities, a human could perform better when throwing a ball. A simplified skeleton model of the shot-putter's throwing movement without the added payload of the ball is shown in Subfig. 1.2a. The black line shows the path the hand follows during a straight throwing motion. The red parallelograms indicate for several poses along the motion the extremal acceleration capabilities. As further away the red line is from the point where the blue line and the black line intersect, as higher is the acceleration capability in that direction. The straight blue line highlights the direction with the highest acceleration capability. For the shot-putter's movement, the blue line is almost constantly perpendicular to the black line of the taken path, which means that the motion is not at all moving along a path with high acceleration capabilities.

In comparison to the case of the shot-putter, Subfig. 1.2b models and visualizes a natural throwing motion of a baseball player. As the black line shows, the path of the over-the-head throwing motion is considerably strong curved, leading from behind the body, over the head to the front of the body. The blue line indicates the direction of highest acceleration capability. Compared to the shot-putter's case, the baseball player's motion path is much more aligned to the blue lines of highest acceleration capability. The baseball player is able to use much more of his acceleration capabilities when throwing a ball. While the shot-putter has to prioritize the static stability of the human arm holding the heavy ball, the baseball player can perform a highly dynamic motion which is not just along a straight line from start to end pose.



(a) Straight Throwing. (b) Natural Throwing.

Figure 1.2.: Acceleration Capabilities for the hand frame: The black line describes the path taken, the red parallelogram describes maximum acceleration capabilities in all directions, the blue line shows direction of highest acceleration capability. The skeletons in the pictures are taken from [19].

The optimized human motion examples from above show us that highly dynamic robot motions require the analysis and modeling of the system's acceleration capabilities. Only if we know the acceleration capabilities of a robot manipulator in its various configurations, we can optimize the acceleration trajectory and therefore the motion path taken from start to target. The term trajectory describes the path a moving object is taking under the action of given forces. Hereinafter, we will use the word trajectory to not only describe a position progression or path, but also its time derivations, for example an "acceleration trajectory".

Motion trajectories are generated/planned in order to achieve smooth motions from a start motion state to a target motion state, see Fig. 1.3. It is not feasible to accelerate and decelerate a robot in "bang-bang" control mode, that is to drive the motor currents instantly to the highest or lowest possible level in order to accelerate or decelerate as much as possible. The motor currents cannot be switched on and off, but need to be increased/decreased at a suitable rate in order to avoid damage of the drivetrain components through premature aging. Furthermore, it decreases motion tracking errors, which means that the desired and current robot motion states are more likely to be the same at every cycle. This limitation in increase and decrease of motor currents can be kinematically described as limiting the jerk. The jerk is the derivation of the acceleration. It is therefore necessary that trajectory

generation takes motion constraints into consideration so that sudden motor current changes, accelerations past the robot's capabilities and too high joint velocities are avoided.

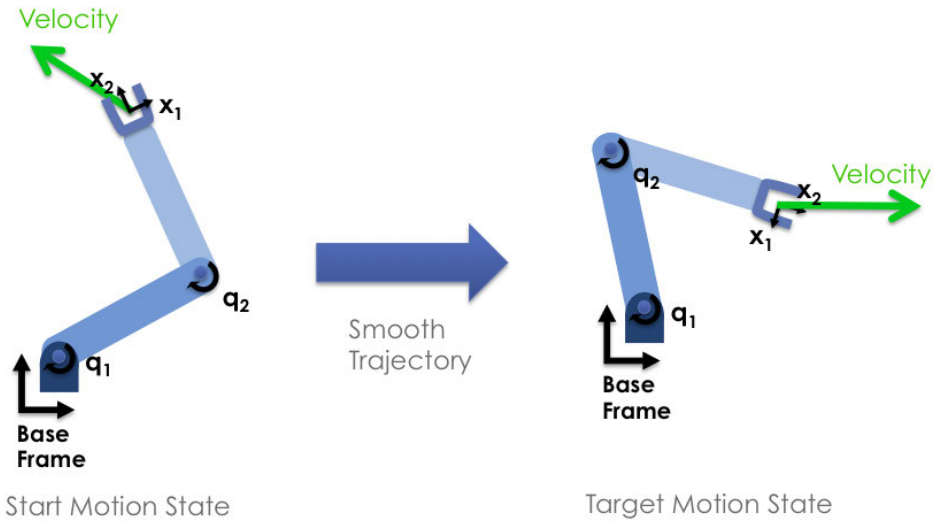


Figure 1.3.: Reason for trajectory planning.

Trajectory Generation in real-time/online is crucial when working with sensor-integrated robot arm controllers. The integration of sensors in the feedback loops of low-level motion controllers require the ability to switch instantaneously from sensor-guided control (e.g. force/torque control [39] or visual servo control [10]) to trajectory following control (and the other way around) at unanticipated times. A simplified flow-chart of such a robot arm controller is given in Fig. 1.4. The figure visualizes how a robot is actuated through a low-level PID controller which in return receives its commands from a sensor-based controller or a trajectory generator.

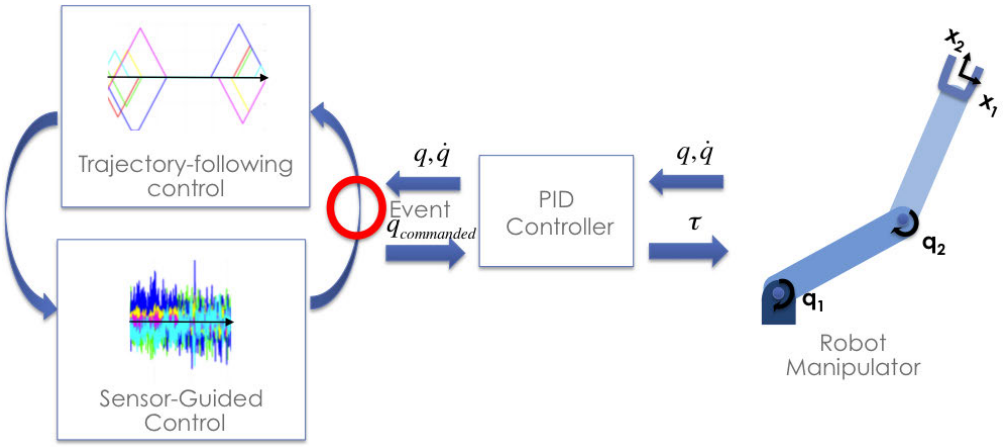


Figure 1.4.: Simplified robot arm control scheme.

This way, new event-based robot programming methodologies can be realized as robots become enabled to react instantaneously in the moment the event is detected. In recent works of Kröger[24, 21, 23], a concept of online trajectory generation considering motion constraints has been proposed. The resulting algorithms run in parallel to low-level motion controllers and are able to compute a full trajectory within the same control cycle that the unforeseen switching occurs. The transition of sensor-guided control to trajectory-following control is shown in Fig. 1.5. The dashed green rectangle shows a motion controlled by sensor measurements, the red line

indicates a sensor event, which causes a switch to the trajectory-following control mode, shown by the orange rectangle.

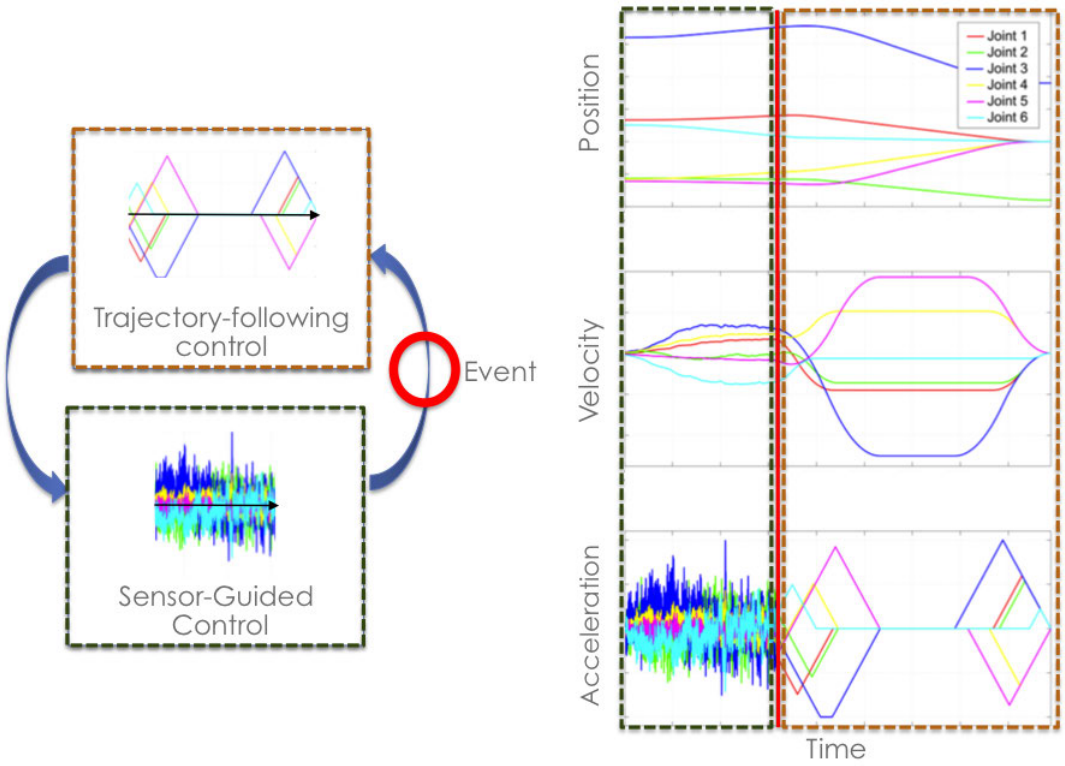


Figure 1.5.: Trajectory transition between sensor-guided and trajectory-following control [24].

The major limitation of the current trajectory generator is that only constant motion constraints can be used as inputs for this algorithm. The algorithms do not allow for the consideration of steadily changing acceleration capabilities. The goal of this thesis work is to deal with this short-coming by enhancing an existing trajectory generator so that it can take into account the acceleration capabilities of the robot manipulator. If an enhancement of the OTG does not work out, the goal would then be a new concept for a trajectory generator able to consider the capabilities.

1.2. Problem Description

Assuming constant motion constraints as input for online trajectory generation, a time-optimal solution can already be generated at any time and at any given start and end state. According to Kröger[24, 21, 23], this is the “kinematic time-optimal case”. The implemented solution for this is the so-called “Online Trajectory Generator (OTG)”.

The problem to be solved in this thesis is to instantly generate a motion trajectory from a given motion state to a desired target motion state within the shortest possible time under consideration of the whole system dynamics. A motion state describes for a specific instant in time the values for position, velocity, acceleration (and any further time derivations from interest). The system dynamics are considered in two steps: calculation of available acceleration capabilities based on actuator forces/torques and then mapping these capabilities to useful acceleration constraints for the trajectory generation. According to Kröger, this is the “dynamic time-optimal case”, more complicated than the “kinematic time-optimal case”.

1.2.1. Reasons for Varying Motion Constraints

There are various reasons why the assumption of constant motion constraints for trajectory generation are in general not valid:

1. The velocity capabilities $\dot{\boldsymbol{q}}_{min/max}$ are in general not constant, but depend on the position of the actuated joints. Since this effect is hard to model and actually comparably small, we can assume that the velocity capabilities are constant for each individual axis:

$$\dot{\boldsymbol{q}}_{max} = -\dot{\boldsymbol{q}}_{min} = const. \text{ for } \forall \boldsymbol{q} \tag{1.1}$$

   The values are either directly provided by the manufacturer or can be calculated based on the robot drive train specifications or can be estimated by experiments with the robot. Setting the velocity capabilities constant allows for using them one-to-one as velocity constraints for the trajectory generation.

2. The jerk capabilities $\dddot{\boldsymbol{q}}_{min/max}$ depend on the magnetization time constants of the motor irons, to be found in the motor specifications. Therefore, there is a minor dependency between the current motion state and how much jerk can be applied. Modeling and calculating this dependency is non-trivial. An empirical approach is used to find an appropriate value for a constant and symmetric jerk capability for every individual axis:

$$\dddot{\boldsymbol{q}}_{max} = -\dddot{\boldsymbol{q}}_{min} = const. \text{ for } \forall \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}} \tag{1.2}$$

   Modeling the capabilities as constant allows them to be used one-to-one as jerk constraints for the trajectory generation.

3. The acceleration capabilities $\ddot{\boldsymbol{q}}_{min/max}$ are not constant, but depend on the whole system dynamics. They are a function of the position, velocity and extremal motor forces/torques:

$$\ddot{\boldsymbol{q}}_{min/max} = f\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}_{min/max}(\dot{\boldsymbol{q}})\right) \tag{1.3}$$

   The change of acceleration capabilities can not be neglected in its effect. Using parameter identification methods, it is possible to find for a robotic system an appropriate model to describe the acceleration capabilities. A further description of the dynamic equations is given in Sec. 3.1.

   The problem of mapping the actual acceleration capabilities to acceleration constraints useful for the online trajectory generation is described in Sec. 1.2.2.

Among all the motion constraints, we only consider the acceleration constraints as variables. The acceleration constraints are derived from the acceleration capabilities $\ddot{\boldsymbol{q}}_{min/max}$. The other constraints are taken one-to-one from the according capabilities, which are considered constant and symmetric in its values.

1.2.2. Mapping Acceleration Capabilities to Acceleration Constraints

The acceleration constraints used for the trajectory generation have to be linearly independent between each single axis. The actual acceleration capabilities calculated with the dynamics equation are generally not linearly independent in its boundaries, but have depending boundaries. In 2D, this can be visualized with a shape like a parallelogram, see Fig. 1.6. The red parallelogram describes the extremal acceleration capabilities. There are infinite options to map the acceleration capabilities back to acceleration constraints suitable for online trajectory generation. In the figure, three possible options are shown, many more options are possible. A suitable approach to do this mapping is part of the problem dealt with in this thesis work.

1.2.3. Future Motion Constraints not Considered

The OTG, which optimizes the execution time for the kinematic time-optimal case, neglects the whole system dynamics. The OTG assumes, that the available acceleration capability remains constant for the whole motion.

A short insight into the setup of the OTG explains this problem. The OTG algorithm is called in every cycle of the motion control loop to generate the motion state of the next discrete time instant. The Fig. 1.7 visualizes this

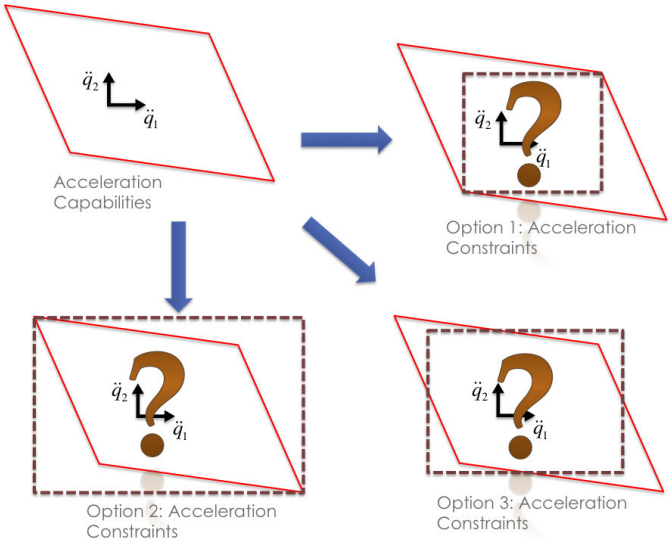


Figure 1.6.: Acceleration capabilities to constraints.

concept. At the current time instant t_i, the input of the algorithm is shown on the left side of the rectangle: the current motion state, the desired target motion state and the motion constraints. The algorithm then calculates internally the whole motion trajectory in form of continuous piecewise polynomials from the current motion state to the target motion state. Based on this set of piecewise polynomials, the OTG returns the motion state for the next time step after t_{cycle} has elapsed. This returned value is the output motion state at the time t_{i+1}. This output motion state is reused as the "new" current motion state in the next time step.

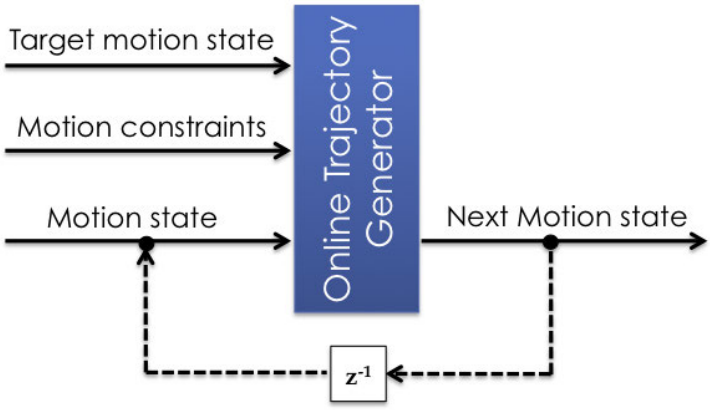


Figure 1.7.: OTG flowchart according to [24].

As a consequence, the OTG has to be called again at every time step. If the input motion state is the same as the output motion state of the last time step and if the motion constraints and target motion state are still the same as they were the step before, the OTG does not do a new planning of all the piecewise polynomials, but simply looks up the next motion state based on the previously generated set of polynomials. This set was either generated in the time step before or even many steps before that. As long as the motion state has moved forward as predicted and the motion constraints and desired target motion are still the same, there is no need to re-run the internal computation.

As soon as one of the input values changes by more than just a small numerical error, the OTG generates

a new set of piecewise polynomials and therefore a new trajectory from the current motion state to the target motion state. Since the value change of the acceleration constraints is comparably large, see Sec. 1.2.1, the OTG would recalculate its trajectory at every time step. As long as the acceleration constraints stay at the same level or above, this would work, because the trajectory generator would have at every time step compared to the step before even more acceleration potential available. But if the acceleration constraints are lowered, it can happen that the planner can not avoid anymore a position overshooting. To plan new at every step is the same as if we would do no planning at all. This problem is visualized in Fig. 1.8.

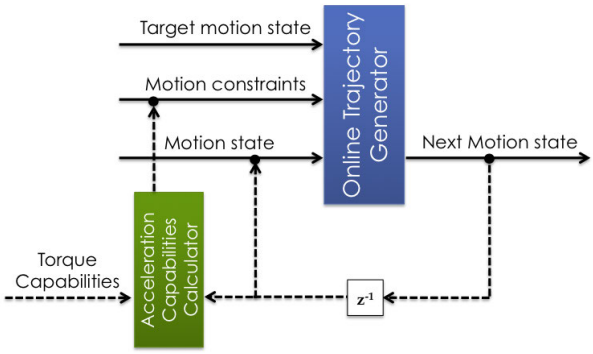


Figure 1.8.: Every step new planning is no planning.

To give an example that illustrates this situation: When you draw up a company's investment plan for the next year, you base this plan on a budget that is available for that upcoming year. You plan with that budget in such a way that you fully use all the resources to achieve the best and fastest growth. If your investors now constantly decrease the available budget in the next weeks and ask you to redo the planning, the previously set goals won't be achievable anymore in the same time frame. Even worse, it might even take longer than if you would have used right from the beginning the lower budget level for the planning.

1.2.4. Current Workaround to the Problem

The current workaround to this planing problem is to choose the motion constraints at a comparably low and constant level for the complete motion. The acceleration capabilities of a robot are highly fluctuating, so that the chosen constraint level has to be at the minimum of this fluctuation to ensure, that the motion trajectory can be performed all the way through with, if at all, negligible tracking-errors. Only a small portion of the available acceleration capability will therefore be used. This consequently leads to a suboptimal reactiveness of the system, since it could potentially accelerate much more with an optimized trajectory generator.

The current OTG implementation is called "Type IV" and only allows the target acceleration to be zero. The unimplemented "Type V" would also allow a target acceleration unequal to zero.

1.3. Thesis Outline

There are various aspects that need to be considered when we try to solve the problem of combining online trajectory generation with robot acceleration capabilities. The algorithm solving this problem will be called "Dynamic Online Trajectory Generator", short DOTG. An overview of the DOTG development process is given in Fig. 1.9. The outline of this thesis work can be structured according to this development process.

Chapter 2 gives an overview about related work dealing with this problem and introduces the basics needed to approach the problem. Chapter 3 describes the concept of acceleration capabilities as parallelotopes and offers first ideas on how to qualify them for path planning. Chapter 4 introduces the new concept of splitting accelerations up into normal and tangential components for a better description of acceleration capabilities. In that same chapter is also explained how the acceleration capabilities can be used to determine the current extremal accelerations constraints and how to map and merge the identified constraints in such a way that they become

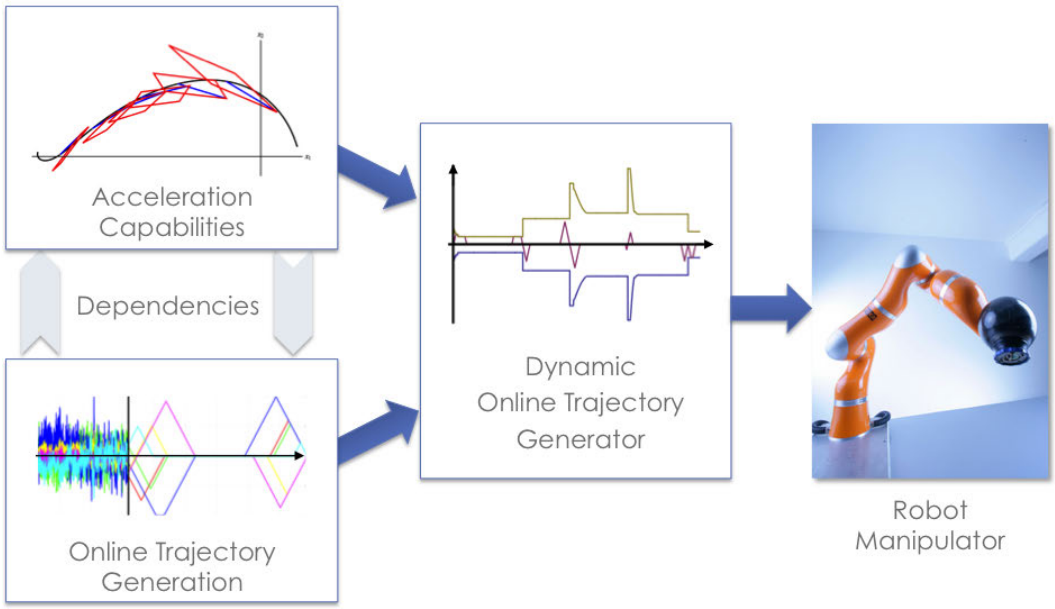


Figure 1.9.: Dynamic OTG development overview.

useful as input to the OTG algorithm. Chapter 5 combines in several ways the method of Ch. 4 with the OTG algorithm to the so-called DOTG. The following Ch. 6 gives details on the implementation and experimental results of the DOTG algorithm on a KUKA light weight robot. A discussion about the parts of the DOTG and further alternative approaches are presented in Ch. 7. A final conclusion and outlook is then given in Ch. 8.

2. Related work and Basics

This chapter first gives an overview over related work and then goes into more detail with the basics on robot dynamics and online trajectory generation.

2.1. Related Work

In this section work related to real-time trajectory generation under consideration of whole system dynamics is given. An ideal real-time capable dynamic trajectory generator would work for multiple DOFs and would allow to define motion constraints for velocities, accelerations and jerks. The start and target velocities and accelerations would not necessarily need to be zero, but any arbitrary value inside the given constraints. Unfortunately, such an ideal trajectory generator does not exist yet, at least none that is proven to be dynamic time-optimal. Nevertheless, there are several works on kinematic time-optimal trajectory generators and off-line dynamic time-optimal trajectory generators that could serve as a basis for the development of an ideal generator.

2.1.1. Online Trajectory Generation

The works most related to field of multiple DOF online trajectory generation for the kinematic time-optimal case are [8, 28, 16, 15, 24, 21, 23], all of which belong to the fields of robot motion control [40] and trajectory generation [1, 18] in robotic systems.

In Biagiotti [2], an overview on trajectory generating algorithms is given. For multiple DOFs, methods described like *linear Trajectory with polynomial blends*, *B-spline Methods (incl. Nurbs)*, *Bezier Curves* and *Piecewise Polynoms* are described. All of these algorithms are kinematic-time optimal, that is none of them take into account the whole system dynamics.

Broquère et al. [8] builds on-line cubic trajectories for an arbitrary number of independently acting DOFs. This approach and also the approach of Liu [28] are based on the classic concept of a seven-segment acceleration profile by Castain [9]. With regard to Kröger [21], it is a Type V on-line trajectory generation approach designed for handling several DOFs individually without time or phase synchronization. A disadvantage of Liu [28] is that it does not allow for initial acceleration values unequal to zero.

Instead of generating motion trajectories, a concept proposed by Haddadin [15] uses virtual springs and damping elements as input values for a Cartesian impedance controller. A further, recent work of Haschke et al. [16] presents an on-line trajectory planner in the very same sense as Kröger [24] does. The proposed algorithm generates jerk-limited trajectories from arbitrary states of motion, but it suffers from numerical stability problems, that is, it may happen, that no jerk-limited trajectory can be calculated. In such a case, a second-order trajectory with infinite jerks is calculated. Furthermore, the algorithm only allows target velocities of zero.

The most promising approach for online trajectory generation has been introduced in the works of Kröger [24, 21, 23]. It generates kinematic-time optimal trajectories for multiple DOFs. The trajectory generator time or phase synchronizes the multiple DOFs. The major limitation is, that only kinematically time-optimal trajectories are generated since only constant kinematic motion constraints are considered. This is already sufficient for many fields of application, in particular those fields, in which only relatively low robot velocities are exerted. But it is of course a disadvantage of major importance for applications that require high-performance robot motions to utilize trajectories that also consider the system dynamics.

2.1.2. Offline Trajectory Generation

We can find a large number of off-line trajectory generation concepts that make use of a dynamic system model. Most approaches have their roots in the concepts of Bobrow, Pfeiffer, Hollerbach and Shin [7, 6, 29, 17, 34, 35].

Pfeiffer [29] gives an off-line path approach by transforming the equations of motion to one DOF. This results in a set of equations that constrain the motion by path geometry and joint torques. The time optimal solution is found by a sequence of "bang-bang" like acceleration/ deceleration extremes without any jerk limitation.

The "Dynamic Scaling" concept introduced by Hollerbach [17] and further elaborated in Biagiotti [3] allows for the consideration of whole system dynamics by using the concept of time scaling. It is done by calculating for a given trajectory the complete dynamics to get the required torques and then scale this torque profile so that the extremal torque capabilities are never exceeded. Even though a dynamics recalculation is not needed, this off-line approach requires a suitable trajectory generator to provide an initial guess. The Dynamic Scaling concept was further extended through Sahar/ Hollerbach [31] by a graph search on a tessellated joint space.

An off-line numerical approach has been shown in Wu [43], which assumes a given path and also limits the jerk. The method does not allow start and target velocities and accelerations to be unequal to zero.

The offline approach by Bobrow [7, 6] assumes a given path and then finds a dynamic time-optimal solution without jerk limitations by allowing steps in the acceleration profile. Even though the full concept is limited, the paper's idea to split the acceleration into its path components is reused for this thesis work. The splitting of the acceleration into a normal an tangential component combined with the online trajectory generator of Kröger [24, 21, 23].

2.2. Robot Dynamics

As with trajectory generation, the description of rigid body system dynamics belongs to the very basic levels in the field of robotics. The following joint and operational space dynamic equations are the basis for the further derivation of the acceleration capabilities in Ch. 3 and the path dynamics concept in Ch. 4.

The work of Featherstone [11] was one milestone in the early 1980s. General overviews are given in his later publications [12] and [13]. In the following, I do not consider the dynamics of a concrete robotic system, but suggest a generic approach that can be applied to a number of robotic systems. $\boldsymbol{\tau}$ stands interchangeable for actuator forces or actuator torques. Thereupon only the term torques will be used.

2.2.1. Joint Space

The standard form of the dynamic equations of motion in joint space is given in Eqn. (2.37) of [13] with

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{\tau}_g(\boldsymbol{q}) = \boldsymbol{\tau} \tag{2.1}$$

The variables are defined as follows

$\boldsymbol{q}$	joint positions
$\dot{\boldsymbol{q}}$	joint velocities
$\ddot{\boldsymbol{q}}$	joint accelerations
$\boldsymbol{M}(\boldsymbol{q})$	symmetric, positive-definite mass matrix, depending on the position
$\boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}})$	Coriolis and centrifugal torques, depending on the position and velocity
$\boldsymbol{\tau}_g(\boldsymbol{q})$	gravity torques, depending on the position
$\boldsymbol{\tau}$	actuator torques

2.2.2. Operational Space

For the operational space, the dynamics equation is according to Eqn. (2.38) of [13]

$$\boldsymbol{\Lambda}(\boldsymbol{x})\,\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}}) + \boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{f} \tag{2.2}$$

where

$\boldsymbol{x}$	operational coordinates, describing end-effector position and orientation
$\dot{\boldsymbol{x}}$	first derivation of the operational coordinates, operational-space velocity
$\ddot{\boldsymbol{x}}$	second derivation of the operational coordinates, operational-space acceleration
$\boldsymbol{\Lambda}(\boldsymbol{x})$	operational space mass matrix depending on $\boldsymbol{x}$
$\boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}})$	vector of the velocity-product depending on $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$
$\boldsymbol{p}(\boldsymbol{x})$	gravity forces depending on $\boldsymbol{x}$
$\boldsymbol{f}$	force exerted on the end-effector

The position components of $\boldsymbol{x}$ can be represented in cartesian, spherical, cylindrical or any other suitable representation. The orientation components of $\boldsymbol{x}$ can be expressed by direction cosines, Euler angles, Euler parameters (quaternions), or any other representation method.

2.2.3. Relation Joint and Operational Space

The mapping between end-effector forces and joint forces is given by Eqn. (2.51) of [13]

$$\boldsymbol{\tau} = \boldsymbol{J}^T \boldsymbol{f} \tag{2.3}$$

where $\boldsymbol{J}$ is the Jacobian Matrix of the described manipulator. The equation can be inverted using the dynamically consistent generalized inverse of the Jacobian, called $\overline{J}^T$. Further explanations to this type of generalized inverse can be found on page 84 of [20]. The other relations between the variables in 2.1 and 2.2 are given in Eqn. (2.49) – (2.50) and Eqn. (2.52) – (2.54) of [13] with

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}} \tag{2.4}$$
$$\ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} \tag{2.5}$$
$$\boldsymbol{\Lambda} = (\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^T)^{-1} \tag{2.6}$$
$$\boldsymbol{\mu} = \boldsymbol{\Lambda}(\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{c}\dot{\boldsymbol{q}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) \tag{2.7}$$
$$\boldsymbol{\rho} = \boldsymbol{\Lambda}\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{\tau}_{\boldsymbol{g}} \tag{2.8}$$

These equations assume that the dimension of operational-space coordinates is less or equal to the dimension of the joint-space coordinates, and that the Jacobian $\boldsymbol{J}$ has full rank.

2.3. Online Trajectory Generation

This section introduces the functionality and nomenclature used for the OTG, further elaborations and details can be found in [24, 21, 23]. Since the OTG is the basis for the development of the Dynamic Online Trajectory Generator in Ch. 5, it is important to understand how the OTG works. The DOTG will later be described based on the nomenclature of the OTG.

2.3.1. OTG Input

For PC- or micro-controller-based robot motion control systems, we assume a time-discrete system with a set of time instants

$$\boldsymbol{t} = \{t_0, \ldots, t_i, \ldots, t_N\} \tag{2.9}$$
$$\text{with } t_i = t_{i-1} + t_{cycle} \text{ and } i \in \{1, \ldots, N\}, \tag{2.10}$$

where t_{cycle} represents the cycle time of the system. Time-discrete values are represented by the additional subscript i, time-continuous values do not have this subscript. The position of the robotic system at time t_i is $\boldsymbol{q}_i = ({}_1q_i, \ldots, {}_kq_i, \ldots, {}_Kq_i)^T$, where K is the number of DOFs. Velocities, accelerations, and jerks are analogously represented by $\dot{\boldsymbol{q}}_i$, $\ddot{\boldsymbol{q}}_i$, and $\dddot{\boldsymbol{q}}_i$.

A complete motion state at time t_i is described by the matrix

$$\begin{aligned} \boldsymbol{\Xi}_i &= (\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{q}}_i) && (2.11) \\ &= ({}_1\boldsymbol{\xi}_i, ..., {}_k\boldsymbol{\xi}_i, ..., {}_K\boldsymbol{\xi}_i)^T. && (2.12) \end{aligned}$$

${}_k\boldsymbol{\xi}_i$ stands for the k-th row of the motion state matrix $\boldsymbol{\Xi}_i$. The desired target motion state at time instant t_i is called $\boldsymbol{\Xi}_{i,trgt}$.

The kinematic-motion constraints at a time t_i are denoted as

$$\boldsymbol{B}_i = (\{\dot{\boldsymbol{q}}_{i,min}, \dot{\boldsymbol{q}}_{i,max}\}, \{\ddot{\boldsymbol{q}}_{i,min}, \ddot{\boldsymbol{q}}_{i,max}\}, \{\dddot{\boldsymbol{q}}_{i,min}, \dddot{\boldsymbol{q}}_{i,max}\}) \tag{2.13}$$

and constrain the motion state $\boldsymbol{\Xi}_i$ in the following way:

$$\begin{aligned} &\forall\, k \in \{1,\ldots,K\} \wedge \forall\, i \in \{1,\ldots,N\} : \\ &{}_k\dot{q}_{i,min} \leq_k \dot{q}_i \leq_k \dot{q}_{i,max} \wedge {}_k\ddot{q}_{i,min} \leq_k \ddot{q}_i \leq_k \ddot{q}_{i,max} \wedge {}_k\dddot{q}_{i,min} \leq_k \dddot{q}_i \leq_k \dddot{q}_{i,max} \end{aligned} \tag{2.14}$$

The selection vector $\boldsymbol{S}_i$ is a Boolean vector and determines which of the K DOFs have to be controlled by the OTG algorithm. The DOFs, which are controlled by other open- or closed-loop controllers are not considered by the algorithm.

All these variables combined are the input parameters for the OTG, together they form a the input matrix

$$\boldsymbol{W}_i = (\boldsymbol{\Xi}_i, \boldsymbol{\Xi}_{i,trgt}, \boldsymbol{B}_i, \boldsymbol{S}_i) \tag{2.15}$$

2.3.2. OTG Functionality

The OTG algorithm receives at the moment t_i the command to find for all DOFs in real-time a motion trajectory $\boldsymbol{\Phi}_i(t)$ (cf. Eq. 2.22), which transfers time-synchronized the state of motion from $\boldsymbol{\Xi}_i$ to $\boldsymbol{\Xi}_{i,trgt}$ within the shortest possible time. This has to be done without exceeding the given motion constraints $\boldsymbol{B}_i$.

In the following is explained what is meant by time-synchronization, motion trajectory, time-optimality and real-time capability.

Time-Synchronization

One important property of OTG is, that all DOFs, which are selected for trajectory-following control, have to reach their target state of motion $\boldsymbol{\Xi}_{i,trgt}$ at the same time instant, namely at $t_{i,sync}$ in order to achieve *time-synchronization*. As consequence of this requirement, we can already state, that

$$t_n - t_{i,sync} \leq t_{cycle}\ . \tag{2.16}$$

Just to give an impression of time dimensions, t_{cycle} lies in the range of one millisecond or less, that is, the resulting OTG algorithms are designed to be applicable on a very low control level.

Motion Trajectory

The OTG limits velocities, accelerations and jerks, but the derivation of the jerk can become infinite in its value. In order to be time-optimal and allow time-synchronization between several DOFs, only the three jerk states

$$\dddot{\boldsymbol{q}}_i \in \{\dddot{\boldsymbol{q}}_{i,min}, \boldsymbol{0}, \dddot{\boldsymbol{q}}_{i,max}\} \tag{2.17}$$

are used to form the trajectories. Combining the three jerk possibilities in its various ways leads to a finite set of acceleration profiles that can be used to build trajectories for every possible input value $\boldsymbol{W}_i$. Every acceleration profile is a piecewise continuous combination of first order polynomials, a result of the integration of constant jerks. The position progression is therefore described by several third-order polynomials

$${}^l_k q_i(t) = a_3(t-\Delta t)^3 + a_2(t-\Delta t)^2 + a_1(t-\Delta t) + a_0 \tag{2.18}$$

for each DOF k, time-shifted by Δt, and calculated at time instant t_i. Several of these polynomials combined result in a continuous piecewise function of the complete position trajectory of DOF k. The index l describes which polynomial in the piecewise collection is meant. Polynomial vectors for position, velocity, acceleration, and jerk progressions are denoted by the terms ${}^{l}\boldsymbol{q}_i(t)$, ${}^{l}\dot{\boldsymbol{q}}_i(t)$, ${}^{l}\ddot{\boldsymbol{q}}_i(t)$, and ${}^{l}\dddot{\boldsymbol{q}}_i(t)$.

In summary we obtain a matrix of polynomials:

$$\begin{aligned} {}^{l}\boldsymbol{\Xi}_i(t) &= \left({}^{l}\boldsymbol{q}_i(t), {}^{l}\dot{\boldsymbol{q}}_i(t), {}^{l}\ddot{\boldsymbol{q}}_i(t), {}^{l}\dddot{\boldsymbol{q}}_i(t)\right) \\ &= \left({}^{l}_{1}\xi_i(t), \ldots, {}^{l}_{k}\xi_i(t), \ldots, {}^{l}_{K}\xi_i(t)\right)^T \end{aligned} \tag{2.19}$$

where a single row, that is, the polynomials of one single DOF k, is denoted by

$${}^{l}_{k}\xi_i(t) = \left({}^{l}_{k}q_i(t), {}^{l}_{k}\dot{q}_i(t), {}^{l}_{k}\ddot{q}_i(t), {}^{l}_{k}\dddot{q}_i(t)\right) \tag{2.20}$$

Each set of motion polynomials ${}^{l}\boldsymbol{\xi}_i(t)$ for all K DOFs is accompanied by a set of time intervals

$${}^{l}\mathscr{V}_i = \left\{{}^{l}_{1}\vartheta_i, \ldots, {}^{l}_{k}\vartheta_i, \ldots, {}^{l}_{K}\vartheta_i\right\}, \text{ where } {}^{l}_{k}\vartheta_i = \left[{}^{l}_{k}t_i, {}^{(l+1)}_{k}t_i\right] , \tag{2.21}$$

in which a set of polynomials ${}^{l}_{k}\xi_i(t)$ for one single DOF k is valid. A complete motion trajectory $\boldsymbol{\Phi}_i(t)$ is finally composed of a set of motion polynomials with according time intervals:

$$\begin{aligned} \boldsymbol{\Phi}_i(t) &= \Big\{ \left({}^{1}\boldsymbol{\xi}_i(t), {}^{1}\mathscr{V}_i\right), \ldots, \left({}^{l}\boldsymbol{\xi}_i(t), {}^{l}\mathscr{V}_i\right), \\ &\quad \ldots, \left({}^{L}\boldsymbol{\xi}_i(t), {}^{L}\mathscr{V}_i\right) \Big\} \end{aligned} \tag{2.22}$$

Depending on the input value $\boldsymbol{W}_i$, the value L determines the minimally required number of polynomials to describe the complete trajectory from $\boldsymbol{\Xi}_i$ to $\boldsymbol{\Xi}_{i,trgt}$. Every DOF needs a different amount of polynomials, L describes the largest amount.

Time-Optimality

The OTG is a *kinematic time-optimal* algorithm. Kinematic-time optimal means that the system is transferred from an initial state of motion $\boldsymbol{\Xi}_i$ at instant t_i into a desired target state of motion $\boldsymbol{\Xi}_{i,trgt}$ within the shortest possible time without any consideration of couplings between individual DOFs. The whole system dynamics are not considered. The consequence of kinematic time-optimality is, that all K DOFs can be seen as linearly independent. This simplification does not apply for the *dynamic time-optimal* case, the problem dealt with in this thesis.

When calculating a motion trajectory $\boldsymbol{\Phi}_i(t)$ at the time instant t_i, the following requirements have to be fulfilled to achieve kinematic time-optimality:

$$\forall\, k \in \{1, \ldots, K\} :$$

$$\begin{aligned} &\text{Starts with first OTG call:} && {}^{1}_{k}t_i &&= t_i \\ &\text{Same start motion state:} && {}^{1}_{k}\boldsymbol{\xi}_i(t_i) &&= {}_{k}\boldsymbol{\Xi}_i \\ &\text{Continuous transition of polynomials:} && {}^{l}_{k}\boldsymbol{\xi}_i({}^{l}_{k}t_i) &&= {}^{l-1}_{k}\boldsymbol{\xi}_i({}^{l}_{k}t_i) \;\text{ with } l \in \{2, \ldots, L\} \\ &\text{Same target motion state:} && {}^{L}_{k}\boldsymbol{\xi}_i(t_{i,sync}) &&= {}_{k}\boldsymbol{\Xi}_{i,trgt} \end{aligned} \tag{2.23}$$

and inside the constraints:

$$\forall\, (k,l) \in \{1, \ldots, K\} \times \{1, \ldots, L\} :$$

$$\left.\begin{array}{rcccl} {}_k\dot{q}_{i,min} & \leq & {}^l_k\dot{q}_i(t) & \leq & {}_k\dot{q}_{i,max} \\ {}_k\ddot{q}_{i,min} & \leq & {}^l_k\ddot{q}_i(t) & \leq & {}_k\ddot{q}_{i,max} \\ {}_k\dddot{q}_{i,min} & \leq & {}^l_k\dddot{q}_i(t) & \leq & {}_k\dddot{q}_{i,max} \end{array}\right\} \text{ with } t \in \left[{}^l_k t_i,\ {}^{l+1}_k t_i\right] , \tag{2.24}$$

such that

$$t_i^{sync} \longrightarrow \min . \tag{2.25}$$

Equation (2.23) describes the single motion states at the beginning or at the end of a trajectory segment l, respectively. Due to Eqn. (2.24), it is guaranteed, that the motion variables are kept within their kinematic constraints. If the input values of $\boldsymbol{\Xi}_i$ and $\boldsymbol{\Xi}_{i,trgt}$ already exceed the constraints when the algorithm is first called, the OTG generates a trajectory, which first would bring the exceeding values back into range before continuing with a time-optimized trajectory to the target motion state, see also [23].

Real-Time Capability

Instead of calculating numerically for all possible acceleration profiles the solutions, the OTG uses a collection of interlaced decision trees to evaluate which acceleration profile will be the time-optimal solution to the given input $\boldsymbol{W}_i$. This saves enough computational costs to render the OTG real-time capable, assuming an algorithm execution time has to be faster than 1ms to be real-time capable. Such a exemplary decision tree is shown in Fig. 2.1.

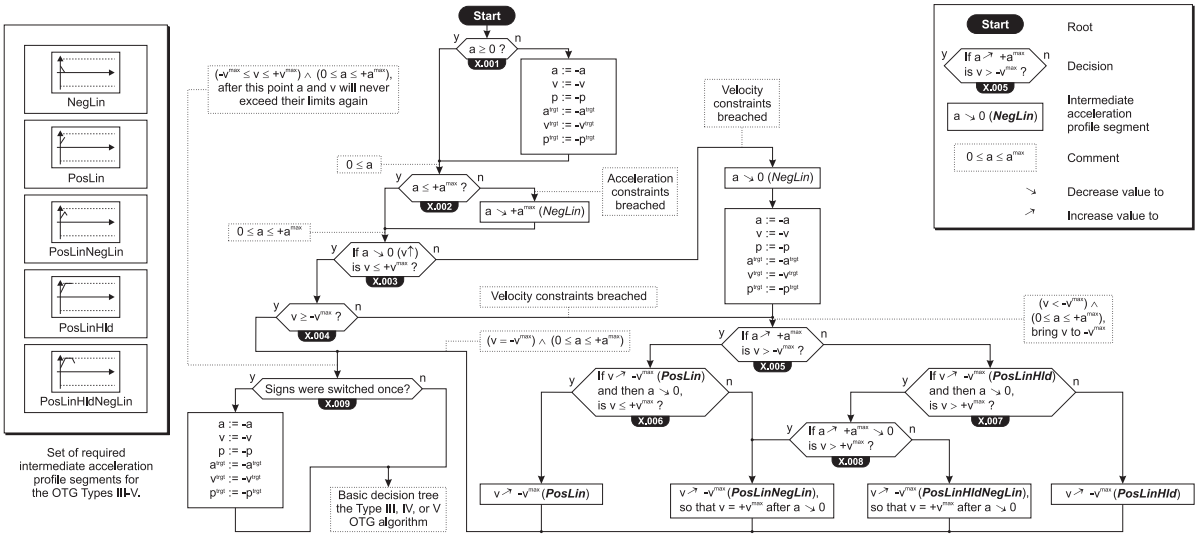


Figure 2.1.: OTG Decision Tree Example [23].

2.3.3. OTG Output

The OTG reads out from the generated trajectory the value of the next motion state $\boldsymbol{\Xi}_{i+1}$ at the time instant t_{i+1} and outputs it. The input-output behavior is shown with a flowchart in Fig. 2.2.

The outputted next motion state $\boldsymbol{\Xi}_{i+1}$ is used as input values for lower-level control. This leads to the requirement, that if the input values $\boldsymbol{\Xi}_{i,trgt}$, $\boldsymbol{B}_i$, and $\boldsymbol{S}_i$ remain constant for $i \in \{0, \ldots, N\}$, and if the output values $\boldsymbol{\Xi}_{i+1}$ are directly fed back as input values for the following control cycle (cf. dotted part of Fig. 2.2), then:

- The value of the synchronization time must remain constant during the whole trajectory execution, that is, $t_{i,sync} = \text{const}\ \forall\ i \in \{0, \ldots, N\}$. This fact is relevant for time-synchronization, such that $\dot{\boldsymbol{q}}_{i,trgt}$, $\ddot{\boldsymbol{q}}_{i,trgt}$, and $\dddot{\boldsymbol{q}}_{i,trgt}$ are coinstantaneously reached in $\boldsymbol{q}_{i,trgt}$ at $t_{i,sync}$.

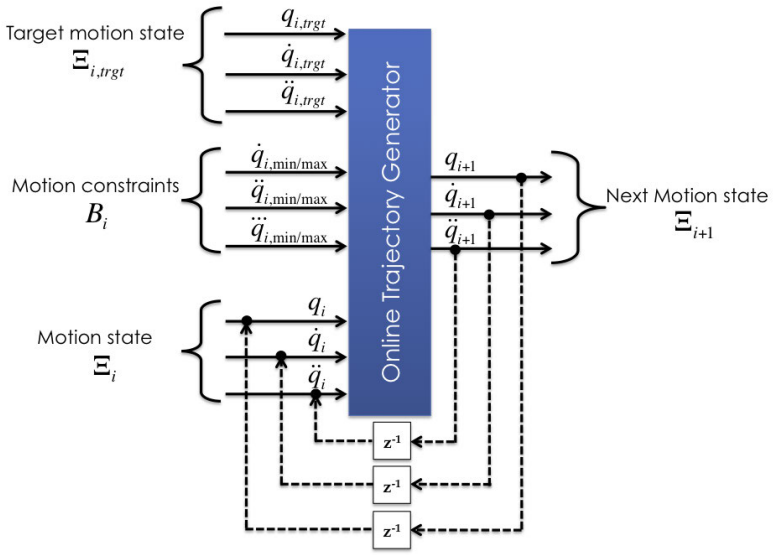


Figure 2.2.: Input and output values of the OTG. The dotted part indicates, how the output values of the OTG are usually fed back with z^{-1} representing a hold element of one time step.

- All trajectories $\mathbf{\Phi}_u(t)$ with $u \in \{1,\ldots,N\}$ must exactly fit into the one of $\mathbf{\Phi}_0(t)$, and
- furthermore, any trajectory $\mathbf{\Phi}_u(t)$ with $u \in \{1,\ldots,N\}$ must exactly fit into all previously calculated motion trajectories $\mathbf{\Phi}_v(t)$ with $v \in \{0,\ldots,u-1\}$.

For the current control cycle at t_i, only the next motion state $\mathbf{\Xi}_{i+1}$ would actually be needed, because in the next control cycle, we might have completely new input values $\boldsymbol{W}_i$ due to an unforeseen event or switching action. As we will see later during the DOTG development, also the complete motion trajectory $\mathbf{\Phi}_i(t)$ is needed at every time instant to deal with the varying acceleration capabilities.

2.3.4. OTG Versions

In the next two sections we will introduce the two version of the OTG: Target Position-Based OTG Version and the Target Velocity-Based OTG Version.

Target Position-Based OTG Version

The target position-based version generates a trajectory from any given start motion state $\mathbf{\Xi}_i$ to a desired target state $\mathbf{\Xi}_{i,trgt}$ with the restriction $\ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}$. This restriction is only existing for the so-called "Type IV"-version of the OTG algorithm, the "Type V" would not have this restriction, but this version has not been implemented yet. Figure 2.3 shows the input and output values of the target position-based algorithm.

The current implementation of the position-based OTG algorithm only works for symmetric motion constraints, that means

$$\begin{aligned} \dot{\boldsymbol{q}}_{i,min} &= -\dot{\boldsymbol{q}}_{i,max} \\ \ddot{\boldsymbol{q}}_{i,min} &= -\ddot{\boldsymbol{q}}_{i,max} \\ \dddot{\boldsymbol{q}}_{i,min} &= -\dddot{\boldsymbol{q}}_{i,max} \end{aligned} \tag{2.26}$$

In order to fully account for variable acceleration capabilities, the deployment of asymmetric acceleration constraints would be required. With symmetric constraints only parts of the potential acceleration capabilities will be used. This leaves room for future improvements of the OTG algorithm.

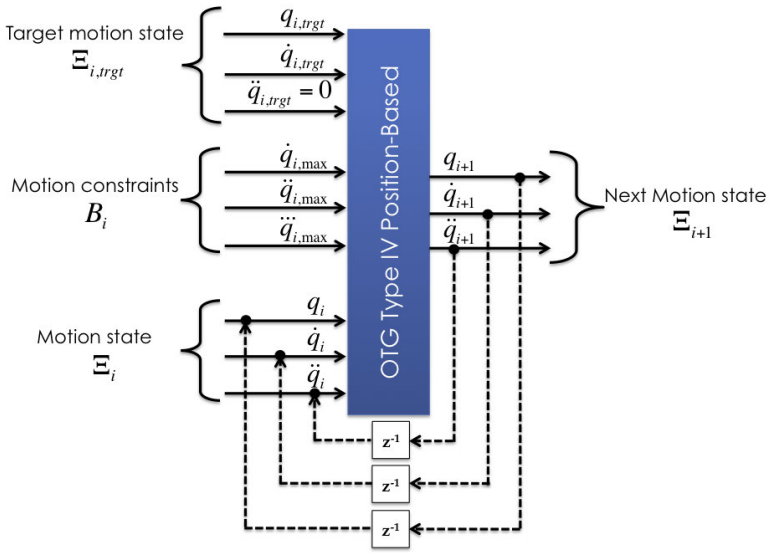


Figure 2.3.: Input and output values of the target position-based Type IV OTG algorithm.

Target Velocity-Based OTG Version

The target velocity-based version generates a trajectory from any given start motion state $\boldsymbol{\Xi}_i$ to a desired target velocity $\dot{\boldsymbol{q}}_{i,trgt}$ and a target acceleration $\ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}$. A target position can not be defined. The velocity-based algorithm outputs a position trajectory that has to be taken in order to bring $\dot{\boldsymbol{q}}_i$ as fast as possible to $\dot{\boldsymbol{q}}_{i,trgt}$. Figure 2.4 shows the input and output values for the target velocity-based algorithm.

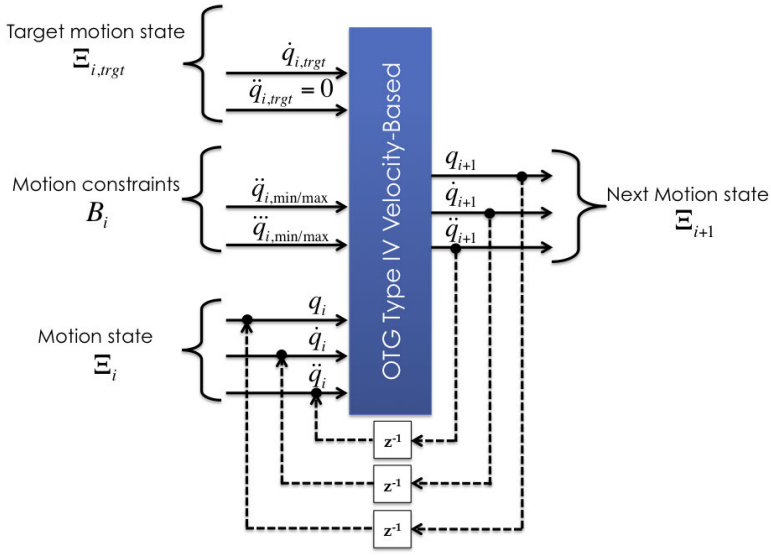


Figure 2.4.: Input and output values of the target velocity-based Type IV OTG algorithm.

The kinematic-motion constraints at a time t_i for the velocity-based OTG are denoted as

$$\boldsymbol{B}_i = (\{\ddot{\boldsymbol{q}}_{i,min}, \ddot{\boldsymbol{q}}_{i,max}\}, \{\dddot{\boldsymbol{q}}_{i,min}, \dddot{\boldsymbol{q}}_{i,max}\}) \tag{2.27}$$

The current implementation of the velocity-based OTG allows asymmetric motion constraints.

2.3.5. OTG Summary

To summarize this section briefly: We introduced a dedicated notation and described the input, functionality and output of the OTG algorithm. The algorithm is kinematic time-optimal and works for multiple DOF. The OTG runs stable and is real-time capable. The OTG time-synchronizes all active DOFs to ensure a smooth trajectory. Its major limitations are that only constant motion constraints can be used and the target accelerations have to be zero.

3. Dynamics and Acceleration Capabilities

A dynamic forward model of a mechanical system describes the resulting trajectory, in particular the resulting acceleration progression, in relationship to the applied actuation forces and/or torques. The inverse dynamic model calculates forces/torques if a motion trajectory is given.

3.1. Mapping Torque to Acceleration Capabilities

Acceleration capabilities can be described in Joint Space or Operational Space. In the following, I will provide the derivation for both cases.

3.1.1. Joint Space Mapping

The mapping of torques $\boldsymbol{\tau}$ to joint accelerations $\ddot{\boldsymbol{q}}$ is depicted in Fig. 3.1. In this figure, a simplified 2 DOF robot manipulator is shown.

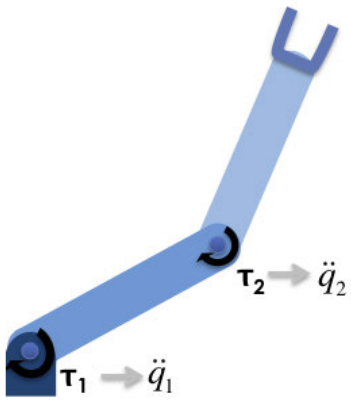


Figure 3.1.: Robot in joint space.

Using Eqn. (2.1) leads to the inverse dynamic model

$$\ddot{\boldsymbol{q}}_{extr} = \boldsymbol{M}(\boldsymbol{q})^{-1} \left(\boldsymbol{\tau}_{extr} - \boldsymbol{\tau}_g(\boldsymbol{q}) - c(\boldsymbol{q}, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}}\right) \tag{3.1}$$

where the extreme actuator torques $\boldsymbol{\tau}_{extr}$ are mapped to the extreme joint accelerations $\ddot{\boldsymbol{q}}_{extr}$. The corner values $\ddot{\boldsymbol{q}}_{extr}$ can be found through combination of the extreme torque values ${}_k\tau_{extr}$ of each individual actuator:

$$\forall\, k \in \{1, \ldots, K\}: \quad {}_k\tau_{extr} \in ({}_k\tau_{min},\ {}_k\tau_{max}) \tag{3.2}$$

$$\text{with} \quad {}_k\tau_{min} \leq {}_k\tau \leq {}_k\tau_{max} \tag{3.3}$$

Figure 3.2 shows for exemplary values the mapping of extremal actuator torques $\boldsymbol{\tau}_{extr}$ to extremal joint accelerations $\ddot{\boldsymbol{q}}_{extr}$. The blue rectangle describes the maximum torque capabilities with ${}_k\tau$ being extremal for at least one degree of freedom. Using Eqn. (3.1), the blue rectangle is mapped to the red parallelogram describing the extremal joint acceleration capabilities of the manipulator, where ${}_k\ddot{q}$ is extremal for at least one degree of freedom.

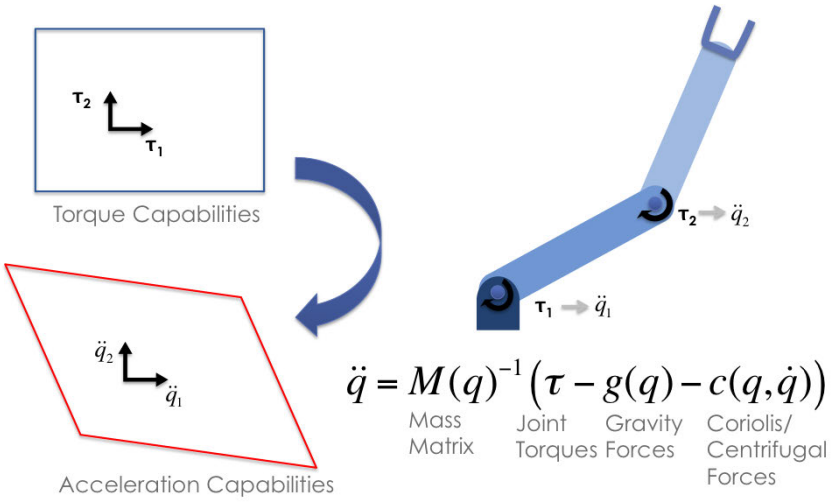
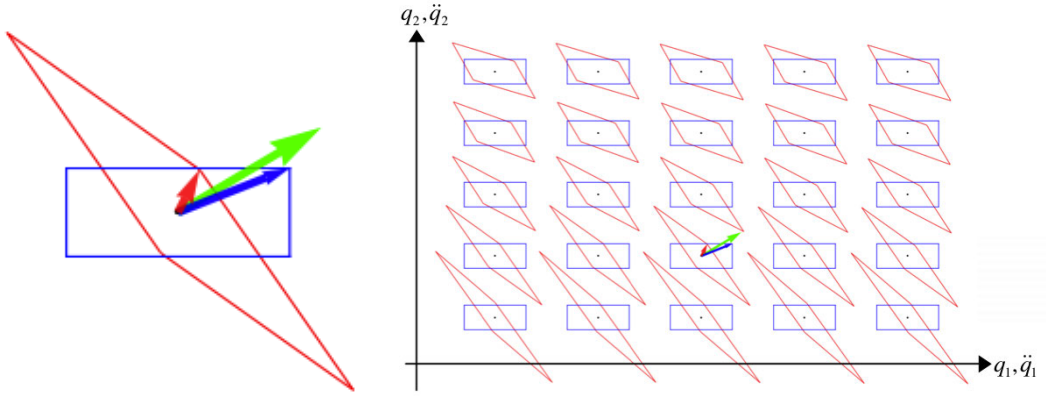


Figure 3.2.: Mapping Torque capabilities to Acceleration capabilities in Joint Space.

An extreme torque combination, for instance $({}_1\tau_{max},\ {}_2\tau_{max})^T$, is shown by the blue arrow pointing to one corner of the blue rectangle in Fig. 3.3a. The blue arrow maps to the red arrow, which stands for the equivalent extreme acceleration corner. Therefore, the blue rectangle maps to the red parallelogram. This mapping is done for the robot in Fig. 3.1 at a certain joint position configuration with a given joint velocity $\dot{\boldsymbol{q}}$. The green arrow in Fig. 3.3a describes the direction of the current joint velocity.

The figure is an excerpt of Fig. 3.3b, where the mapping for other joint position configurations at the same velocity is given as well. The figure shows joint position combinations for both axes in the range of 0° to 180° .



(a) Zoom to one single configuration.

(b) Acceleration capabilities for a Configuration Range in Joint Space.

Figure 3.3.: Acceleration capabilties in Joint Space for one and several configurations.

3.1.2. Operational Space Mapping

High-level motion and force commands are issued in the frame of the end-effector, which is also called operational space (operational space).The mapping of torques $\boldsymbol{\tau}$ to end-effector accelerations $\ddot{\boldsymbol{x}}$ is illustrated in Fig. 3.4. A two DOF robot manipulator is used with the torques mapped to the accelerations of a frame fixed to the end-effector of the manipulator.

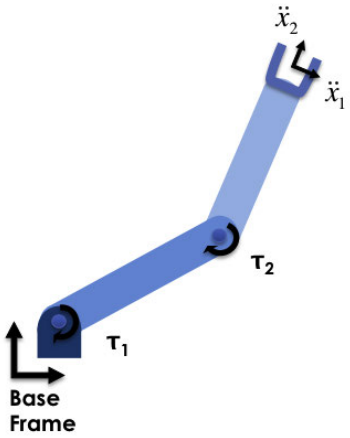


Figure 3.4.: Robot in operational space.

Equation (2.2) and (2.3) combined lead to

$$\ddot{\boldsymbol{x}}_{extr} = \boldsymbol{\Lambda}(\boldsymbol{x})^{-1} \left(\bar{J}^T \boldsymbol{\tau}_{extr} - \boldsymbol{p}(\boldsymbol{x}) - \boldsymbol{\mu}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \right) \tag{3.4}$$

where the extremal actuator torques $\boldsymbol{\tau}_{extr}$ are mapped to the extremal operational space accelerations $\ddot{\boldsymbol{x}}_{extr}$. All the corner values $\ddot{\boldsymbol{x}}_{extr}$ are found through the combinations of the extremal torques, see Eqn. (3.3).

In Fig. 3.5 exemplary values for the extremal actuator torques $\boldsymbol{\tau}_{extr}$ are mapped to extremal operational space accelerations $\ddot{\boldsymbol{x}}_{extr}$. The blue rectangle stands for the extremal actuator capabilities with ${}_k\tau$ being extreme for at least one degree of freedom. This is mapped using Eqn. (3.4) to the red parallelogram describing the extremal operational space acceleration capabilities of the manipulator, where ${}_k\ddot{x}$ is extremal for at least one degree of freedom.

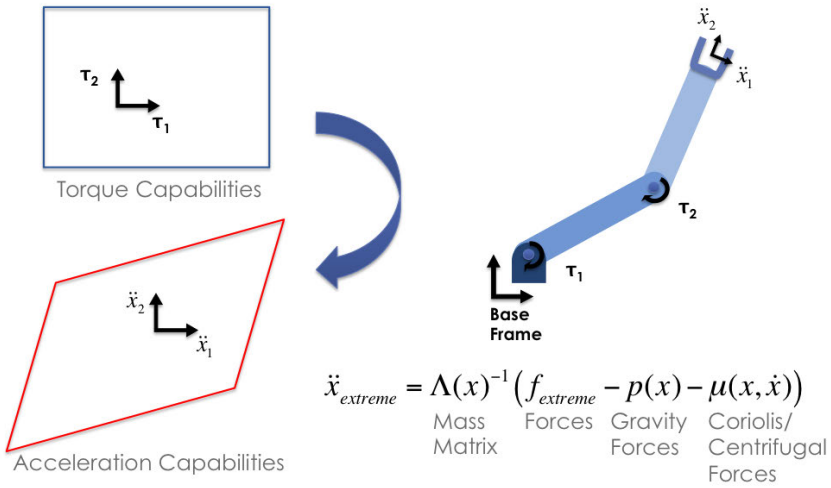


Figure 3.5.: Mapping torque capabilities to operational space acceleration capabilities.

The example in Fig. 3.6 shows for several robot configurations the operational space acceleration capabilities. The single red dots describe the end-effector frame origins of various end-effector positions distributed over one quadrant of the robot workspace. The red parallelogram qualitatively shows the extreme acceleration capabilities, measured from the red dot inside the parallelogram. In case of singular configuration positions, for instance the robot is fully stretched out, the red dots are outside the respective parallelograms for these configurations. Only

non-zero accelerations towards the center are possible if a red dot is outside the parallelogram. The green arrow gives the operational space velocity-direction used for calculating the capability parallelograms.

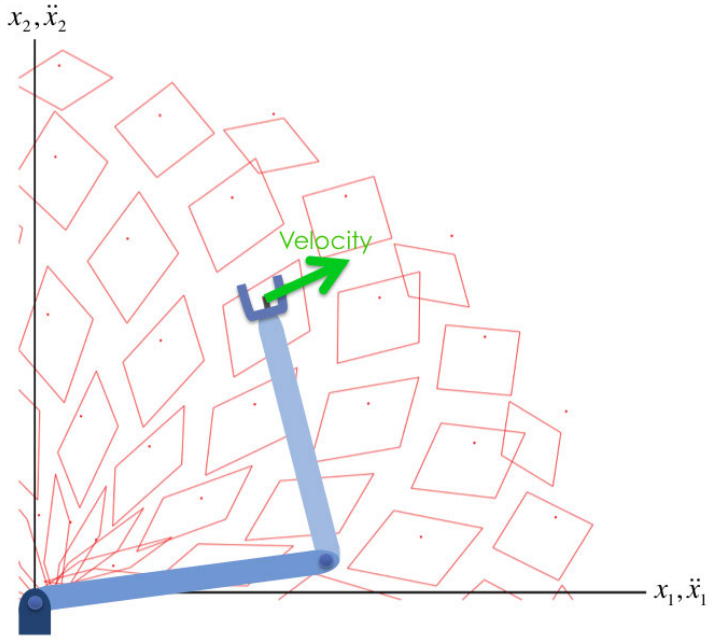


Figure 3.6.: Acceleration capabilities for the whole configuration in operational space.

3.2. Acceleration Capabilities as Parallelotopes

In this section will be described how to describe the acceleration capabilities geometrically and how this can be useful for finding optimized motion trajectories.

3.2.1. Parallelotopes

The two DOF examples given in Sec. 3.1 highlight that the mapping of actuator forces to joint or operational space coordinates is essentially a linear transformation. The Coriolis/centrifugal torques and the gravity torques cause an offset and the mass matrix does the linear transformation.

For two dimensions, the torque rectangles are mapped to a *parallelogram* with a shifted origin compared to the torque-box origin. In three dimensions, the shape of the acceleration capabilities is called *parallelepiped*. For the general case of n-dimensions, the shape is called *parallelotope (PT)*.

Figure 3.7 visualizes this geometrically: Moving a point along a certain direction from an initial position to a final position traces out a *line segment*. Translating this line segment along a new, linearly independent direction will form a *parallelogram*. Translating it again along a new, linearly independent direction will trace out a *parallelepiped*. The generalization of this to n-dimensions is called *parallelotope*.

The origin of the n-dimensional acceleration frame is not necessarily in the geometric center of the PTs, but can be anywhere inside or even outside of these geometric bodies. If the origin is inside the PT, the manipulator could theoretically accelerate in all directions with values anywhere between zero and the extremal values described by the bounds of the PT. Any chosen acceleration direction can also be extended to its negative direction.

For the case that the sum of Coriolis/centrifugal and gravity torques surpass the available actuator torques, the origin of the acceleration frame will move outside of the PT. In this case, the manipulator can only accelerate in one of the directions laying inside a cuted cone with the minimal and maximal acceleration values of a direction vector inside the cone have the same sign. The apex point of the cone is the origin of the acceleration frame, the

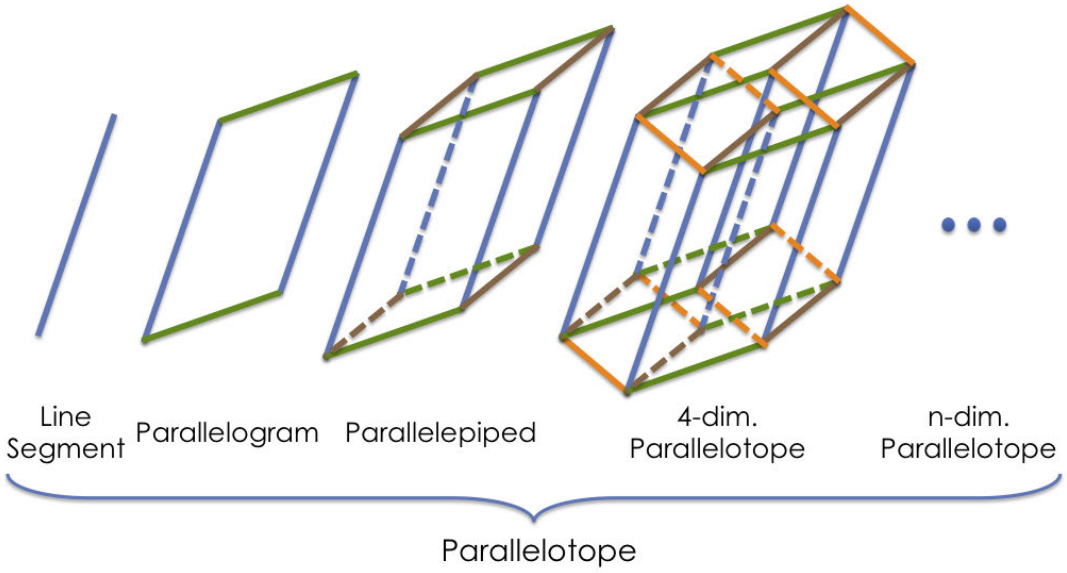


Figure 3.7.: Visualization of parallelotopes.

cap point is the first intersection of the central ray with the PT, the base point is the second intersection with the PT.

In the further discussions we assume, that actuator torques are always high enough, so that the origin of the acceleration frame is always inside of the PT.

Remark 3.2.1 *The notation for joint accelerations ($\ddot{q}$) is used for the rest of this chapter. The same description applies also for operational space accelerations.*

For any given acceleration, we can construct a line $L(\mathbf{0}, \ddot{q}_{given})$ defined as a ray passing through the frame origin as the first point and the acceleration $\ddot{q}_{given}$ as a second point. Based on the extremal accelerations calculated in Eqn. (3.1) – (3.3), PT can be formed. The intersection of the line and the PT leads always to two extremal acceleration points

$$\ddot{q}_{1/2} \quad = \quad L(\mathbf{0}, \ddot{q}_{given}) \cap PT(\ddot{q}_{extr}) \tag{3.5}$$

3.2.2. Reduzed-Size Parallelotopes using Jerk Hypercubes

The acceleration capabilities are shaped like a PT. Transforming the PT coordinate frame so that every boundary becomes perpendicular to each other does not work. Only perpendicular boundaries as in a rectangular box can be described by independent extreme acceleration values for every individual axis. This leads to the problem that in the general case with more than one degree of freedom, we can not define acceleration capabilities for each degree of freedom, which are independent to each other.

If we don't have a motion trajectory, we would not know from which to which motion state we will progress in the next instant. We could theoretically accelerate in any direction in the next time instant. Therefore, we have to take into account that all sides of the PT are possible acceleration constraints. The PT shape is continuously changing when moving along a trajectory, therefore the constraints should be adjusted for every time instant a trajectory generator is relying on the acceleration constraints as input values. The OTG algorithm is relying on accurate acceleration constraints as input values, this is why the constraints need to be adjusted for every time instant.

For a jerk limited trajectory generator, which means that every DOF has a constant jerk constraints, we can do an approximation for every single time instant. A given acceleration state $\ddot{q}_{given}$ can not change more than by the product of the extremal jerk $\dddot{q}_{extr}$ times the length of the considered single time instant Δt:

$$\Delta\ddot{\boldsymbol{q}} = \dddot{\boldsymbol{q}}_{extr}\,\Delta t \tag{3.6}$$

$$\forall\, k \in \{1,\ldots,K\}: \quad {}_k\ddot{\boldsymbol{q}}_{extr} \in ({}_k\ddot{\boldsymbol{q}}_{min}, {}_k\ddot{\boldsymbol{q}}_{max}) \tag{3.7}$$

This leads to a limiting n-dimensional hypercube around the current acceleration state, in which the acceleration could change within the next time instant.

In order to minimize the problem of finding the linear independent acceleration constraints for an unknown path, we could use acceleration hypercubes to narrow the PT boundaries. The hypercubes can be shifted linearly along the coordinate system axis until they touch the hull of the PT. Moving the cubes along the hull while touching it, leads to a smaller PT. This is illustrated for two dimensions in Fig. 3.8. We can now consider this smaller PT in the following section, but this approach still leaves the problem open on how to now calculate linear independent acceleration constraints that can be used for a Dynamic OTG.

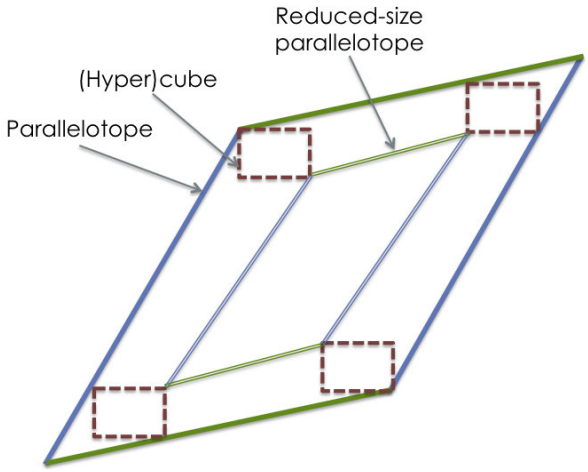


Figure 3.8.: Jerk Hypercubes Approach - Reduzed-Size Parallelotopes.

3.2.3. Features of Parallelotopes

For every motion state the PT changes it shape and position in relation to the acceleration frame. We therefore could qualify a PT by certain features, for example we define the minimum distance from the origin to the PT. This could be done by placing into the PT a circle/sphere/hypersphere with its center in the frame origin and then extend the hypersphere's radius till it touches the PT. This is shown by the red circle and the red arrow in Fig. 3.9. Another way of describing this is to find the shortest distance between all PT boundaries and the origin and then choose the smallest of these. The shortest distance can be calculated by finding a perpendicular hyperline that intersects the PT's bounding hyperplanes in an angle of 90°and goes through the frame origin. This is shown for the 2D case in Fig. 3.9 by the four black arrows.

We could also define a maximal distance to the furthest corner of the PT in order to define another feature of it. Furthermore, we could extend every coordinate axis in the acceleration frame and intersect them with the PT, according to Eqn. (3.5) we would then find two extremal acceleration values. We would get 2 times K acceleration vectors, with K standing for the total number of degrees of freedom. These values would then qualify a PT for a given position and velocity.

3.2.4. Parallelotopes for the Whole Configuration Field

For the current position, we could then vary the velocity to several values between its constraints and get for every value an additional PT description. We would then repeat this for other positions by evenly distribute points in

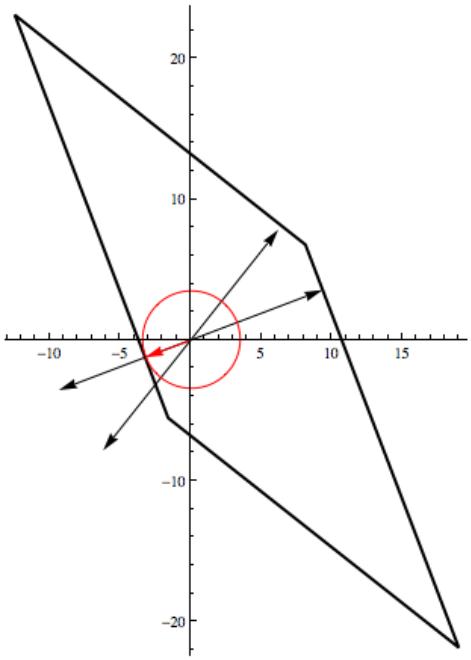


Figure 3.9.: Qualifying parallelotope features: Find the shortest distance to every boundary (black arrows) and identify the shortest distance to the origin (red arrow/circle).

the whole position configuration space (cf. Fig. 3.3b and 3.6). Depending on the resolution of this position field and the amount of velocity values stored, we could get a huge data collection of PT values. The nodes of this found network would be the motion states and the edges between the nodes would describe the acceleration value between the one motion state to the other motion state. The value of the acceleration would then stand for the weight of the edge. This data network would then allow for a graph search along the branches in order to find an optimal acceleration path that could be used.

This method seems promising as an offline approach to find a first feasible solution for a motion path, but it is not possible to merge this approach with the OTG algorithm, which only takes start and target motion states as inputs. We therefore have to look into an alternative way of qualifying a PT for use as acceleration constraints of the OTG. The finally used Path Dynamics approach is such an alternative and will be explained in the next chapter.

3.3. Summary - Dynamics and Acceleration Capabilities

In this chapter has been described how acceleration capabilities are calculated for joint space (cf. Fig. fig:3.2) and operational space (cf. Fig. 3.5). Furthermore, a geometric description of the acceleration capabilities as so-called parallelotopes has been given, see Fig. 3.7.

In addition, concepts on how to use parallelotopes for optimized path planning are given: Reduze the size of the parallelotopes according to the jerk limitation (cf. Fig. 3.8) and qualify these smaller parallelotopes by its features (cf. Fig. 3.9) in order to do a graph search along a network of parallelotope parameter values calculated for the whole configuration space (cf. Fig. 3.3b and 3.6).

4. Acceleration Capabilities on a Path

In Sec. 3.2 is described how the acceleration capabilities due to the dynamic limitations of the system can be described using a geometric shape called parallelotope. All acceleration points inside of this n-dimensional geometric shape are theoretically achievable for a given position and velocity of a robot.

If you want to find the global optimum for a motion trajectory that considers the complete system's dynamics, the graph search approach described in Sec. 3.2.4 could help to find a feasible initial solution for a path from start to end point. This initial path would then need to be iteratively optimized. This approach becomes rather complicated and computationally very expensive when applying it to higher degrees of freedom.

Since the scope of this thesis is to develop an algorithm which can do real-time trajectory optimization under consideration of dynamics, we choose to optimize the path using a initial OTG trajectory solution. In this chapter is shown how we calculate the dynamics along a motion path and then make these results useful for the OTG by applying the Path Dynamics Merging Algorithm to them.

4.1. Path Dynamics

A given starting motion trajectory simplifies the analysis of the acceleration capabilities to a one-dimensional path problem. For a multiple DOF robot, this simplification eases the calculation of linearly independent acceleration constraints, as it will be shown later in this chapter.

4.1.1. One-Dimensional Path Representation

We call $\boldsymbol{x}(\boldsymbol{q}(t))$ the normal representation to describe a multidimensional position trajectory depending on the joint positions $\boldsymbol{q}(t)$. According to [7], the normal representation $\boldsymbol{x}(\boldsymbol{q}(t))$ can also be described in dependency to the one-dimensional path variable $s(t)$. We call $\boldsymbol{r}(s(t))$ the alternative representation:

$$\boldsymbol{x}(\boldsymbol{q}(t)) := \boldsymbol{r}(s(t)) \tag{4.1}$$

The path describes the arc length of a continuous position progression and is mathematically defined as the integral of the norm of the velocity vector:

$$s(t) = \int_{t_0}^{t} ||\dot{\boldsymbol{x}}(\boldsymbol{q}(t))||\, dt \tag{4.2}$$

The norm of the velocity vector $||\dot{\boldsymbol{x}}(\boldsymbol{q}(t))||$ is a unit vector tangential to the path, therefore called "tangential path vector". The velocity progression of the trajectory computed by the OTG algorithm is defined as a piecewise function of polynomials. These polynomials can be up to second order. The piecewise defined time intervals are not the same for every DOF. It is therefore not possible to solve the integral in Eqn. (4.2) in an analytical way, but fortunately only the derivations of $s(t)$ are needed, as it will be shown later on.

Remark 4.1.1 *In order to improve legibility, the dependency to time will be dropped.*

Differentiating the normal representation $\boldsymbol{x}(\boldsymbol{q})$ with respect to time yields

$$\dot{\boldsymbol{x}}(\boldsymbol{q}) = \frac{\partial \boldsymbol{x}(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial t} = \boldsymbol{J}(\boldsymbol{q})\, \dot{\boldsymbol{q}} \tag{4.3}$$

With $\boldsymbol{J}$ describing the Jacobian matrix of $\boldsymbol{x}$ with respect to $\boldsymbol{q}$.

Differentiating the alternative representation $\boldsymbol{r}(s)$ with respect to time yields

$$\dot{\boldsymbol{x}}(\boldsymbol{q}) = \frac{\partial \boldsymbol{r}(s)}{\partial s}\,\dot{s} = \boldsymbol{r}_s(s)\,\dot{s} \tag{4.4}$$

With $\boldsymbol{r}_s(s)$ describing the first partial derivation of $\boldsymbol{r}(s)$ by s.

The second derivation of the normal representation leads to:

$$\begin{aligned} \ddot{\boldsymbol{x}}(\boldsymbol{q}) &= \boldsymbol{J}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \frac{\partial \boldsymbol{J}(\boldsymbol{q})}{\partial \boldsymbol{q}}\,\frac{\partial \boldsymbol{q}}{\partial t}\,\dot{\boldsymbol{q}} \\ &= \boldsymbol{J}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{H}(\boldsymbol{q})\,\dot{\boldsymbol{q}}^2 \end{aligned} \tag{4.5}$$

where $\boldsymbol{H}$ stands for the Hessian matrix of $\boldsymbol{x}$ with respect to $\boldsymbol{q}$.

The second derivation of the alternative representation follows analogously:

$$\begin{aligned} \ddot{\boldsymbol{x}}(\boldsymbol{q}) &= \ddot{\boldsymbol{r}}(s) \\ &= \boldsymbol{r}_s(s)\,\ddot{s} + \frac{\partial \boldsymbol{r}_s(s)}{\partial s}\,\frac{\partial s}{\partial t}\,\dot{s} \\ &= \underbrace{\boldsymbol{r}_s(s)\,\ddot{s}}_{tang.acc.} + \underbrace{\boldsymbol{r}_{ss}(s)\,\dot{s}^2}_{norm.acc.} \end{aligned} \tag{4.6}$$

The expression $\boldsymbol{r}_{ss}(s)$ describes the second partial derivation of $\boldsymbol{r}(s)$ by s. Eqn. (4.6) shows, that the acceleration can be split into a tangential component along the path and a normal component perpendicular to the path.

The Fig. 4.1 shows an example path s from start to target. For two exemplary motion states, the velocity vectors are shown in orange. Velocity vectors are always tangential to the path. The tangential and normal acceleration vectors are also shown for one of the motion states. The normal acceleration is shown in yellow, the tangential acceleration in green. The two acceleration vectors combined would lead to the acceleration of that particular motion state, but it is not shown in the figure.

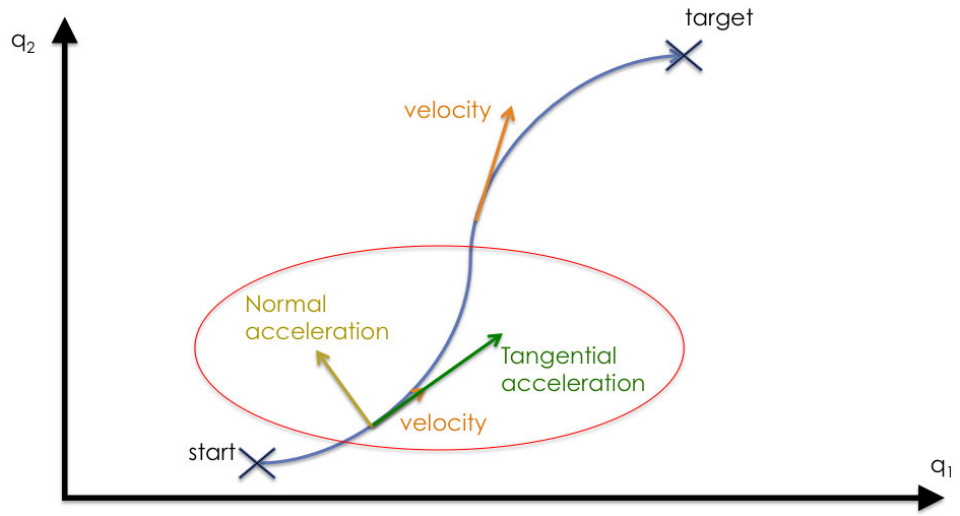


Figure 4.1.: An exemplary motion path from start to target with two velocity vectors and a normal and tangential acceleration vector.

To prove the statements about the figure, we derive Eqn. (4.2) by time which leads to:

$$\dot{s} = ||\dot{\boldsymbol{x}}(\boldsymbol{q})|| \tag{4.7}$$

The change along the path is the norm of the current velocity. Combining Eqn. (4.7) with Eqn. (4.4) leads to:

$$\dot{\boldsymbol{x}}(\boldsymbol{q}) = \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})}{||\dot{\boldsymbol{x}}(\boldsymbol{q})||} \, ||\dot{\boldsymbol{x}}(\boldsymbol{q})|| = \boldsymbol{r}_s(s) \, \dot{s} \tag{4.8}$$

This shows that the velocity consist of a tangential path unit vector

$$\boldsymbol{r}_s(s) = \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})}{||\dot{\boldsymbol{x}}(\boldsymbol{q})||} = \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})}{\dot{s}} \tag{4.9}$$

$\boldsymbol{r}_s(s)$ multiplied with the scalar path velocity $\dot{s}$ results in the velocity vector. $\boldsymbol{r}_s(s)$ times $\ddot{s}$ results in the tangential acceleration of Eqn. (4.6) and the vector $\boldsymbol{r}_{ss}(s)$ describes the change along the path s and is therefore always perpendicular to $\boldsymbol{r}_s(s)$. $\boldsymbol{r}_{ss}(s)$ is the normal path vector.

$\boldsymbol{r}_{ss}(s)$ is calculated using Eqn. (4.6):

$$\boldsymbol{r}_{ss}(s) \quad = \quad \frac{\ddot{\boldsymbol{x}}(\boldsymbol{q}) \; - \; \boldsymbol{r}_s(s) \, \ddot{s}}{\dot{s}^2} \tag{4.10}$$

The second derivation of s is calculated in the following way:

$$\begin{aligned}
\ddot{s} &= \frac{\partial \dot{s}}{\partial t} \\
&= \frac{\partial ||\dot{\boldsymbol{x}}(\boldsymbol{q})||}{\partial t} \\
&= \frac{\partial \sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}{\partial t} \\
&= \frac{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}} \\
&= \frac{\dot{\boldsymbol{x}}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{x}}(\boldsymbol{q})}{||\dot{\boldsymbol{x}}(\boldsymbol{q})||}
\end{aligned} \tag{4.11}$$

After having shown that the velocity and acceleration can be split into its tangential and normal components, in the next section is shown how these components are calculated for any given motion state.

4.1.2. Calculation of the Path Representation

The first approach to calculate the path representation variables was done in a numerical manner and is fully documented in Sec. B.1. Fortunately, the calculation of the path representation can also be done avoiding any numerical methods: If the current motion state Ξ, consisting of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$, is given, we can calculate for the general case the components of the alternative representation as follows:

$$\dot{s} \quad = \quad ||\dot{\boldsymbol{x}}(\boldsymbol{q})|| \tag{4.12}$$

$$\ddot{s} \quad = \quad \frac{\dot{\boldsymbol{x}}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{x}}(\boldsymbol{q})}{\dot{s}} \tag{4.13}$$

$$\boldsymbol{r}_s(s) \quad = \quad \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})}{\dot{s}} \tag{4.14}$$

$$\boldsymbol{r}_{ss}(s) \quad = \quad \frac{\ddot{\boldsymbol{x}}(\boldsymbol{q}) - \boldsymbol{r}_s(s) \, \ddot{s}}{\dot{s}^2} \tag{4.15}$$

$\ddot{s}$, $\boldsymbol{r}_s(s)$ and $\boldsymbol{r}_{ss}(s)$ all have $\dot{s}$ in their denominator. In the special case, that $\dot{\boldsymbol{q}} == 0$ and therefore $\dot{s} == 0$, we have to consider L'Hôpital's Rule to find a feasible solution:

Case 1

with $\dot{\boldsymbol{x}}(\boldsymbol{q}) \to \mathbf{0}$ and $\ddot{\boldsymbol{x}}(\boldsymbol{q}) \to \mathbf{0}$:

- $\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \ddot{s}$:

$$\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \ddot{s} = \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})\cdot\ddot{\boldsymbol{x}}(\boldsymbol{q})}{\dot{s}} = \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\overbrace{\sum_{i=1}^{K} {}_i\dot{x}_i\dddot{x}}^{\to 0}}{\underbrace{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}_{\to 0}}$$

$$\overset{L'Hosp.}{=} \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\sum_{i=1}^{K}({}_i\ddot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\frac{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}} = \frac{\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\ddot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\underbrace{\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}}_{=\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0}\ddot{s}}} = \frac{\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\ddot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\ddot{s}}$$

$$\Rightarrow \quad \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \ddot{s}^2 = \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \sum_{i=1}^{K} \Big(\underbrace{{}_i\ddot{x}^2}_{\to 0} + \underbrace{{}_i\dot{x}}_{\to 0}\, {}_i\dddot{x} \Big) \overset{0<{}_i\dddot{x}<\infty}{=} 0 \quad \Rightarrow \quad \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \ddot{s} = 0 \tag{4.16}$$

- $\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \boldsymbol{r}_s(s)$:

$$\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \boldsymbol{r}_s(s) = \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\overbrace{\dot{\boldsymbol{x}}(\boldsymbol{q})}^{\to 0}}{\underbrace{||\dot{\boldsymbol{x}}(\boldsymbol{q})||}_{\to 0}} \tag{4.17}$$

$$\overset{L'Hosp.}{=} \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\overbrace{\ddot{\boldsymbol{x}}(\boldsymbol{q})}^{\to 0}}{\underbrace{||\ddot{\boldsymbol{x}}(\boldsymbol{q})||}_{\to 0}} \tag{4.18}$$

$$\overset{L'Hosp.}{=} \lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \frac{\dddot{\boldsymbol{x}}(\boldsymbol{q})}{||\dddot{\boldsymbol{x}}(\boldsymbol{q})||} \overset{0<{}_i\dddot{x}<\infty}{=} \frac{\dddot{\boldsymbol{x}}(\boldsymbol{q})}{||\dddot{\boldsymbol{x}}(\boldsymbol{q})||} \tag{4.19}$$

The tangential path vector therefore is the normed jerk.

- $\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \boldsymbol{r}_{ss}(s)$: Since the velocity and the acceleration are both zero, the tangential path vector becomes the normed jerk. $\boldsymbol{r}_{ss}(s)$ only needs to be perpendicular to this tangential path vector, which in the general case would lead to infinite possibilities of achieving this. We neither have any velocity nor acceleration, so there is no need to calculate this normal path vector. We assume for this case:

$$\lim_{\dot{\boldsymbol{x}},\ddot{\boldsymbol{x}}\to 0} \boldsymbol{r}_{ss}(s) = \mathbf{0} \tag{4.20}$$

Case 2

with $\dot{\boldsymbol{x}}(\boldsymbol{q}) \to \mathbf{0}$ and $\ddot{\boldsymbol{x}}(\boldsymbol{q}) \neq \mathbf{0}$:

- $\lim_{\dot{\boldsymbol{x}}\to 0} \ddot{s}$:

$$\lim_{\dot{\boldsymbol{x}}\to 0} \ddot{s} = \lim_{\dot{\boldsymbol{x}}\to 0} \frac{\dot{\boldsymbol{x}}(\boldsymbol{q})\cdot\ddot{\boldsymbol{x}}(\boldsymbol{q})}{\dot{s}} = \lim_{\dot{\boldsymbol{x}}\to 0} \frac{\overbrace{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}^{\to 0}}{\underbrace{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}_{\to 0}}$$

$$\overset{L'Hosp.}{=} \lim_{\dot{\boldsymbol{x}}\to 0} \frac{\sum_{i=1}^{K}({}_i\dot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\frac{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}} = \frac{\lim_{\dot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\dot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\underbrace{\lim_{\dot{\boldsymbol{x}}\to 0} \frac{\sum_{i=1}^{K} {}_i\dot{x}_i\ddot{x}}{\sqrt{\sum_{i=1}^{K} {}_i\dot{x}^2}}}_{=\lim_{\dot{\boldsymbol{x}}\to 0}\ddot{s}}} = \frac{\lim_{\dot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\dot{x}^2 + {}_i\dot{x}_i\dddot{x})}{\ddot{s}}$$

$$\Rightarrow \lim_{\dot{\boldsymbol{x}}\to 0} \ddot{s}^2 = \lim_{\dot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\ddot{x}^2 + \underbrace{{}_i\dot{x}}_{\to 0}\ \underbrace{{}_i\dddot{x}}_{0<{}_i\dddot{x}<\infty}) = \lim_{\dot{\boldsymbol{x}}\to 0}\sum_{i=1}^{K}({}_i\ddot{x}^2) \quad \Rightarrow \quad \lim_{\dot{\boldsymbol{x}}\to 0}\ddot{s} = ||\ddot{\boldsymbol{x}}|| \tag{4.21}$$

- $\lim_{\dot{\boldsymbol{x}}\to 0} \boldsymbol{r}_s(s)$:

$$\lim_{\dot{\boldsymbol{x}}\to 0} \boldsymbol{r}_s(s) = \lim_{\dot{\boldsymbol{x}}\to 0} \frac{\overbrace{\dot{\boldsymbol{x}}(\boldsymbol{q})}^{\to 0}}{\underbrace{||\dot{\boldsymbol{x}}(\boldsymbol{q})||}_{\to 0}} \tag{4.22}$$

$$\overset{L'Hosp.}{=} \lim_{\dot{\boldsymbol{x}}\to 0} \frac{\ddot{\boldsymbol{x}}(\boldsymbol{q})}{||\ddot{\boldsymbol{x}}(\boldsymbol{q})||} \overset{0<{}_i\ddot{x}<\infty}{=} \frac{\ddot{\boldsymbol{x}}(\boldsymbol{q})}{||\ddot{\boldsymbol{x}}(\boldsymbol{q})||} \tag{4.23}$$

- $\lim_{\dot{\boldsymbol{x}}\to 0} \boldsymbol{r}_{ss}(s)$: Since the velocity is zero, the tangential path vector becomes the normed acceleration. $\boldsymbol{r}_{ss}(s)$ now would only need to be perpendicular to this tangential path vector, which in the general case would lead to infinite solutions. We do not have any velocity, so there is physically no normal acceleration component possible. We assume for this case:

$$\lim_{\dot{\boldsymbol{x}}\to 0} \boldsymbol{r}_{ss}(s) = \boldsymbol{0} \tag{4.24}$$

In this section we have shown that we can calculate the path derivations and path vectors ($\dot{s}$, $\ddot{s}$, $\boldsymbol{r}_s(s)$ and $\boldsymbol{r}_{ss}(s)$) for any given motion state $\boldsymbol{\Xi}$.

4.1.3. Joint Space Path Dynamics

In the next step, we will consider the case that we control the robot in joint space. Therefore, the position trajectory in Eqn. (4.1) describes only the joint positions:

$$\boldsymbol{x}(\boldsymbol{q}) = \boldsymbol{q} = \boldsymbol{r}(s) \tag{4.25}$$

The Jacobian Matrix $\boldsymbol{J}$ in Eqn. (4.3) and (4.5) becomes the identity matrix and the Hessian Matrix $\boldsymbol{H}$ becomes the zero matrix.

$$\dot{\boldsymbol{x}}(\boldsymbol{q}) = \dot{\boldsymbol{q}} = \boldsymbol{r}_s(s)\cdot\dot{s} \tag{4.26}$$

$$\ddot{\boldsymbol{x}}(\boldsymbol{q}) = \ddot{\boldsymbol{q}} = \underbrace{\boldsymbol{r}_s(s)\ \ddot{s}}_{tang.acc.} + \underbrace{\boldsymbol{r}_{ss}(s)\ \dot{s}^2}_{norm.acc.} \tag{4.27}$$

Remark 4.1.2 *From this point on we use the short form of representing the derivations of $\boldsymbol{r}(s)$, that means $\boldsymbol{r}_s(s)$ is written short $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}(s)$ is written short $\boldsymbol{r}_{ss}$.*

We will now show how to derive the extremal joint space accelerations for a path described by the variable $\ddot{s}$. First, we substitute $\ddot{\boldsymbol{q}}$ in Eqn. (2.1) with Eqn. (4.27):

$$\boldsymbol{M}(\boldsymbol{q})\left(\boldsymbol{r}_s\,\ddot{s}+\boldsymbol{r}_{ss}\,\dot{s}^2\right)=\boldsymbol{\tau}\;-\;\boldsymbol{\tau}_g(\boldsymbol{q})\;-\;c(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} \tag{4.28}$$

Then we leaving only the term containing $\ddot{s}$ on the left-hand side:

$$\boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{r}_s\,\ddot{s}=\boldsymbol{\tau}\;-\;\boldsymbol{\tau}_g(\boldsymbol{q})\;-\;c(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}-\boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.29}$$

Introducing the abbreviations $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$:

$$\boldsymbol{\alpha} \;:=\; \boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{r}_s \tag{4.30}$$

$$\boldsymbol{\beta} \;:=\; \boldsymbol{\tau}_g(\boldsymbol{q})\;+\;c(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}+\boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.31}$$

This leads to the abbreviated form of Eqn. (4.29):

$$\boldsymbol{\alpha}\,\ddot{s} \;=\; \boldsymbol{\tau}\;-\;\boldsymbol{\beta} \tag{4.32}$$

The three variables $\boldsymbol{\tau}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ depend only on $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\dot{s}$, $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}$. $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ can be acquired from sensor inputs in the first instant the DOTG is used and after that, it can be calculated in every time step using the analytical descriptions of the polynomials generated by the OTG algorithm. $\dot{s}$, $\boldsymbol{r}_s$, $\boldsymbol{r}_{ss}$ are then calculated in every time step using the values of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. The calculation steps are given in Sec. 4.1.2.

We write the k-th row of Eqn. (4.32), which stands for the k-th DOF:

$${}_k\alpha\,\ddot{s} \;=\; {}_k\tau\;-\;{}_k\beta \tag{4.33}$$

In a next step, we take into account the torque capabilities, which leads to K individual inequalities, one for each DOF:

$${}_k\tau_{min}-\;{}_k\beta\;\leq\;{}_k\alpha\,\ddot{s}\leq\;{}_k\tau_{max}-\;{}_k\beta \tag{4.34}$$

Note that ${}_k\alpha\neq 0$, because the mass matrix $\boldsymbol{M}(\boldsymbol{q})$ is always positive definite and $\boldsymbol{r}_s$ is the non-zero path vector. Therefore, the Eqn. (4.34) can be written as:

$${}_kl\leq\;\ddot{s}\leq\;{}_kh \tag{4.35}$$

where

$${}_kl=\begin{cases}\frac{{}_k\tau_{min}-\;{}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha>0\\ \frac{{}_k\tau_{max}-\;{}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha<0\end{cases} \tag{4.36}$$

and

$${}_kh=\begin{cases}\frac{{}_k\tau_{max}-\;{}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha>0\\ \frac{{}_k\tau_{min}-\;{}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha<0\end{cases} \tag{4.37}$$

Equations (4.35) $\sim$ (4.37) describe the range of joint accelerations $\ddot{s}$ for which a single actuator can hold the manipulator on its joint path without violating the k-th constraint. To fulfill all the constraints

$$\ddot{s}\in[\;{}_kl,\;{}_kh\;]\text{ for }\forall k \tag{4.38}$$

has to be fulfilled. Only if the absolute velocity is too high, the intervals $[\;{}_kl,\;{}_kh\;]$ will have no intersection for $\forall k$. Otherwise there will be an intersection of all the intervals.

It follows that an admissible acceleration is any tangential acceleration $\ddot{s}$ that does not violate Eqn. (4.34) for $\forall k$, that is:

$$\ddot{s}_{min} \leq \ddot{s} \leq \ddot{s}_{max} \tag{4.39}$$

where

$$\ddot{s}_{min} = \max_k {}_k l \tag{4.40}$$

and

$$\ddot{s}_{max} = \min_k {}_k h \tag{4.41}$$

Having found $\ddot{s}_{min}$ and $\ddot{s}_{max}$, we have to transform these values back into the n-dimensional space. We therefore use the tangential path unit vector $\boldsymbol{r}_s$ and the normal acceleration $\boldsymbol{r}_{ss}\,\dot{s}^2$.

The minimal and maximal acceleration values in joint space, that are the intersections with the PT, are:

$$\ddot{\boldsymbol{q}}_{min,path} = \ddot{s}_{min}\,\boldsymbol{r}_s + \boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.42}$$

$$\ddot{\boldsymbol{q}}_{max,path} = \ddot{s}_{max}\,\boldsymbol{r}_s + \boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.43}$$

For a motion state $\boldsymbol{\Xi}$, these two values express for the current moment in time how much the robot could accelerate/decelerate on the current path while still staying on it.

The minimal and maximal path acceleration capabilities ($\ddot{\boldsymbol{q}}_{min,path}$ and $\ddot{\boldsymbol{q}}_{max,path}$) along the tangential path unit vector intersecting with the PT can be translated in absolute minimal and absolute maximal acceleration values in terms of the acceleration frame:

$$\text{for } \forall k \in (1 \ldots K): \tag{4.44}$$

$${}_k\ddot{q}_{min,abs} = \min({}_k\ddot{q}_{min,path},\ {}_k\ddot{q}_{max,path}) \tag{4.45}$$

$${}_k\ddot{q}_{max,abs} = \max({}_k\ddot{q}_{min,path},\ {}_k\ddot{q}_{max,path}) \tag{4.46}$$

4.1.4. Operational Space Path Dynamics

In the more general case, where $\boldsymbol{x} \neq \boldsymbol{q}$ and therefore $\boldsymbol{J} \neq \boldsymbol{1}$ and $\boldsymbol{H} \neq \boldsymbol{0}$, we can choose any description for $\boldsymbol{x}(\boldsymbol{q})$ to describe the path taken by the robot, most useful is the operational space description. In operational space the position trajectory in Eqn. (4.1) describes the end-effector frame position and/or frame orientation

$$\boldsymbol{x}(\boldsymbol{q}) = \boldsymbol{r}(s) \tag{4.47}$$

For the operational space representation, we use the non-simplified version of Eqn. (4.4) and Eqn. (4.6)

$$\dot{\boldsymbol{x}}(\boldsymbol{q}) = \boldsymbol{r}_s(s)\,\dot{s} \tag{4.48}$$

$$\ddot{\boldsymbol{x}}(\boldsymbol{q}) = \boldsymbol{r}_s(s)\,\ddot{s} + \boldsymbol{r}_{ss}(s)\,\dot{s}^2 \tag{4.49}$$

Remark 4.1.3 *We use again the short form for representing the derivations of $\boldsymbol{r}(s)$ with $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}$.*

We can now derive the maximum available operational space accelerations/decelerations for a path described by the variable $\ddot{s}$.

We substitute $\ddot{\boldsymbol{x}}$ in Eqn. (2.2) with Eqn. (4.49), which yields to:

$$\boldsymbol{\Lambda}(\boldsymbol{x})\left(\boldsymbol{r}_s\,\ddot{s} + \boldsymbol{r}_{ss}\,\dot{s}^2\right) = \boldsymbol{f} - \boldsymbol{p}(\boldsymbol{x}) - \mu(\boldsymbol{x}, \dot{\boldsymbol{x}}) \tag{4.50}$$

Leaving only the term containing $\ddot{s}$ on the left-hand side results in:

$$\boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{r}_s\,\ddot{s} = \boldsymbol{f} - \boldsymbol{p}(\boldsymbol{x}) - \mu(\boldsymbol{x}, \dot{\boldsymbol{x}}) - \boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.51}$$

Introducing the abbreviations $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$:

$$\tilde{\boldsymbol{\alpha}} := \boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{r}_s \tag{4.52}$$

$$\tilde{\boldsymbol{\beta}} := \boldsymbol{p}(\boldsymbol{x}) + \boldsymbol{\mu}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.53}$$

This leads to the abbreviated form of Eqn. (4.51):

$$\tilde{\boldsymbol{\alpha}}\,\ddot{s} = \boldsymbol{f} - \tilde{\boldsymbol{\beta}} \tag{4.54}$$

The variables $\boldsymbol{f}$, $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ all depend only on $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, $\dot{s}$, $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}$. $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$ is gained through a forward kinematics transformation of sensor inputs for $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ in the first instant the algorithm is run and after that, the analytical descriptions of the trajectory polynomials generated by the OTG are used. $\dot{s}$, $\boldsymbol{r}_s$, $\boldsymbol{r}_{ss}$ are then calculated in every time step using $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$. How these path variables are calculated is explained in Sec. 4.1.2.

We abbreviate the notation of every variable by leaving out its dependency to other variables. We can write Eqn. (4.54) for each individual DOF using the index k:

$${}_k\tilde{\alpha}\,\ddot{s} = {}_k f - {}_k\tilde{\beta} \tag{4.55}$$

Taking into account the torque capacities leads to individual inequalities for each joint/d.o.f.:

$${}_k f_{min} - {}_k\tilde{\beta} \leq {}_k\tilde{\alpha}\,\ddot{s} \leq {}_k f_{max} - {}_k\tilde{\beta} \tag{4.56}$$

Note that ${}_k\tilde{\alpha} \neq 0$, because the mass matrix $\boldsymbol{\Lambda}(\boldsymbol{x})$ is always positive definite and $\boldsymbol{r}_s$ is the non-zero path vector. Therefore, the Eqn. (4.56) can be written as:

$${}_k\tilde{l} \leq \ddot{s} \leq {}_k\tilde{h} \tag{4.57}$$

where

$${}_k\tilde{l} = \begin{cases} \frac{{}_k f_{min} - {}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha > 0 \\ \frac{{}_k f_{max} - {}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha < 0 \end{cases} \tag{4.58}$$

and

$${}_k\tilde{h} = \begin{cases} \frac{{}_k f_{max} - {}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha > 0 \\ \frac{{}_k f_{min} - {}_k\beta}{{}_k\alpha} & \text{for } {}_k\alpha < 0 \end{cases} \tag{4.59}$$

Equations (4.57) – (4.59) describe the range of the scalar path acceleration $\ddot{s}$ for which a single actuator can hold the manipulator on its operational space path without violating the k-th constraint. To fulfill all the constraints

$$\ddot{s} \in [\,{}_k\tilde{l},\ {}_k\tilde{h}\,] \text{ for } \forall k \tag{4.60}$$

has to hold true. Only if the absolute velocity is too high, this constraint will not be fulfilled.

It follows that an admissible acceleration is any tangential acceleration $\ddot{s}$ that does not violate Eqn. (4.56) for $\forall k$, that is:

$$\ddot{s}_{min} \leq \ddot{s} \leq \ddot{s}_{max} \tag{4.61}$$

where

$$\ddot{s}_{min} = \max_k\ {}_k\tilde{l} \tag{4.62}$$

and

$$\ddot{s}_{max} = \min_k\ {}_k\tilde{h} \tag{4.63}$$

Having found $\ddot{s}_{min}$ and $\ddot{s}_{max}$, we have to transform these values back into the n-dimensional space. We therefore use the tangential path unit vector $\boldsymbol{r}_s$ and the normal acceleration $\boldsymbol{r}_{ss}\,\dot{s}^2$.

The real minimum and maximum acceleration values in operational space, that are describing the intersecting points with the parallelotope, are:

$$\ddot{\boldsymbol{x}}_{min,path} = \ddot{s}_{min}\,\boldsymbol{r}_s + \boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.64}$$

$$\ddot{\boldsymbol{x}}_{max,path} = \ddot{s}_{max}\,\boldsymbol{r}_s + \boldsymbol{r}_{ss}\,\dot{s}^2 \tag{4.65}$$

The minimal and maximal path acceleration capabilities ($\ddot{\boldsymbol{x}}_{min,path}$ and $\ddot{\boldsymbol{x}}_{max,path}$) along the tangential path unit vector intersecting with the PT can be translated in absolute minimal and absolute maximal acceleration values in terms of the acceleration frame:

$$\text{for } \forall k \in (1 \dots K): \tag{4.66}$$

$$_k\ddot{x}_{min,abs} = \min(\,_k\ddot{x}_{min,path},\,_k\ddot{x}_{max,path}) \tag{4.67}$$

$$_k\ddot{x}_{max,abs} = \max(\,_k\ddot{x}_{min,path},\,_k\ddot{x}_{max,path}) \tag{4.68}$$

4.2. OTG's Limited Interpretation of Constraints

In the last section we derived the minimal and maximal acceleration capabilities on a path and then transformed them to absolute minimal and maximal accelerations ($\ddot{\boldsymbol{q}}_{min,abs}$ / $\ddot{\boldsymbol{q}}_{max,abs}$ in Eqn. (4.45) and (4.46) or $\ddot{\boldsymbol{x}}_{min,abs}$ / $\ddot{\boldsymbol{x}}_{max,abs}$ in Eqn. (4.67) and (4.68)). These are not useful for the OTG yet. The OTG only accepts a negative minimal acceleration and a positive maximal acceleration constraint as input, the derived absolute extremal accelerations do not always fit into this limitation.

This limitation has to do with the planning character of the OTG algorithm. In order to plan trajectories for every axis from start to end, there has to be a period of acceleration and a period of deceleration. The OTG Algorithm can not take into account that the acceleration constraints will become more or less in the future. The OTG needs to know inside which acceleration boundaries he can accelerate and decelerate, even tough in the current motion state only one of both is needed to move on to the next time step.

To visualize this problem for various scenarios, I will give three 2D examples that all show the limitations of using the absolute minimal and maximal accelerations as input for the OTG. Visualizing the problem in 2D is best for explaining the concept, but the problem is the same for higher dimensions. All the visualizations in the following are based on the path dynamics equations explained in the last section.

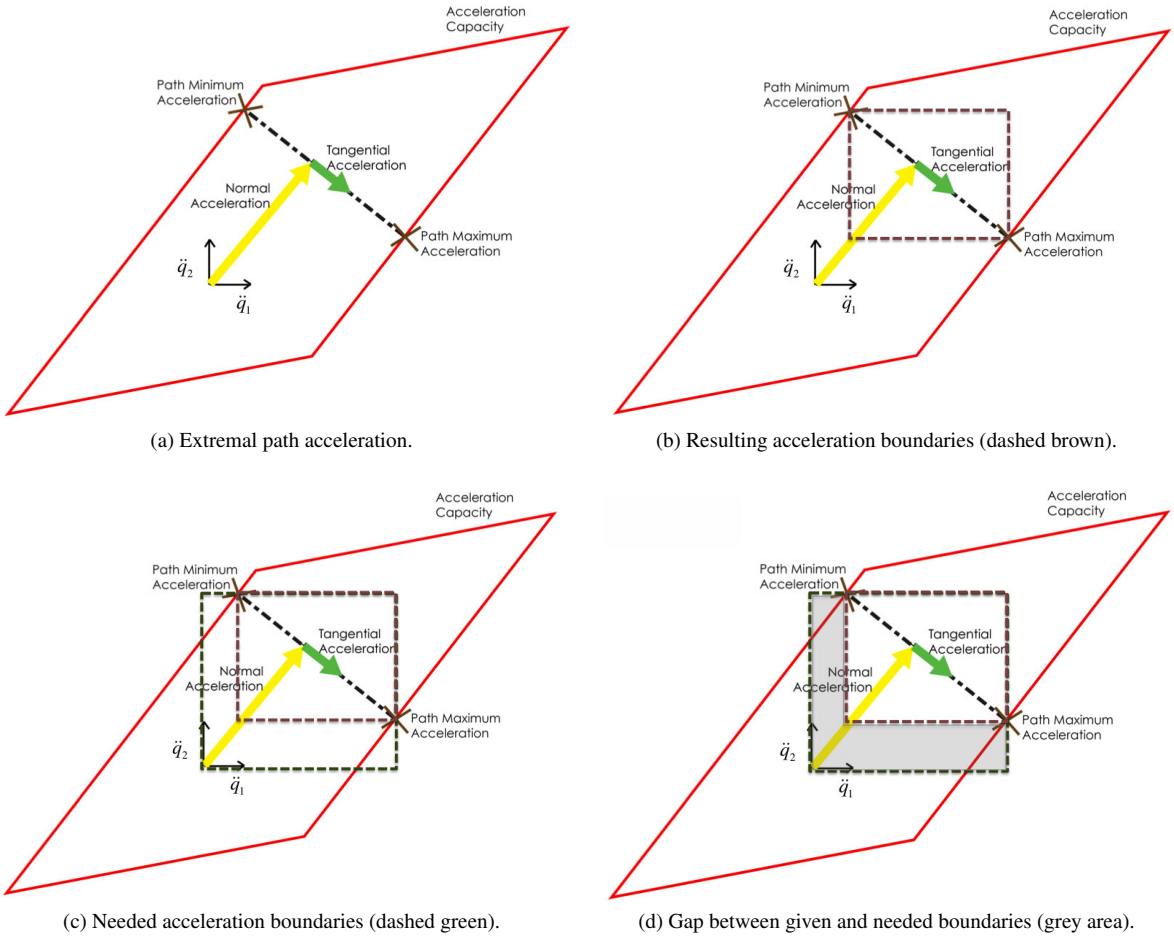
Remark 4.2.1 *In the following, the accelerations are described in joint space with $\ddot{\boldsymbol{q}}$, but in can be applied one-to-one to operational space using $\ddot{\boldsymbol{x}}$.*

4.2.1. First Example

The first example is shown in Fig. 4.2. In the first Subfig. 4.2a of it, the red parallelogram describes the acceleration capacity. The point of zero acceleration is the origin of the acceleration frame ($\ddot{\boldsymbol{q}}_1/\ddot{\boldsymbol{q}}_2$). The yellow arrow stands for the normal acceleration, the green arrow describes the tangential acceleration. Both combined lead to the actual acceleration, this vector is not shown in the figure. The tangential acceleration is prolonged in both directions till the line intersects with the boundary. The minimal and maximal accelerations ($\ddot{\boldsymbol{q}}_{min,path}/\ddot{\boldsymbol{q}}_{max,path}$) are found. In a 3D or even higher dimensional case, the prolonged (hyper)line would intersect with two of the (hyper)planes of the PT.

In Subfig. 4.2b is shown, how the absolute acceleration limits $\ddot{\boldsymbol{q}}_{min,abs}$ and $\ddot{\boldsymbol{q}}_{max,abs}$ form a boundary rectangle, shown with a brown dashed line. The rectangle does not include the origin, the found lower and upper boundaries are all positive in its value for both axis. The value can not be used as acceleration constraint input to the OTG.

Both of the lower limits of the given acceleration boundaries would need to be extended in the negative direction in order to at least include the origin of the acceleration frame. This *at least needed* boundary is shown as a green dashed rectangle in Subfig. 4.2c. The discrepancy between what is given and what is needed as a boundary is shown as a grey area in Subfig. 4.2d.



(a) Extremal path acceleration.

(b) Resulting acceleration boundaries (dashed brown).

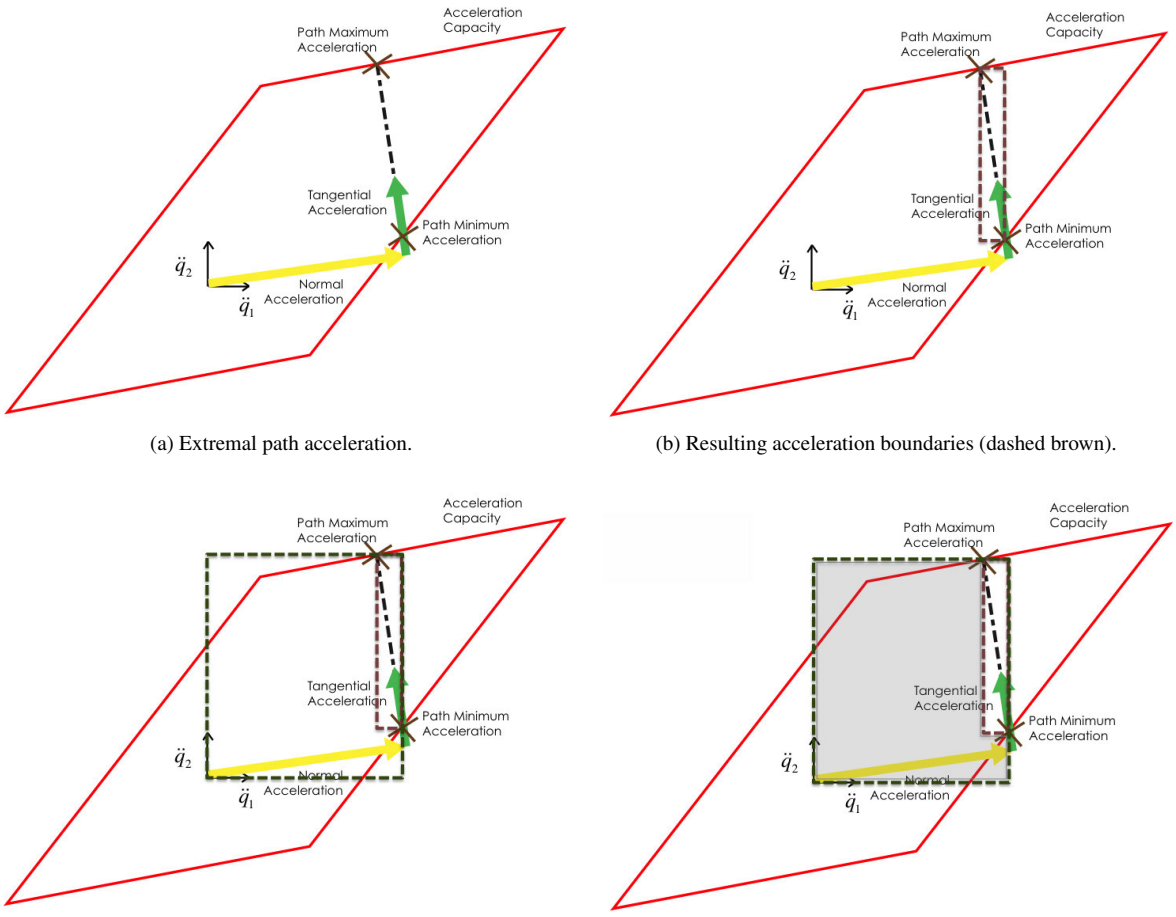
(c) Needed acceleration boundaries (dashed green).

(d) Gap between given and needed boundaries (grey area).

Figure 4.2.: Example 1: Acceleration boundaries for a certain path instant.

Even if we would use the dashed green rectangle, we would not be able to design an efficient enhancement of the OTG algorithm. The complete green rectangle only consists of *acceleration* capabilities, no *deceleration* capabilities are given. The OTG plans ahead to the target motion state when constructing the trajectories, every acceleration towards the target needs a deceleration at the end to arrive with the intended velocity and zero acceleration. With only a very small deceleration capability, the execution time gets very long and is not optimal at all. Just the extension of the given boundary to the at least needed boundary won't do it.

4.2.2. Second Example



(a) Extremal path acceleration.

(b) Resulting acceleration boundaries (dashed brown).

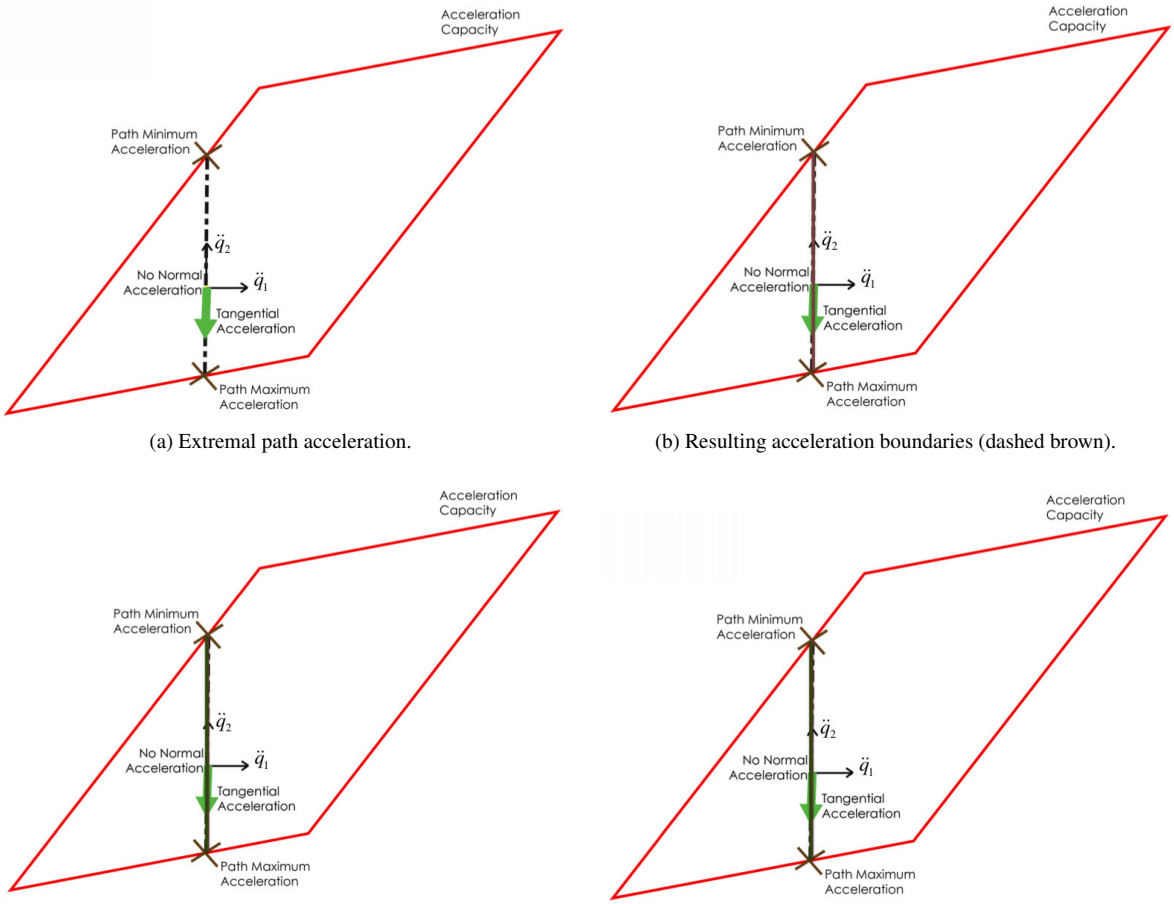
(c) Needed acceleration boundaries (dashed green).

(d) Gap between given and needed boundaries (grey area).

Figure 4.3.: Example 2: Acceleration boundaries for certain path instant.

While moving along the path, the tangential acceleration vector is always changing its direction, the second example is showing in Fig. 4.3 such a new path acceleration state. Like in the first example, the top-left Subfig. 4.3a shows the minimal and maximal accelerations $\ddot{\boldsymbol{q}}_{min,path}$ and $\ddot{\boldsymbol{q}}_{max,path}$.

The tangential acceleration vector is almost aligned with the $\ddot{\boldsymbol{q}}_2$-axis-vector. This leads to a relatively small width of the rectangle. This is shown in Subfig. 4.3b. The discrepancy to what is minimally needed as a boundary is even larger than in the first example. This is demonstrated in Subfig. 4.3c and Subfig. 4.3d.



(a) Extremal path acceleration.

(b) Resulting acceleration boundaries (dashed brown).

(c) Needed acceleration boundaries (dashed green).

(d) Gap given and needed boundaries not existing.

Figure 4.4.: Example 3: Acceleration boundaries for certain path instant.

4.2.3. Third Example

If the tangential acceleration vector is fully aligned with one of the acceleration axis, we get an *ill-conditioned* boundary. This is shown in the third example in Fig. 4.4. The Subfig. 4.4a shows an acceleration vector, which has only a tangential acceleration component. This is the case, when the current velocity is zero. The brown rectangle showing the given boundaries is actually almost only a line, as you can see in Subfig. 4.4b.

The rectangle has the origin inside, so that the needed boundaries are not larger than the given ones. The green rectangle in Subfig. 4.4c is therefore also only a line. The grey area is not existing at all, as shown in Subfig. 4.4d.

This third example shows even more, that only extending the given boundaries in such a way that the origin is included, won't output acceleration constraints useful for the OTG. A large proportion of the PT is unused, only because the current tangential acceleration vector is aligned to one of the axis.

An algorithm is needed that can also take into account more of the "potentially" available acceleration capabilities of the PT.

4.3. Path Dynamics Merging Algorithm

In the following is shown how the limitation of the OTG (see Sec.4.2) in using the acceleration capabilities calculated through the path dynamics approach of Sec. 4.1 is solved.

The merging algorithm is capable of taking into account more of the available acceleration capabilities of the PT, even tough they are not needed for the current motion state. Before we explain how that algorithm works, an introductory example related to Sec. 4.2.1 is given to explain the concept behind it. The complete algorithm is then described in Sec. 4.3.2.

4.3.1. Introductory Example

Subfigure 4.2b shows the given acceleration boundaries, which only allow for positive acceleration, because the lower limits on both axes have a positive value. Extending the axes starting from the origin to the PT results in two intersections per axis. We call these the "axes acceleration limits". This is visualized in Subfig. 4.5b with the dotted purple line.

In the next step, we compare the tangential acceleration values with the axis acceleration values and generate the finally used acceleration constraints. *The key principle used herefore is the fact that we only need to limit the minimal and maximal tangential acceleration points by it's largest axis values. For the other values, the axis acceleration values can be used.* No matter how many DOFs are used, this principle can always be applied. The merged boundary of the first example can be seen as the dotted green box in Subfig. 4.5c and the gain in potentially usable acceleration capability is shown by the grey box in Subfig. 4.5d.

4.3.2. Complete Algorithm Outline

The following steps outline the complete Path Dynamics Merging algorithm. Where needed, references are made to previous and following sections of this thesis:

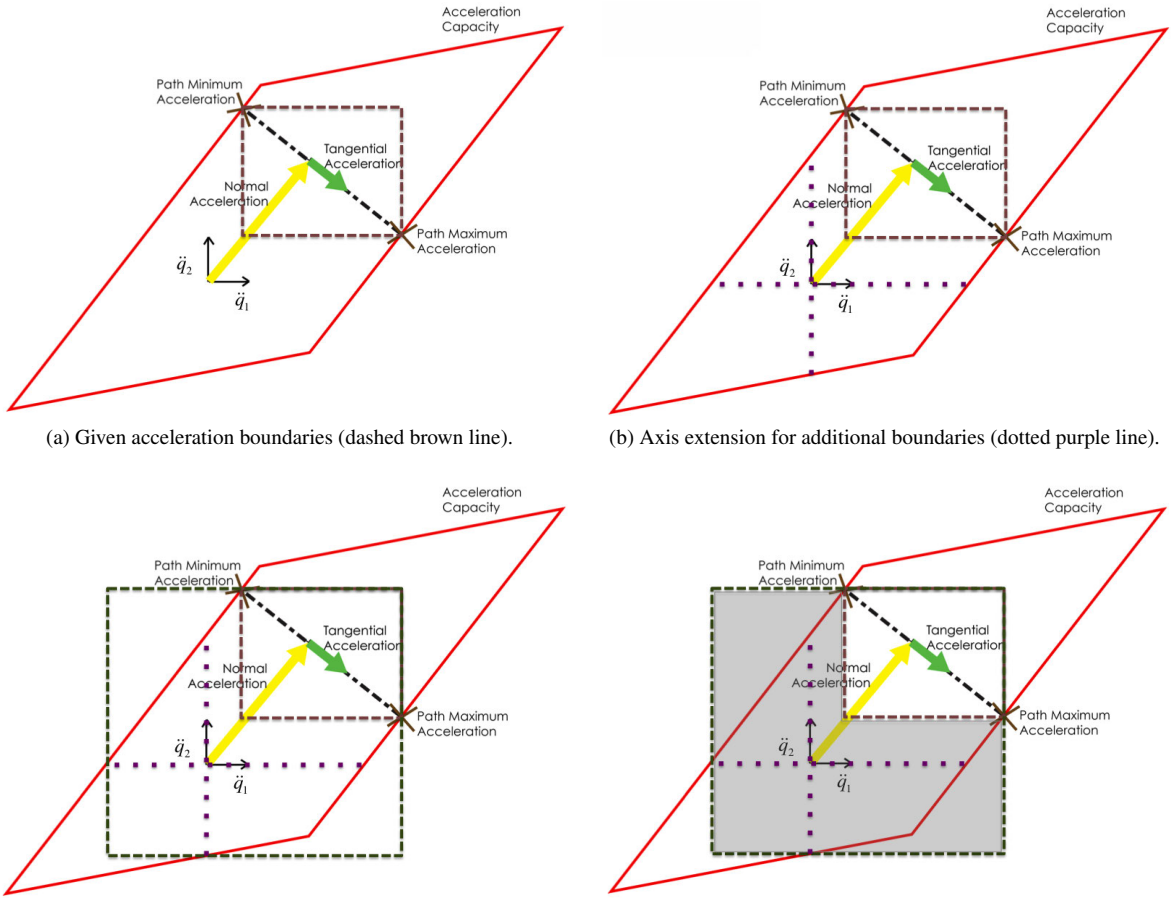
a Calculate the path derivations $\dot{s}$, $\ddot{s}$, $\boldsymbol{r}_s$, $\boldsymbol{r}_{ss}$ according to Sec. 4.1.2. As input, only the current motion state $\boldsymbol{\Xi}$ is needed.

b Only if $\dot{\boldsymbol{q}} = \boldsymbol{0} \wedge \ddot{\boldsymbol{q}} = \boldsymbol{0}$:

- then we only calculate the extreme possible accelerations by finding the neg. and pos. boundaries on every axis, this is explained in Sec. 4.3.3.

c Otherwise:

- We calculate the path acceleration constraints with the output: $\ddot{s}_{min}$, $\ddot{s}_{max}$ and map them to multiple dimensions to get the acceleration boundary boxes. Output: $\ddot{\boldsymbol{q}}_{min,abs}$, $\ddot{\boldsymbol{q}}_{max,abs}$. This is done according to Sec. 4.1.3 for joint-space or according to Sec. 4.1.4 for operational space.



(a) Given acceleration boundaries (dashed brown line).

(b) Axis extension for additional boundaries (dotted purple line).

(c) Merged acceleration boundaries (dotted green line).

(d) Gap betwen given and merged boundaries (grey area).

Figure 4.5.: Example 1: Application of the Merging Algorithm.

- We calculate axes acceleration limits with the output: $\ddot{\boldsymbol{q}}_{min,axes}$, $\ddot{\boldsymbol{q}}_{max,axes}$ according to Sec. 4.3.3.
- We compare and merge $\ddot{\boldsymbol{q}}_{min,abs}$, $\ddot{\boldsymbol{q}}_{max,abs}$ with $\ddot{\boldsymbol{q}}_{min,axes}$, $\ddot{\boldsymbol{q}}_{max,axes}$ and get the output $\ddot{\boldsymbol{q}}_{min}$, $\ddot{\boldsymbol{q}}_{max}$. This is done according to Sec. 4.3.4.

the optimized acceleration constraints $\ddot{\boldsymbol{q}}_{min}$, $\ddot{\boldsymbol{q}}_{max}$ can now be used as input for the OTG.

4.3.3. Axes Acceleration Limits

Joint Space Version

In the next step, we will consider the case that we extend each coordinate axis in negative and positive direction until we intersect with the boundaries. To describe an axis vector with variable scalar value, we describe it with

$$\boldsymbol{\nu}_l = \boldsymbol{e}_l \, \nu_l \tag{4.69}$$

where $\boldsymbol{e}_l$ is a unit vector along each axis with $l \in (1,\ldots,K)$. ν_l describes the scalar value of the vector along axis l. We substitute the acceleration $\ddot{\boldsymbol{q}}$ in Eqn. (2.1) with the axis vector $\boldsymbol{\nu}_l$, which yields to:

$$\boldsymbol{M}(\boldsymbol{q})\, \boldsymbol{e}_l \, \nu_l = \boldsymbol{\tau} \, - \, \boldsymbol{\tau}_g(\boldsymbol{q}) \, - \, c(\boldsymbol{q},\dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}} \tag{4.70}$$

We then introduce the abbreviations $\hat{\boldsymbol{\alpha}}_l$ and $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\alpha}}_l \;\; := \;\; \boldsymbol{M}(\boldsymbol{q})\, \boldsymbol{e}_l \quad \text{for} \quad \forall l \in (1,\ldots,K) \tag{4.71}$$

$$\hat{\boldsymbol{\beta}} \;\; := \;\; \boldsymbol{\tau}_g(\boldsymbol{q}) \, + \, c(\boldsymbol{q},\dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}} \tag{4.72}$$

This leads to the abbreviated form of Eqn. (4.70):

$$\hat{\boldsymbol{\alpha}}_l \; \nu_l \;\; = \;\; \boldsymbol{\tau} \, - \, \hat{\boldsymbol{\beta}} \tag{4.73}$$

We can write Eqn. (4.73) for each individual DOF using the index k:

$${}_k\hat{\alpha}_l \; \nu_l \;\; = \;\; {}_k\tau \, - \, {}_k\hat{\beta} \tag{4.74}$$

Taking into account the torque capabilities leads to individual inequalities for every DOF:

$${}_k\tau_{min} - \, {}_k\hat{\beta} \leq \, {}_k\hat{\alpha}_l \; \nu_l \leq \, {}_k\tau_{max} - \, {}_k\hat{\beta} \tag{4.75}$$

Note that $\hat{\boldsymbol{\alpha}}_l \neq \boldsymbol{0}$, because the mass matrix $\boldsymbol{M}(\boldsymbol{q})$ is always positive definite and $\boldsymbol{e}_l$ is a non-zero unit vector. Nevertheless, part of the coordinates of $\hat{\boldsymbol{\alpha}}_l$ can still become zero. The Eqn. (4.75) can be written as:

$${}_k\hat{l}_l \leq \, \nu_l \leq \, {}_k\hat{h}_l \tag{4.76}$$

where

$${}_k\hat{l}_l = \begin{cases} \frac{{}_k\tau_{min} - \, {}_k\hat{\beta}}{{}_k\hat{\alpha}_l} & \text{for } {}_k\hat{\alpha}_l > 0 \\ \frac{{}_k\tau_{max} - \, {}_k\hat{\beta}}{{}_k\hat{\alpha}_l} & \text{for } {}_k\hat{\alpha}_l < 0 \\ -\infty & \text{for } {}_k\hat{\alpha}_l = 0 \end{cases} \tag{4.77}$$

and

$${}_k\hat{h}_l = \begin{cases} \frac{{}_k\tau_{max} - \, {}_k\hat{\beta}}{{}_k\hat{\alpha}_l} & \text{for } {}_k\hat{\alpha}_l > 0 \\ \frac{{}_k\tau_{min} - \, {}_k\hat{\beta}}{{}_k\hat{\alpha}_l} & \text{for } {}_k\hat{\alpha}_l < 0 \\ \infty & \text{for } {}_k\hat{\alpha}_l = 0 \end{cases} \tag{4.78}$$

For at least one $k \in (1,\ldots,K)$, ${}_k\hat{l}_l \neq -\infty$ and for at least one $k \in (1,\ldots,K)$, ${}_k\hat{h}_l \neq \infty$, because $\hat{\boldsymbol{\alpha}}_l \neq \boldsymbol{0}$.

Equations (4.76) $\sim$ (4.78) describe the range of acceleration values v_l along axis l for which a single actuator can hold the manipulator on its joint path without violating the k-th constraint. To fulfill all the constraints

$$v_l \in [\ {}_k\hat{l}_l,\ {}_k\hat{h}_l\ ] \text{ for } \forall k \tag{4.79}$$

has to be fulfilled. Assuming that the actuator forces are high enough to compensate the gravity forces in a low velocity motion state, only if the absolute velocity $|\dot{\boldsymbol{q}}|$ is too high, the constraints are not fulfilled.

It follows that an extremal acceleration along axis l is any value for v_l that does not violate Eqn. (4.75) for $\forall k$, that means:

$$v_{l,min} \leq\ v_l \leq\ v_{l,max} \tag{4.80}$$

where

$$v_{l,min} = \max_k\ {}_k\hat{l}_l \tag{4.81}$$

and

$$v_{l,max} = \min_k\ {}_k\hat{h}_l \tag{4.82}$$

$v_{l,min}$ and $v_{l,max}$ are transformed back into the n-dimensional space. We use the axis unit vector $\boldsymbol{e}_l$ defined in Eqn. (4.69) to do so. This yields the extremal acceleration constraints for axis l:

$$\boldsymbol{\nu}_{l,min} = v_{l,min}\boldsymbol{e}_l \tag{4.83}$$

$$\boldsymbol{\nu}_{l,max} = v_{l,max}\boldsymbol{e}_l \tag{4.84}$$

We repeat the calculations in Eqn. (4.71) – (4.84) for $\forall l \in (1,\ldots,K)$.

Operational Space Version

For operational space, it is the same procedure as for joint-space, only the underlying dynamics equations are different. We substitute the acceleration $\ddot{\boldsymbol{x}}$ in Eqn. (2.2) with the axis vector $\boldsymbol{\nu}_l$, which yields to:

$$\boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{e}_l\,v = \boldsymbol{f}\ -\ \boldsymbol{p}(\boldsymbol{x})\ -\ \boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}}) \tag{4.85}$$

The abbreviations $\hat{\boldsymbol{\alpha}}_l$ and $\hat{\boldsymbol{\beta}}$ are redefined with:

$$\hat{\boldsymbol{\alpha}}_l\ :=\ \boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{e}_l \quad \text{for} \quad \forall l \in (1,\ldots,K) \tag{4.86}$$

$$\hat{\boldsymbol{\beta}}\ :=\ \boldsymbol{p}(\boldsymbol{x})\ +\ \boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}}) \tag{4.87}$$

The rest is analogue to the joint-space version, except for substituting $\boldsymbol{\tau}$ with $\boldsymbol{f}$:

$$\hat{\boldsymbol{\alpha}}_l\ v_l\ =\ \boldsymbol{f}\ -\ \hat{\boldsymbol{\beta}} \tag{4.88}$$

4.3.4. Compare and Merge

We take the two previously found minimal and maximal path accelerations and compare them with each other. We consider these two path acceleration vectors as "points" in the acceleration coordinate frame. Only one coordinate of each point needs to be used in order to restrict the extremal path accelerations described by that point. It seems most appropriate to choose the largest coordinates of each points, but this does not always work as shown in the following detailed algorithm description:

Remark 4.3.1 *Notation in the following listings: Variables inside round brackets are parameters of the preceding function, variables inside square brackets describe how to reorder/rank the coordinates of the preceding variable.*

Preparation Step

Before we can compare the two points with each other, we need to find the absolute values of their coordinates, sort and rank their coordinates in descending order and create a list which resembles the signs of the coordinates. The algorithm for this is shown in list. 4.1.

```
// Take component-wise the absolute values
absMin = cwiseAbs(qddMinPath);
absMax = cwiseAbs(qddMaxPath);

// Sort the coordinate values from high to low and then rank the sorted
    values in relation to the order of the original values
rankMin = rankDescending(absMin);
rankMax = rankDescending(absMax);

// Use the ranking to reorder the original values in a list from highest to
    lowest
sortMin= qddMinPath[rankMin];
sortMax= qddMaxPath[rankMax];

// Create a list with signs of the sorted list: a negative value results in
    '0', a positive value results in '1'
signSortMin = sign(sortMin);
signSortMax = sign(sortMax);
```

Listing 4.1: Preparation step: Sort, rank and find signs of the extremal path accelerations

Comparision Step

For the case, that both points have on the same coordinate axis with the same sign their largest value, we store the larger one as a constraint on that axis and take a further look into the point, which was smaller on that coordinate. Let's call the larger point A and the smaller point B. For B, we look at its second largest coordinate and compare it with the same coordinate axis of A. If both points have on that axis a value with the same sign and A is larger B, we look at the third largest coordinate of B in the same way again. We repeat this till they have either not the same sign or till they have the same sign and B is larger than A. Given this case occurs before all the coordinates have been checked through, we use this found value of point B as the second constraint. If it does not occur after checking all coordinates, the point B is not used at all. The algorithm of this is shown in list. 4.2.

```
// Define to start with the largest values of the sorted lists
mi = 0;
ma = 0;

// Compare the results of the two extremal path accelerations with each other
// Check if the values have the same sign and are on the same axis
if(signSortMin[mi] == signSortMax[ma]
   && rankMin[mi] == rankMax[ma] ) {
   // compare by size their absolute value
   if(absMin[rankMin[mi]] < absMax[rankMax[ma]]){
      // absMin coordinate on that axis is smaller
      // therefore choose the second largest coordinate of absMin
      mi++;
      while (mi < dof) {
         // compare with the other acceleration
         if(absMin[rankMin[mi]] < absMax[rankMin[mi]]
            && signSortMin[mi] == sign(qddMaxPath[rankMin[mi]]) ){
            // it is smaller, check the next value
            mi++;
         }
         else{
         break;
         };
      };
   }
   else{
      // absMax coordinate on that axis has a smaller size
      // therefore choose the second largest coordinate of absMax
      ma++;
      while(ma < dof){
         // compare with the other acceleration
         if(absMax[rankMax[ma]] < qddMinPathAbs[rankMax[ma]]
           && signSortMax[ma] == sign(qddMinPath[rankMax[ma]]) ){
            // it is smaller, check the next value
            ma++;
         }
         else {
            break;
         };
      };
   };
};
```

Listing 4.2: Compare extremal path acceleration points

Merging Step

After having found the one or two coordinate constraints, we merge them with the axis acceleration values. How to find the axis acceleration limits is explained in Sec. 4.3.3. These axis accelerations limits are called $\nu_{l,min}$ and $\nu_{l,max}$, we join them to a matrix with two rows and K columns and call it “qddAxis”. The found integer values for “mi” and “ma” allows to substitute the found coordinate constraints in the “qddAxis” matrix. For the remaining coordinates of the sorted points list, if one of them is larger than the according coordinate value in the matrix “qddAxis”, we substitute the value. This is done to ensure, that the merged boundaries always encompass the given extremal path acceleration points. The algorithm is described in list. 4.3.

```
// Substitute the two coordinate axis values in the matrix qddAxis, which
    need to be redefined in order to limit qddMinPath and qddMaxPath with one
    coordinate each
if (mi != dof){
   qddAxis( signSortMin[mi] , rankMin[mi] ) = sortMin[mi];
};
if (ma != dof){
   qddAxis( signSortMax[ma] , rankMax[ma]) = sortMax[ma];
};

// Check for the remaining axes, which value is larger and adjust it if
    necessary
while(mi < dof){
   if(qddMinPathAbs[ rankMin[mi] ] > absolute(qddAxis( signSortMin[mi] ,
       rankMin[mi]))) {
      qddAxis( signSortMin[mi] , rankMin[mi] ) = sortMin[mi];
   };
   mi++;
};
while(ma < dof){
   if(absMax[ rankMax[ma] ] > absolute(qddAxis( signSortMax[ma] , rankMax[ma
       ]) ) ) {
      qddAxis( signSortMax[ma] , rankMax[ma] ) = sortMax[ma];
   };
   ma++;
};

// set the acceleration constraints by splitting up again the matrix qddAxis
    into two vectors
qddMin = getFirstRow(qddAxis);
qddMax = getSecondRow(qddAxis);
```

Listing 4.3: Merging Step: path accelerations merged with axis acceleration limits

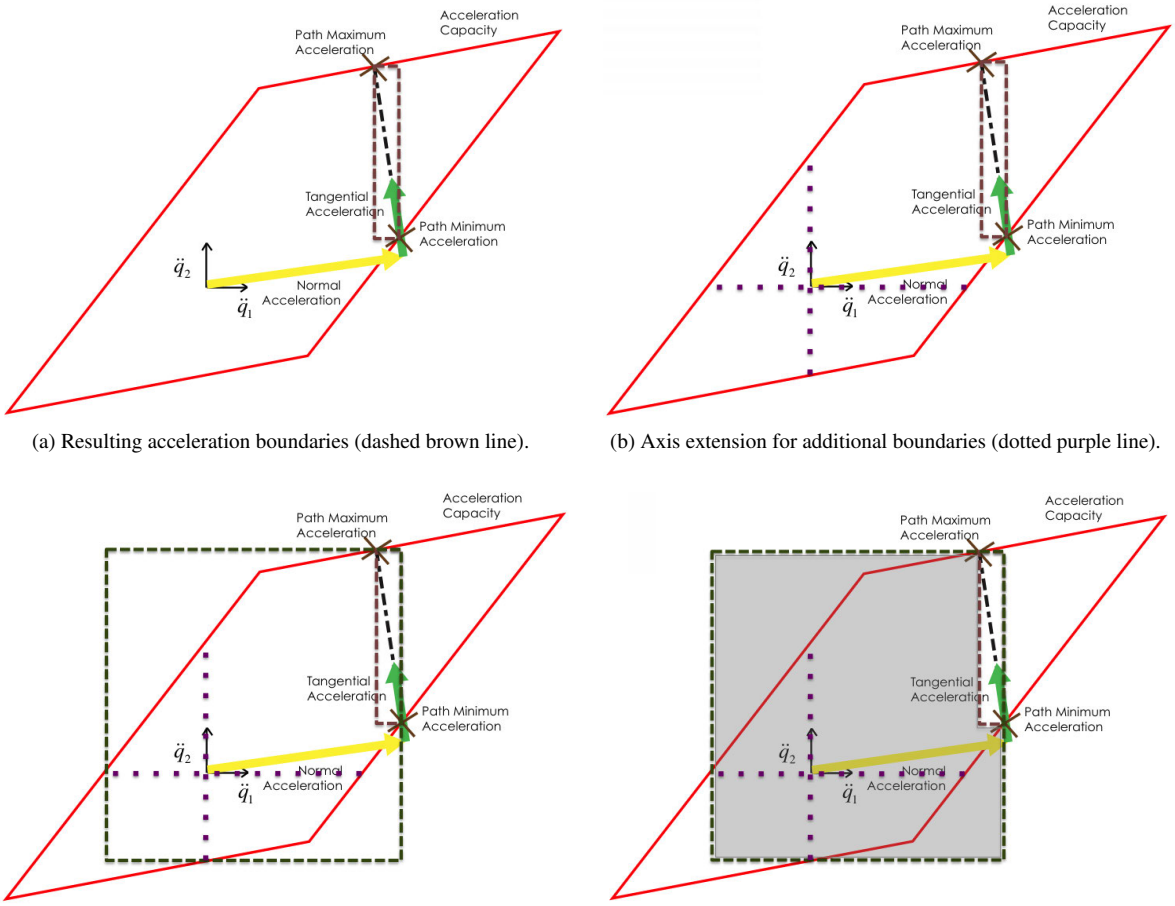
The values *qddMin* and *qddMax* are the final acceleration constraints useful as input values for the OTG algorithm.

4.3.5. Two Further Path Dynamics Merging Algorithm Examples

The merging boundaries algorithm processes the acceleration capabilities so that as a result we gain more useful acceleration constraints as input for the OTG. For the exemplary problem given in Sec. 4.2.1, the positive effect of the merging boundary algorithm was already given in the introductory example of Sec. 4.3.1. For the other two exemplary problems of Sec. 4.2.2 and 4.2.3, the effect and result of the merging algorithm is given in Fig. 4.6 and Fig. 4.7.

Figure 4.6 shows how the current acceleration state resembles a line in the outer corner of a parallelotope. The resulting acceleration capabilities are so limited in its range that you can not use them as a acceleration constraint input for the OTG. Nevertheless, the Path Dynamics Merging algorithm extends all axes to get additional acceleration values and allows the complete grey area shown in Subfig. 4.6d to be used as an OTG input.

Figure 4.7 further demonstrates that if the current acceleration resembles a line aligned to one of the axis, without the Path Dynamics Merging algorithm, no useful values could be obtained.



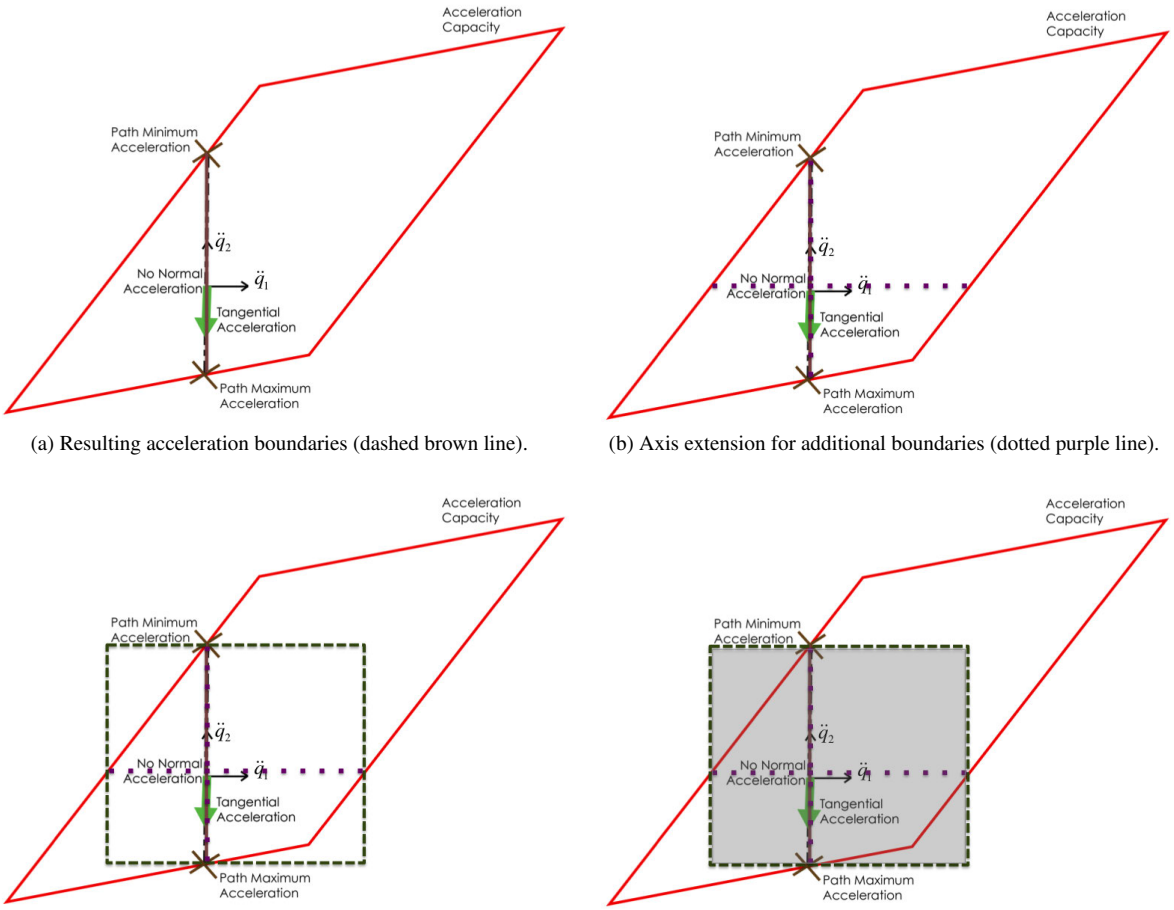
(a) Resulting acceleration boundaries (dashed brown line).

(b) Axis extension for additional boundaries (dotted purple line).

(c) Merged acceleration boundaries (dotted green line).

(d) Gap betwen given and merged boundaries (grey area).

Figure 4.6.: Example 2: Application of the Merging Algorithm.



(a) Resulting acceleration boundaries (dashed brown line).

(b) Axis extension for additional boundaries (dotted purple line).

(c) Merged acceleration boundaries (dotted green line).

(d) Gap betwen given and merged boundaries (grey area).

Figure 4.7.: Example 3: Application of the Merging Algorithm.

4.4. Summary - Acceleration Capabilities on a Path

In Sec. 4.1 has been shown how we calculate the robot dynamics along a motion path, that means, how much acceleration capability is available for any given motion state. The Fig. 4.1 introduces the concept of splitting the acceleration into a tangential and normal portion. How this is then applied to splitting the acceleration capabilities in joint and operational space is shown mathematically in Sec. 4.1.3 and Sec. 4.1.4.

In Sec. 4.2 the input limitations of the OTG are explained with its consequence for the development of the DOTG. Three geometry examples in Fig. 4.2, 4.3 and4.4 are given in order to show why the acceleration capabilities calculated with the Path Dynamics method can not directly be used as acceleration constraints for the OTG algorithm.

In the last part of this Chapter, Sec. 4.3 introduces an algorithm which is able to merge the results of the acceleration capabilities along a path into proper acceleration constraints that can always be used with the OTG. The basic structure and main overview for this algorithm is given in Sec. 4.3.2 and its application to the three geometry examples is given in Fig. 4.5, 4.6 and 4.7.

5. Dynamic Online Trajectory Generation

The acceleration constraints $\{\dot{\boldsymbol{q}}_{i,min}, \dot{\boldsymbol{q}}_{i,max}\}$ used by the OTG were so far conservatively estimated and defined for a given robot before the system was actually run. A first approach to improve this would be the following:

The moment when a new target state of motion $\boldsymbol{\Xi}_{i,trgt}$ is chosen, calculate the acceleration constraints $\ddot{\boldsymbol{q}}_{min}$, $\ddot{\boldsymbol{q}}_{max}$ for the current motion state $\boldsymbol{\Xi}_i$ according to Sec. 4.3.2. Then use only a small fraction of these constraints values as constant input to the OTG. This "safety factor" ensures that the acceleration trajectory generated by the OTG will always be executable without tracking-errors, even if the motion goes through a robot configuration which has only relatively small acceleration capabilities. This is obviously not an optimal solution, nevertheless a better approach compared to what has been done before.

We do not settle with this rather pragmatic approach, instead provide new algorithms that consider the dynamics throughout the motion. In the following will be shown how we use the results of Ch. 3 and Ch. 4 for the calculation of acceleration constraints and combine them with the two versions of the OTG. We first provide two approaches for the velocity-based OTG (cf. sec 2.3.4), which we call *VDOTG*. Later in this chapter, an approach for the position-based OTG (cf. sec 2.3.4), the so-called *PDOTG*, is given.

5.1. Velocity-Based Dynamic OTG - First Approach

An unforeseen event happens at t_i, and the system is in a motion state $\boldsymbol{\Xi}_i$ from which we have to generate a new trajectory in order to reach a user-given value

$$\boldsymbol{\Xi}_{i,trgt} = \{\dot{\boldsymbol{q}}_{i,trgt}, \ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}\} \tag{5.1}$$

in the shortest possible time. The OTG version used is the *target velocity-based OTG Type IV algorithm* (cf. sec 2.3.4). For this version of the OTG, the target position $\boldsymbol{q}_{i,trgt}$ can not be defined, the OTG will bring the current velocity as fast as possible to the target velocity by choosing the shortest possible position trajectory to the target velocity state. This version is useful if the system has to do an emergency brake or if it has to get to a specific velocity as fast as possible when approaching a workspace.

The new algorithm will do an optimization along an initially calculated motion path using the acceleration constraints for that path. The path is computed by an initial run of the OTG algorithm. This algorithm will not necessarily find the global optimum for the dynamic behavior, but will at least try to optimize it in real-time along the defined path.

5.1.1. Algorithm

The algorithmic procedure for the first approach of the VDOTG is detailed in the following:

1. Compute new values for the minimum and maximum accelerations $\ddot{\boldsymbol{q}}_{0,min}$, $\ddot{\boldsymbol{q}}_{0,max}$ according to Sec. 4.3.2. The values of $\boldsymbol{q}_0$ and $\dot{\boldsymbol{q}}_0$ are taken from the current state of motion $\boldsymbol{\Xi}_0$.
2. *Optional:* $\forall\, k \in \{1, \ldots, K\}$: $\begin{cases} \text{If } {}_k\ddot{q}_0 > {}_k\ddot{q}_{0,max} \quad , & \text{then } {}_k\ddot{q}_0 = {}_k\ddot{q}_{0,max} \\ \text{If } {}_k\ddot{q}_0 < {}_k\ddot{q}_{0,min} \quad , & \text{then } {}_k\ddot{q}_0 = {}_k\ddot{q}_{0,min} \end{cases}$
3. Call the Type IV velocity-based OTG algorithm. This will provide a trajectory (incl. path) for all K DOFs to reach $\dot{\boldsymbol{q}}_{0,trgt}$ and $\ddot{\boldsymbol{q}}_{0,trgt} = \boldsymbol{0}$ at the same time instant.
   - Input values: $\boldsymbol{\Xi}_0$, $\ddot{\boldsymbol{q}}_{0,min}$, $\ddot{\boldsymbol{q}}_{0,max}$, $\dddot{\boldsymbol{q}}_{0,min}$, $\dddot{\boldsymbol{q}}_{0,max}$, $\dot{\boldsymbol{q}}_{0,trgt}$.
   - Output values:

– New state of motion

$$\Xi_1 = (\boldsymbol{q}_1, \dot{\boldsymbol{q}}_1, \ddot{\boldsymbol{q}}_1) \tag{5.2}$$

– Trajectory

$$\Phi_0(t) = \left\{ \left({}^1\xi_0(t), {}^1\gamma_0\right), \ldots, \left({}^l\xi_0(t), {}^l\gamma_0\right), \ldots, \left({}^L\xi_0(t), {}^L\gamma_0\right) \right\} \tag{5.3}$$

– The index of the DOF that determined the execution time of the trajectory, defined as $\kappa \in \{1, \ldots, K\}$.

If the trajectory is executed on a robot system, the computed state of motion Ξ_1 will be send to the controllers.

4. Using $\Phi_0(t)$, find for the DOF κ the maxima and minima in every polynomial section. How this is done is described in Sec. 5.1.3.

5. Define index i and the times η_0 and η_1:

$$i := 0 \tag{5.4}$$
$$\eta_0 := 0 \tag{5.5}$$
$$\eta_1 := t_{cycle} = t_1 - t_0 \tag{5.6}$$

6. Define the motion state input values for the reference DOF κ (underlined variables):

$$\underline{q}_1 := {}_\kappa q_1 \tag{5.7}$$
$$\underline{\dot{q}}_1 := {}_\kappa \dot{q}_1 \tag{5.8}$$
$$\underline{\ddot{q}}_1 := {}_\kappa \ddot{q}_1 \tag{5.9}$$

7. Compute new values for the minimum and maximum accelerations $\ddot{\boldsymbol{q}}_{1,min}$ and $\ddot{\boldsymbol{q}}_{1,max}$ according to Sec. 4.3.2 with $\boldsymbol{q}_1$ and $\dot{\boldsymbol{q}}_1$ taken from Eqn. (5.2).

Remark 5.1.1 *The initialization steps are finished with step 7. Step 8 is the beginning of the loop run until the desired target velocity $\dot{\boldsymbol{q}}_{0,trgt}$ and zero acceleration $\ddot{\boldsymbol{q}}_{0,trgt} = \mathbf{0}$ is reached, see step 18.*

8. Increment i.

9. Compare the acceleration constraints at index i with the the start accelerations constraints at index 0: Compute the vectors $\boldsymbol{\rho}_{i,min}$ and $\boldsymbol{\rho}_{i,max}$ that define the ratios between $\ddot{\boldsymbol{q}}_{0,min}$ and $\ddot{\boldsymbol{q}}_{i,min}$ and $\ddot{\boldsymbol{q}}_{0,max}$ and $\ddot{\boldsymbol{q}}_{i,max}$ respectively

$$\boldsymbol{\rho}_{i,min} = ({}_1\rho_{i,min}, \ldots, {}_k\rho_{i,min}, \ldots, {}_K\rho_{i,min})^T \tag{5.10}$$
$$\boldsymbol{\rho}_{i,max} = ({}_1\rho_{i,max}, \ldots, {}_k\rho_{i,max}, \ldots, {}_K\rho_{i,max})^T \tag{5.11}$$

with

$$\forall\, k \in \{1, \ldots, K\} : \begin{cases} {}_k\rho_{i,min} = \dfrac{{}_k\ddot{q}_{i,min}}{{}_k\ddot{q}_{0,min}} \\ {}_k\rho_{i,max} = \dfrac{{}_k\ddot{q}_{i,max}}{{}_k\ddot{q}_{0,max}} \end{cases} \tag{5.12}$$

10. Define the maximum element of $\boldsymbol{\rho}_{i,min}$ and the minimum element of $\boldsymbol{\rho}_{i,max}$:

$$\gamma_{i,min} = \max({}_1\rho_{i,min}, \ldots, {}_k\rho_{i,min}, \ldots, {}_K\rho_{i,min}) \tag{5.13}$$
$$\gamma_{i,max} = \min({}_1\rho_{i,max}, \ldots, {}_k\rho_{i,max}, \ldots, {}_K\rho_{i,max}) \tag{5.14}$$

11. Calculate $\underline{\ddot{q}}_{i,min}$, $\underline{\ddot{q}}_{i,max}$:

$$\underline{\ddot{q}}_{i,min} := \gamma_{i,min}\ {}_{\kappa}\ddot{q}_{0,min} \tag{5.15}$$
$$\underline{\ddot{q}}_{i,max} := \gamma_{i,max}\ {}_{\kappa}\ddot{q}_{0,max} \tag{5.16}$$
$$\tag{5.17}$$

12. Call the Type IV velocity-based OTG algorithm for *one DOF*
    - Input values: ${}_{\kappa}\boldsymbol{\xi}_i$, $\underline{\ddot{q}}_{i,min}$, $\underline{\ddot{q}}_{i,max}$, ${}_{\kappa}\dddot{q}_{0,min}$, ${}_{\kappa}\dddot{q}_{0,max}$, ${}_{\kappa}\dot{q}_{0,trgt}$
    - Output values:
      - New state of motion
$${}_{\kappa}\boldsymbol{\xi}_{i+1} = ({}_{\kappa}q_{i+1},\ {}_{\kappa}\dot{q}_{i+1},\ {}_{\kappa}\ddot{q}_{i+1}) \tag{5.18}$$

13. Compute the time value of the original trajectory using the inverse function defined in Eqn. (5.27)

$$\eta_{i+1} := {}_{\kappa}\lambda({}_{\kappa}q_{i+1})\ . \tag{5.19}$$

    In case of Eqn. (5.25) or Eqn. (5.26), the value of η_i is used to find the plausible solution among the two or three possible solutions.

14. For all the DOFs other than κ, compute the new positions along the original trajectory, computed in step 3.

$$\forall\, k \in \{1,\ldots,\kappa-1,\kappa+1,\ldots,K\} :\ {}_{k}q_{i+1} = {}_{k}q_0(\eta_{i+1}) \tag{5.20}$$

15. Calculate according to the new positions calculated in step 14 the new velocities and accelerations:

$$\forall\, k \in \{1,\ldots,\kappa-1,\kappa+1,\ldots,K\} : \begin{array}{lcl} {}_{k}\dddot{q}_i & = & \frac{6}{t_{cycle}^3}({}_{k}q_{i+1} - {}_{k}q_i) - \frac{6}{t_{cycle}^2}{}_{k}\dot{q}_i - \frac{3}{t_{cycle}}{}_{k}\ddot{q}_i \\ {}_{k}\dot{q}_{i+1} & = & \frac{1}{2}{}_{k}\dddot{q}_i\, t_{cycle}^2 + {}_{k}\ddot{q}_i\, t_{cycle} + {}_{k}\dot{q}_i \\ {}_{k}\ddot{q}_{i+1} & = & {}_{k}\dddot{q}_i\, t_{cycle} + {}_{k}\ddot{q}_i \end{array} \tag{5.21}$$

16. Compute new values for the minimum and maximum accelerations $\ddot{\boldsymbol{q}}_{i+1,min}$, $\ddot{\boldsymbol{q}}_{i+1,max}$ according to Sec. 4.3.2. The values of $\boldsymbol{q}_{i+1}$ and $\dot{\boldsymbol{q}}_{i+1}$ are taken from Eqn. (5.18, 5.20, 5.21).

17. If the trajectory is executed on a robot system, the computed state of motion

$$\boldsymbol{\Xi}_{i+1} = (\boldsymbol{q}_{i+1}, \dot{\boldsymbol{q}}_{i+1}, \ddot{\boldsymbol{q}}_{i+1}) \tag{5.22}$$

    will be send to the controllers.

18. Go back to step 8 until

$$\dot{\boldsymbol{q}}_i + \ddot{\boldsymbol{q}}_i\, t_{cycle} + \frac{1}{2}\dddot{\boldsymbol{q}}_{min}\, t_{cycle}^2 \le \dot{\boldsymbol{q}}_{0,trgt} \le \dot{\boldsymbol{q}}_i + \ddot{\boldsymbol{q}}_i\, t_{cycle} + \frac{1}{2}\dddot{\boldsymbol{q}}_{max}\, t_{cycle}^2 \tag{5.23}$$

    is reached.

    Remark 5.1.2 *Step 18 was the last step of the loop.*

19. The value of t_{i+1} equals the minimum execution time to reach $\dot{\boldsymbol{q}}_{0,trgt}$ at $\ddot{\boldsymbol{q}}_{0,trgt} = \boldsymbol{0}$ while considering the dynamics of the system. $\boldsymbol{\Xi}_{i+1}$ describes the motion state at the control cycle t_{i+1}, at which $\dot{\boldsymbol{q}}_{0,trgt}$ is reached.

5.1.2. Inverse Functions for Time Lookup

For the algorithm described in Sec. 5.1.1 we need to describe a procedure to calculate the inverse functions of polynomials: A position progression $\boldsymbol{q}_i(t)$ computed at instant t_i consists of single polynomials ${}^{l}_{k}q_i(t)$ that describe for the k-th DOF the l-th polynomial in the collection of $\boldsymbol{q}_i(t)$, see also Eqn. (2.18). This polynomial is represented by either of the following three types (cf. Eqn. (2.20))

$$\text{Linear:} \quad q(t) = a_1 \cdot (t-\Delta t) + a_0 \tag{5.24}$$
$$\text{Quadratic:} \quad q(t) = a_2 \cdot (t-\Delta t)^2 + a_1 \cdot (t-\Delta t) + a_0 \tag{5.25}$$
$$\text{Cubic:} \quad q(t) = a_3 \cdot (t-\Delta t)^3 + a_2 \cdot (t-\Delta t)^2 + a_1 \cdot (t-\Delta t) + a_0 \tag{5.26}$$

a_j with $j \in \{1,2,3\}$ describes the polynomial parameters and Δt stands for the time-shifted argument of the polynomial. The inverse functions of eqns. (5.26) – (5.24)

$$\lambda(q) := (q)^{-1}(q) \tag{5.27}$$

can be unambiguously computed, as shown in the following three sections.

Linear Function Inverse

Only one solution is possible:

$$\lambda(q) = \frac{a_1 \Delta t - a_0 + q}{a_1} \tag{5.28}$$

Quadratic Function Inverse

Two solutions are possible:

1. Define the input form:

$$v := t - \Delta t \tag{5.29}$$
$$q(v) = a_2\, v^2 + a_1\, v + a_0 \tag{5.30}$$

2. Transform Eqn. (5.30) to a root-finding problem:

$$a_0^* = a_0 - q(v) \tag{5.31}$$
$$0 = a_2\, v^2 + a_1\, v + a_0^* \tag{5.32}$$

3. Transform Eqn. (5.32) by dividing it with a_2 to the normal form:

$$0 = v^2 + b_1\, v + b_0 \tag{5.33}$$
$$\text{with} \quad b_1 = \frac{a_1}{a_2} \text{ and } b_0 = \frac{a_0^*}{a_2} \tag{5.34}$$

4. Calculate the discriminant

$$D = \frac{b_1^2}{4} - b_0 \tag{5.35}$$

5. Three possible cases depending on the discriminant value:
   a) Case: $D > 0$, two real solutions:

$$v_{1,2} = -\frac{b_1}{2} \pm \sqrt{D} \tag{5.36}$$

   b) Case: $D \equiv 0$, one real solutions:

$$v_1 = -\frac{b_1}{2} \tag{5.37}$$

c) Case: $D < 0$, no real solutions, two conjugated complex solutions:

$$v_{1,2} = -\frac{b_1}{2} \pm \sqrt{D} \tag{5.38}$$

6. Calculate the time $\lambda(q)$:

$$\lambda_{1,2}(q) \quad = \quad v_{1,2} + \Delta t \tag{5.39}$$

Which solution to choose in the case of two solutions will be described later.

Cubic Function Inverse

Three solutions are possible:

1. Define the input form:

$$v \quad = \quad t - \Delta t \tag{5.40}$$
$$q(v) \quad = \quad a_3\, v^3 + a_2\, v^2 + a_1\, v + a_0 \tag{5.41}$$

2. Transform Eqn. (5.41) to a root-finding problem:

$$a_0^* \quad = \quad a_0 - q(v) \tag{5.42}$$
$$0 \quad = \quad a_3\, v^3 + a_2\, v^2 + a_1\, v + a_0^* \tag{5.43}$$

3. Transform Eqn. (5.43) by dividing it with a_3 to the normal form:

$$0 \quad = \quad v^3 + b_2\, v^2 + b_1\, v + b_0 \tag{5.44}$$
$$\text{with} \quad b_2 = \frac{a_2}{a_3}, b_1 = \frac{a_1}{a_3} \text{ and } b_0 = \frac{a_0^*}{a_3} \tag{5.45}$$

4. Calculate the reduced form of Eqn. (5.44) with no more quadratic part. This is done by using the substitution $v = y - \frac{b_2}{3}$:

$$0 \quad = \quad y^3 + c_1\, y + c_0 \tag{5.46}$$
$$\text{with} \quad c_1 = b_1 - \frac{b_2^2}{3} \text{ and } c_0 = \frac{2\, b_2^3}{27} - \frac{b_2\, b_1}{3} + b_0 \tag{5.47}$$

5. Calculate the discriminant:

$$D = \left(\frac{c_1}{3}\right)^3 + \left(\frac{c_0}{2}\right)^2 \tag{5.48}$$

6. Three possible cases depending on the discriminant value follow:

a) For $D > 0$, one real and two complex solutions, use the solution by Cardano & Tartaglia (published

1545):

$$u = \begin{cases} \sqrt[3]{-\frac{c_0}{2}+\sqrt{D}} & \text{for } \left(-\frac{c_0}{2}+\sqrt{D}\right) \geq 0 \\ -\sqrt[3]{-(-\frac{c_0}{2}+\sqrt{D})} & \text{for } \left(-\frac{c_0}{2}+\sqrt{D}\right) < 0 \end{cases} \tag{5.49}$$

$$w = \begin{cases} 0 & \text{for } u \equiv 0 \\ -\frac{c_1}{3\,u} & \text{else} \end{cases} \tag{5.50}$$

$$\rho_{1/2} := -\frac{1}{2} \pm \frac{1}{2}\sqrt{3}i \tag{5.51}$$

$$y_1 = u+w \tag{5.52}$$
$$y_2 = \rho_1 u + \rho_2 w \tag{5.53}$$
$$y_3 = \rho_2 u + \rho_1 w \tag{5.54}$$

b) Case: $D \equiv 0$, three real solutions, two are the same, use the solution by Cardano/Tartaglia (see 5.a.)

c) Case: $D < 0$, three different real solutions, this is the so-called "casus irreducibilis", solved with trigonometric functions to avoid calculations with complex numbers:

$$u = \sqrt{-\frac{{c_1}^3}{3}} \tag{5.55}$$
$$w = \arccos\left(-\frac{c_0}{2u}\right) \tag{5.56}$$
$$y_1 = 2\,\sqrt[3]{u}\,\cos\left(\frac{w}{3}\right) \tag{5.57}$$
$$y_2 = 2\,\sqrt[3]{u}\,\cos\left(\frac{w}{3}+2\frac{\pi}{3}\right) \tag{5.58}$$
$$y_3 = 2\,\sqrt[3]{u}\,\cos\left(\frac{w}{3}+4\frac{\pi}{3}\right) \tag{5.59}$$

7. Undo the substitution $v = y - \frac{b_2}{3}$ (cf. Eqn. (5.40)):

$$v_{1,2,3} = y_{1,2,3} - \frac{b_2}{3} \tag{5.60}$$

8. Calculate the time $\lambda(q)$:

$$\lambda_{1,2,3}(q) = v_{1,2,3} + \Delta t \tag{5.61}$$

Which solution to choose for the case of having more than one will be described later.

5.1.3. Positional Extremes of DOF κ

The position progression of DOF κ can be described by $\{{}^{l}_{\kappa}q_0(t), {}^{l}_{\kappa}\vartheta_0\}$, where ${}^{l}_{\kappa}\vartheta_0 = [{}^{l}_{\kappa}t_0, {}^{(l+1)}_{\kappa}t_0]$ stands for the time intervals every polynomial ${}^{l}_{\kappa}q_0(t)$ is valid.

We first find all the time instants ${}^{l}_{\kappa}t_{0,extr}$ when the velocity ${}^{l}_{\kappa}\dot{q}_0({}^{l}_{\kappa}t_{0,extr}) = 0$ by using the inverse function defined in Eqn. (5.27). These found time instants ${}^{l}_{\kappa}t_{0,extr}$ indicate the times when the position polynomial ${}^{l}_{\kappa}q_0(t)$ is at one of his extremes. Every position polynomial can only be of third order or less, this is why there can not be more than a minimum and a maximum for every polynomial.

We check for every time instant the acceleration value ${}^{l}_{\kappa}\ddot{q}_0({}^{l}_{\kappa}t_{0,extr})$ to see if at this time we have a maximum or a minimum for the position polynomial and then store it accordingly. The time instant ${}^{l}_{\kappa}t_{0,extr}$ for a minimum is called ${}^{l}_{\kappa}t_{0,min}$ and for a maximum it is called ${}^{l}_{\kappa}t_{0,max}$.

5.1.4. Inverse Lookup on Original Trajectory

Every polynomial is defined for a certain time frame ${}^{l}_{\kappa}\vartheta_0 = [{}^{l}_{\kappa}t_0, {}^{(l+1)}_{\kappa}t_0]$. Check for every polynomial section l starting from $l = 1 \ldots L$ if $\eta_i \leq {}^{(l+1)}_{\kappa}t_0$. If true, the current polynomial section l could contain the possible solution:

For that polynomial we first check if the new position value has gone past an extreme of the polynomial ("past a turning point"). The extremes have been calculated in step 4. We can have the case of a minimum and maximum or only a minimum or only a maximum. For example, if only a minimum exists, we check if $\eta_i < {}^{l}_{\kappa}t_{0,min}$ and ${}_{\kappa}q_{i+1} \leq {}^{l}_{\kappa}q_0({}^{l}_{\kappa}t_{0,min})$. If this is the case, the lookup indicates that ${}_{\kappa}q_{i+1}$ is lower than the minimum of the original position trajectory which we use for the lookup. We therefore define $\eta_{i+1} = {}^{l}_{\kappa}t_{0,min}$ as the output time. For the case of having a maximum or a minimum/maximum, this check is done in an analog way.

If we are not past a turning point, we use the inverse function defined in Eqn. (5.27) to lookup the time instant(s) when the position value ${}_{\kappa}q_{i+1}$ is valid on the polynomial ${}^{l}_{\kappa}q_0(t)$. We can have at max three possible solutions. The possible time instant has to be inside the time frame: ${}^{l}_{\kappa}t_0 \leq \eta_i \leq {}^{(l+1)}_{\kappa}t_0$. The smallest of the possible time solutions which fulfills this requirement is chosen as η_{i+1}.

In a last step, we check for overshooting, which means if we have surpassed the last value of the position trajectory ${}^{l}_{\kappa}q_0(t_{sync})$. If that is the case, we will go to step 18 of this VDOTG algorithm.

5.2. Velocity-Based Dynamic OTG - Second Approach

In the following we will describe the second approach taken to optimize the OTG Type IV as a target velocity-based version. An unforeseen event happens at t_i, and the system is in a motion state Ξ_i from which we have to generate a new trajectory in order to reach a user-given value

$$\Xi_{i,trgt} = \{\dot{\boldsymbol{q}}_{i,trgt}, \ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}\} \tag{5.62}$$

in the shortest possible time. The new algorithm will optimize an initially calculated motion trajectory by adjusting it at every time step, if needed. The optimization is based on the acceleration constraints calculated at every motion state. The used path is computed by an initial run of the OTG algorithm. This algorithm will not necessarily find the global optimum for a trajectory accounting for a dynamic acceleration behavior, but it will optimize in real-time the initially given trajectory to a more suitable motion.

5.2.1. VDOTG Algorithm

An overview of the algorithmic procedure is given in Fig. 5.1. The steps are detailed in the following:

1. Define $i = 0$
2. For the values $\boldsymbol{q}_i$ and $\dot{\boldsymbol{q}}_i$ of the current motion state Ξ_i, calculate the joint space dynamic parameters of the system as described in Sec. 2.2. The "Rigid Body Dynamics Algorithm" of Featherstone [12] is used.
3. For the current motion state Ξ_i, calculate new values for the minimum and maximum accelerations $\ddot{\boldsymbol{q}}_{i,min}$, $\ddot{\boldsymbol{q}}_{i,max}$ as described in Sec. 4.3.2.
4. Call the Type IV target velocity-based OTG algorithm. This will provide a trajectory (incl. path) for all K DOFs to reach $\dot{\boldsymbol{q}}_{i,trgt}$ and $\ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}$ at the same time instant.
   - Input values:
     - motion state: Ξ_i
     - current motion constraints: $\ddot{\boldsymbol{q}}_{min}$, $\ddot{\boldsymbol{q}}_{max}$, $\dddot{\boldsymbol{q}}_{0,min}$, $\dddot{\boldsymbol{q}}_{0,max}$,
     - target motion state: $\dot{\boldsymbol{q}}_{0,trgt}$
   - Output values: new state of motion
     - next motion state: Ξ_{i+1}

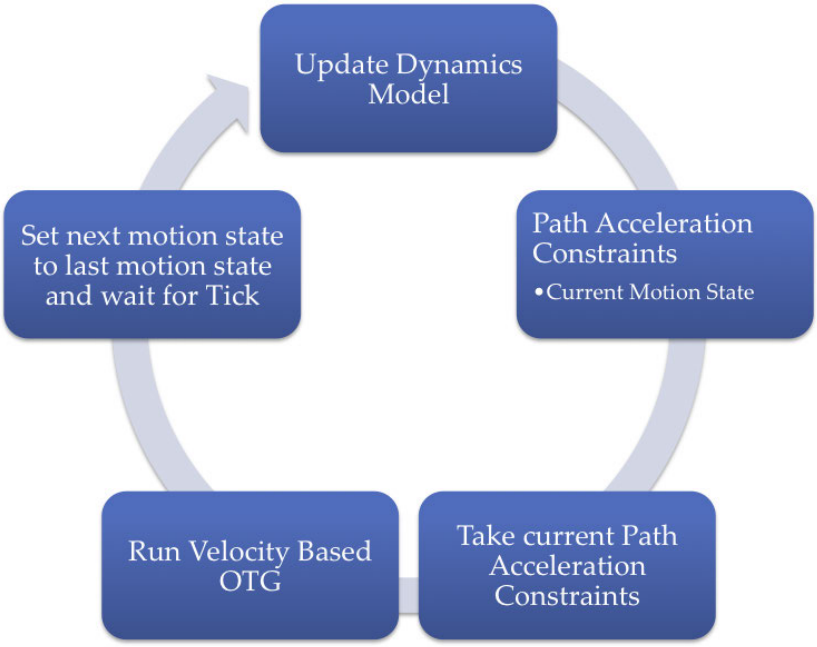


Figure 5.1.: Velocity-Based Dynamic OTG - Second Approach.

5. *Optional:* If the trajectory is executed on a robot system, Ξ_{i+1} will be send to the controllers.
6. Increment i
7. Check if $\dot{\boldsymbol{q}}_{0,trgt}$ is reached. If not, go back to step 2.
8. Ξ_{i+1} describes the motion state at which $\dot{\boldsymbol{q}}_{i,trgt}$ is reached.

5.3. Position-Based Dynamic OTG

Based on the second VDOTG approach given in Sec. 5.2, we will now introduce the Position-Based Dynamic OTG. This is about optimizing the OTG Type IV as a target position-based version. An unforeseen event happens at t_i, and the system is in a motion state Ξ_i from which we have to generate a new trajectory in order to reach a user-given value

$$\Xi_{i,trgt} = \{\boldsymbol{q}_{i,trgt}, \dot{\boldsymbol{q}}_{i,trgt}, \ddot{\boldsymbol{q}}_{i,trgt} = \boldsymbol{0}\} \tag{5.63}$$

in the shortest possible time. The new algorithm will optimize an initially calculated motion trajectory by adjusting it at every time step, if needed. The optimization is based on the acceleration constraints calculated at every current and also a future motion state. The first used path is computed by an initial run of the OTG algorithm. This algorithm will not necessarily find the global optimum for a trajectory accounting for a dynamic acceleration behavior, but it will optimize in real-time the initially given trajectory to a more suitable one.

5.3.1. PDOTG Algorithm

A brief overview of the algorithmic procedure is given in Fig. 5.2. The detailed steps to take are listed in the following:

1. Define $i = 0$

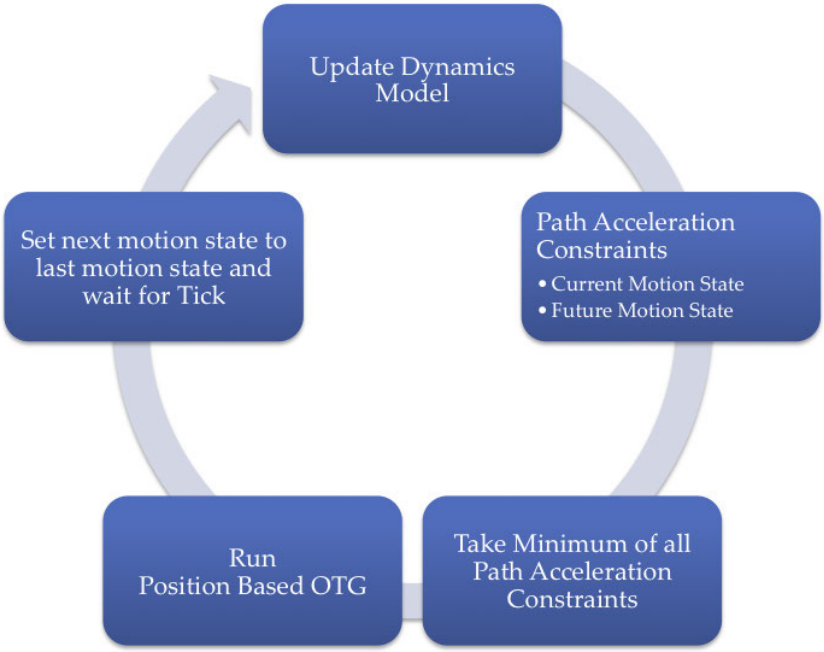


Figure 5.2.: Position-Based Dynamic OTG.

2. Update dynamic parameters $\boldsymbol{M}$, $\boldsymbol{c}$ etc. for target motion state: Calculate the joint space dynamic parameters of the system as described in Sec. 2.2 for the values $\boldsymbol{q}_{0,trgt}$ and $\dot{\boldsymbol{q}}_{0,trgt}$ of the target motion state $\boldsymbol{\Xi}_{0,trgt}$. The "Rigid Body Dynamics Algorithm" of Featherstone [12] is used.

3. Acceleration Constraints for target motion state: Calculate new values for the minimum and maximum accelerations $\ddot{\boldsymbol{q}}_{i,trgt,min}$, $\ddot{\boldsymbol{q}}_{i,trgt,max}$ as described in Sec. 4.3.2 for the target motion state $\boldsymbol{\Xi}_{0,trgt}$.

4. Update the dynamic parameters for the current motion state $\boldsymbol{\Xi}_i$, see Sec. 2.2.

5. Calculate the acceleration constraints $\ddot{\boldsymbol{q}}_{i,cur,min}$, $\ddot{\boldsymbol{q}}_{i,cur,max}$ for the current motion state $\boldsymbol{\Xi}_i$ according to the outline given in Sec. 4.3.2.

6. Compare and find minimum of the constraints of the target and current motion state:

$$\forall k \in (1,\ldots,K): \tag{5.64}$$
$$_k\ddot{q}_{i,min} = \max(_k\ddot{q}_{i,cur,min}, {}_k\ddot{q}_{i,trgt,min}) \tag{5.65}$$
$$_k\ddot{q}_{i,max} = \min(_k\ddot{q}_{i,cur,max}, {}_k\ddot{q}_{i,trgt,max}) \tag{5.66}$$
$$\tag{5.67}$$

7. Call the Type IV target position-based OTG algorithm. This will provide a trajectory (incl. path) for all K DOFs to reach $\boldsymbol{q}_{0,trgt}$, $\dot{\boldsymbol{q}}_{0,trgt}$ and $\ddot{\boldsymbol{q}}_{0,trgt} = \boldsymbol{0}$ at the same time instant.
   - Input values:
     - motion state: $\boldsymbol{\Xi}_i$
     - current motion constraints: $\dot{\boldsymbol{q}}_{0,min}$, $\dot{\boldsymbol{q}}_{0,max}$, $\ddot{\boldsymbol{q}}_{i,min}$, $\ddot{\boldsymbol{q}}_{i,max}$, $\dddot{\boldsymbol{q}}_{0,min}$, $\dddot{\boldsymbol{q}}_{0,max}$,
     - target motion state: $\boldsymbol{q}_{0,trgt}$, $\dot{\boldsymbol{q}}_{0,trgt}$
   - Output values:
     - next motion state: $\boldsymbol{\Xi}_{i+1}$

– Motion trajectory

$$\mathbf{\Phi}_i(t) = \left\{ \left({}^1\boldsymbol{\xi}_i(t), {}^1\mathscr{V}_i\right), \ldots, \left({}^l\boldsymbol{\xi}_i(t), {}^l\mathscr{V}_i\right), \ldots, \left({}^L\boldsymbol{\xi}_i(t), {}^L\mathscr{V}_i\right) \right\} \tag{5.68}$$

– synchronization time, which is the time needed till the target motion state would be reached: $t_{i,sync}$

8. *Optional:* If the trajectory is executed on a robot system, $\mathbf{\Xi}_{i+1}$ will be send to the controllers.
9. Increment i ($i = 1$)
10. Set next motion state to the new current motion state: $\mathbf{\Xi}_i = \mathbf{\Xi}_{i-1}$
11. Check if $\mathbf{\Xi}_{0,trgt}$ is reached. If yes, go to step 23
12. *Future Motion State Principle:* Define the future time instant between the current motion state and the target motion state in the following way:

$$t_{i,fut} = t_{cycle}\, i\, (k_{exp} - 1) \tag{5.69}$$

with k_{exp} is an expansion factor. Good values are between 1 and 3, the best value turned out to be 2. We choose an additional motion state in the future in order to predict the behavior of the acceleration constraints and allow therefore constant inputs to the OTG after the future motion state lookup has reached the target motion state. With an expansion factor of 2, that would be approximately after half of the execution time has elapsed. The concept is illustrated in Fig. 5.3.

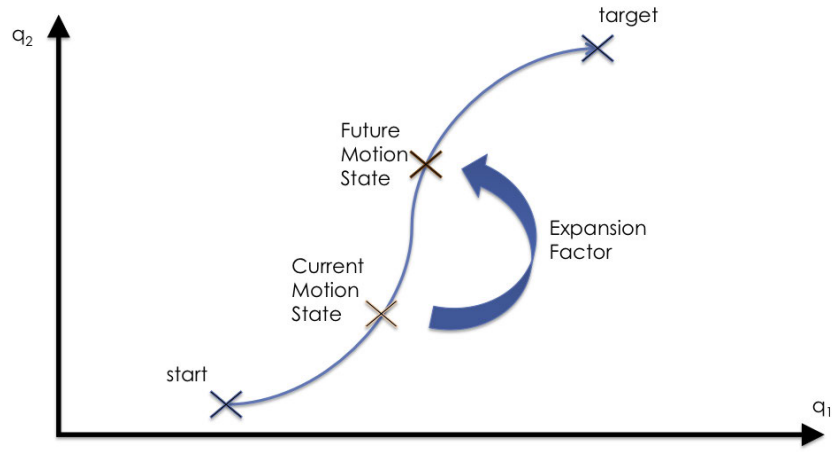


Figure 5.3.: The Future Motion State Principle.

13. if $t_{i,fut} \leq t_{i,sync}$
    - then
        a Calculate trajectory value at $t_{i,fut}$:

$$\mathbf{\Xi}_{i,fut} = \mathbf{\Phi}_i(t_{i,fut}) \tag{5.70}$$

        b Update dynamic parameters for this trajectory value.
        c Calculate acceleration constraints $\ddot{\boldsymbol{q}}_{i,fut,min}$, $\ddot{\boldsymbol{q}}_{i,fut,max}$ for this future motion state $\mathbf{\Xi}_{i,fut}$.
    - else we have already fully predicted all the future motion steps to look at, since $t_{i,fut} > t_{i,sync}$. As higher the expansion factor, as faster this condition is reached.

14. Update dynamic parameters for current motion state Ξ_i, see Sec. 2.2.

15. Calculate acceleration constraints $\ddot{\boldsymbol{q}}_{i,cur,min}$, $\ddot{\boldsymbol{q}}_{i,cur,max}$ for current motion state Ξ_i, see Sec. 4.3.2.

16. if $t_{i,fut} \leq t_{i,sync}$

    - then compare and find the minimum of the constraints for future and current motion state:

$$\forall k \in (1,\ldots,K): \tag{5.71}$$
$$_k\ddot{q}_{i,min} = \max(_k\ddot{q}_{i,cur,min}, {}_k\ddot{q}_{i,fut,min}) \tag{5.72}$$
$$_k\ddot{q}_{i,max} = \min(_k\ddot{q}_{i,cur,max}, {}_k\ddot{q}_{i,fut,max}) \tag{5.73}$$
$$\tag{5.74}$$

    - else set:

$$\forall k \in (1,\ldots,K): \tag{5.75}$$
$$\ddot{\boldsymbol{q}}_{i,min} = \ddot{\boldsymbol{q}}_{i,cur,min} \tag{5.76}$$
$$\ddot{\boldsymbol{q}}_{i,max} = \ddot{\boldsymbol{q}}_{i,cur,max} \tag{5.77}$$

17. Wait for real time cycle to finish.

18. Call the Type IV target position-based OTG algorithm. This will provide a trajectory (incl. path) for all K DOFs to reach $\boldsymbol{q}_{0,trgt}$, $\dot{\boldsymbol{q}}_{0,trgt}$ and $\ddot{\boldsymbol{q}}_{0,trgt} = \boldsymbol{0}$ at the same time instant.
    - Input values:
      - motion state: Ξ_i
      - current motion constraints: $\dot{\boldsymbol{q}}_{0,min}$, $\dot{\boldsymbol{q}}_{0,max}$, $\ddot{\boldsymbol{q}}_{i,min}$, $\ddot{\boldsymbol{q}}_{i,max}$, $\dddot{\boldsymbol{q}}_{0,min}$, $\dddot{\boldsymbol{q}}_{0,max}$,
      - target motion state: $\boldsymbol{q}_{0,trgt}$, $\dot{\boldsymbol{q}}_{0,trgt}$
    - Output values:
      - next motion state: Ξ_{i+1}
      - Motion trajectory

$$\boldsymbol{\Phi}_i(t) = \left\{ \left({}^1\boldsymbol{\xi}_i(t), {}^1\mathscr{V}_i\right), \ldots, \left({}^l\boldsymbol{\xi}_i(t), {}^l\mathscr{V}_i\right), \ldots, \left({}^L\boldsymbol{\xi}_i(t), {}^L\mathscr{V}_i\right) \right\} \tag{5.78}$$

      - synchronization time, which is the time needed till the target motion state would be reached: $t_{i,sync}$

19. *Optional:* If the trajectory is executed on a robot system, Ξ_{i+1} will be send to the controllers.

20. Increment i

21. Set next motion state to the new current motion state: $\Xi_i = \Xi_{i-1}$

22. Check if $\boldsymbol{q}_{0,trgt}$ and $\dot{\boldsymbol{q}}_{0,trgt}$ is reached. If not, go back to step 12.

23. Ξ_{i+1} describes the motion state at which the target motion state is reached. This is the end of the algorithm.

5.4. Summary - Dynamic Online Trajectory Generation

In the previous chapter has been described how to calculate the acceleration constraints for a given motion state (cf. Sec. 4.3.2). This knowledge has now been used in this chapter to combine it with the OTG algorithm to develop three versions of a Dynamic Online Trajectory Generator (DOTG). The first version uses the target

velocity-based OTG and is outlined in Sec. 5.1.1. This approach uses the idea to first calculate an initial trajectory and then optimize it at every time step using the OTG only for one DOF, the one that determines the execution time. The input constraints of the OTG are calculated taking into account the acceleration constraints for every DOF. The one-dimensional output result of the OTG is feed to an inverse lookup of the original trajectory in order to determine the new motion state of the remaining DOFs.

The second version, again an approach using the target velocity-based OTG, uses for all DOFs the current acceleration constraints in every time step as input for the OTG. An overview of this simplified approach is given in Fig. 5.1.

The third version is an enhancement of the second version so it can be used with the target position-based OTG. Not only the current acceleration constraints but also future motion constraints are considered so that an actual planning ahead of time becomes possible(cf. Fig. 5.3). An overview of the algorithm is given in Fig. 5.2.

6. Implementation and Results

6.1. Implementation

6.1.1. Simulation

All initial ideas and approaches for suitable DOTG algorithms were coded and simulated using the functional programming language Mathematica [41]. The Mathematica language provides a large collection of predefined functions and visualization options that allows for fast testing and simulation of ideas for algorithms.

The OTG Algorithm, which is implemented in C++, needed to be linked/integrated into the Mathematica environment. The MathLink Library [42] provided by Mathematica was used for this. All other parts needed for the simulation including a dynamics engine we written from scratch.

6.1.2. Real-Time Code

The second approach of the VDOTG (Sec. 5.2) and the PDOTG (Sec. 5.3) were implemented in C++ after testing them in Mathematica. The dynamics engine used is the so-called "Tao Dynamics Engine" [37], which allows to model dynamic behavior of branching structures in an accurate and simple form while taking computational efficiency into account. For Matrix Calculations, we used the "Eigen" library [14]. The trajectories were implemented using the OTG Type IV implementations, which are part of the "Reflexxes Motion Libraries" (RML) [30]. The joint space dynamic parameters of the system

- mass matrix $M(\boldsymbol{q})$
- actuator force/torque $\boldsymbol{\tau}$
- gravity force/torque $\boldsymbol{\tau}_g(\boldsymbol{q})$
- and Coriolis/centrifugal force/torque $c(\boldsymbol{q}, \dot{\boldsymbol{q}})$

for the current state of motion $\boldsymbol{\Xi}_i$ are calculate using the "Rigid Body Dynamics Algorithm" of Featherstone [12].

For implementation of the algorithm on a robot platform, the "KUKA Light-Weight Robot" [4, 5] has been used. The arm has an excellent mass-payload ratio if no additional payload is added to it. To demonstrate more realistic scenarios of moving a tool through space, the robot received a metal disk as a payload with a weight of 4.5 kg. This setup is shown in Fig.6.1.

The interface between the LWR position controller and the DOTG application is called "Fast Research Interface" [32, 26, 22]. A cycle time of 2 ms was chosen.

6.1.3. OTG Limitations

The current implementation of the target-position based OTG Type IV only works for symmetric motion constraints:

$$\begin{aligned} \dot{\boldsymbol{q}}_{i,min} &= -\dot{\boldsymbol{q}}_{i,max} \\ \ddot{\boldsymbol{q}}_{i,min} &= -\ddot{\boldsymbol{q}}_{i,max} \\ \dddot{\boldsymbol{q}}_{i,min} &= -\dddot{\boldsymbol{q}}_{i,max} \end{aligned} \tag{6.1}$$

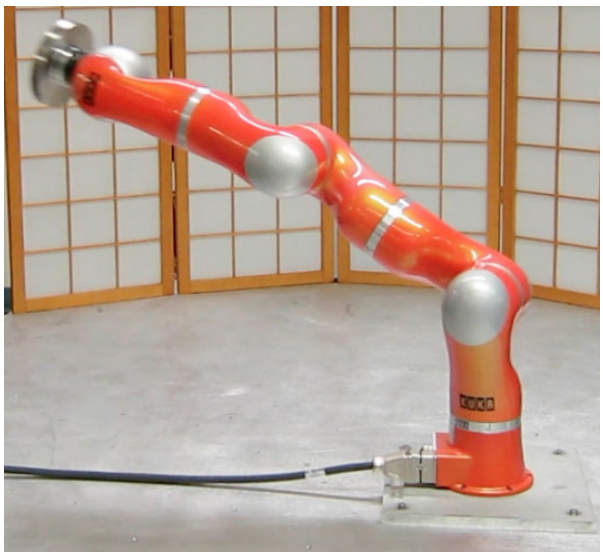


Figure 6.1.: Light Weight Robot arm.

In order to make full use of a system's variable acceleration capabilities, the target position-based OTG implementation should be enhanced by taking into account asymmetric constraints. For now, the implementation done in this thesis can only use what exists and therefore does the following simplification:

$$\ddot{\boldsymbol{q}}_{i,min} = -\ddot{\boldsymbol{q}}_{i,max} \tag{6.2}$$

$$\tag{6.3}$$

will hold permanently, whereas $\ddot{\boldsymbol{q}}_{i,max}$ is computed as

$$\forall\, k \in \{1,\ldots,K\}: \quad {}_k\ddot{q}_{i,max} := \begin{cases} {}_k\ddot{q}_{i,max} & \text{if } |{}_k\ddot{q}_{i,max}| \geq |{}_k\ddot{q}_{i,min}| \\ |{}_k\ddot{q}_{i,min}| & \text{otherwise} \end{cases} . \tag{6.4}$$

The same is done for $\dot{\boldsymbol{q}}_{i,min/max}$ and $\dddot{\boldsymbol{q}}_{i,min/max}$.

6.1.4. LWR Model Identification and Verification

An identification for the dynamic model for the KUKA LWR is required in order to calculate the acceleration capabilities. A simple experiment was designed for the fast acquisition of the manipulator masses. The robot has seven DOFs. Its kinematic setup can be found in the robot manual [25].

The robot's links 1,2,3 and 4 are almost identical except for different electronics in the interior. We simply assume that these four links are identical so we can simplify the identification process. The 7 DOF robot is considered a 3 DOF device by lumping links 2 and 3 together to link 2* and lumping links 4, 5, 6 and 7 to link 3*. We then identify the masses of theses lumped links using the internal torque sensor measurements for six distinct static position configurations. This collection of positions can be summarized as following:

$$\text{Coll.}(\boldsymbol{q}^*) = \{(0°,0°,90°),(0°,0°,-90°)(0°,90°,0°),(0°,-90°,0°),(0°,90°,90°)(0°,-90°,-90°)\} \tag{6.5}$$

The velocities and accelerations are zero for every motion state are zero.

We make assumptions for the gravity center position and then use the torque measurements to calculate the masses of the lumped links. Based on the first assumptions of viewing the first 4 links identical, we can then determine the mass for this link type and also get the value for the rest mass of links 5,6 and 7. Splitting this rest

mass through an estimate in its parts leads to 7 mass values.

We use these 7 mass values and the geometric measures of the robot to calculate the moment of inertias. This is done using the formula of the moment of inertia for hollow cylindrical and hollow spherical geometric shapes. The links 1 to 5 and 7 are considered as hollow cylinders whereas link 6 is considered as a hollow sphere.

On top of Link 7, a payload is also lumped. The payload was identified separately before mounting it, its values can be found in tab. B.2. The finally obtained values for the masses, moments of inertias and center of masses can be found among other values in tab. B.1.

A demonstration of the identification results is given in Fig.6.2: the calculated and measured torques for an exemplary motion of the LWR is given. In the first line of the figure, the blue line describes the measured torques, the purple line the calculated torques using the identified dynamic robot parameters. The calculated and measured torques match throughout the dynamic motion which shows that the identified parameters are good enough for testing the DOTG. The graphs for the joints 1, 2 and 4 are shown, because just these three joints are enough to allow positioning of the LWR in its full workspace.

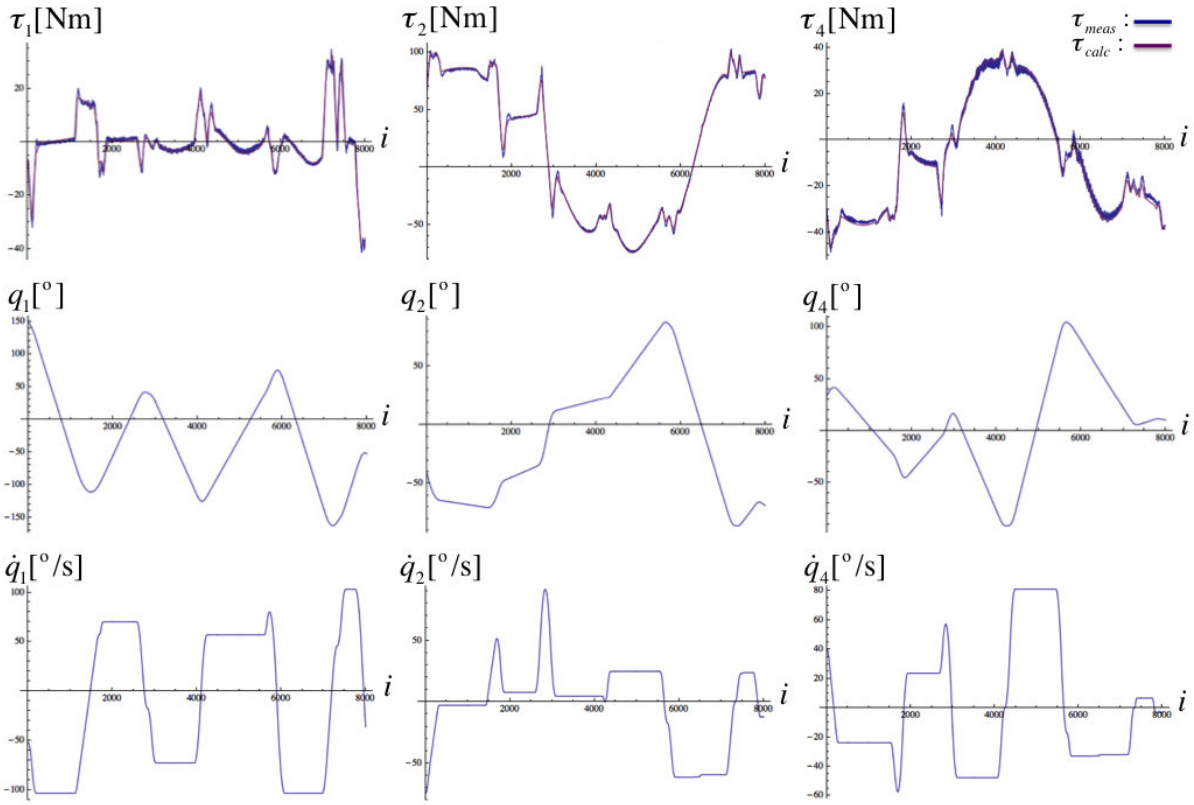


Figure 6.2.: Comparision of measured and calculated torques for a complete motion trajectory. The execution time for this trajectory ist at 1.6 seconds which is equal to 8000 steps *i*.

6.2. Experimental Results

In the following will we describe experimental results of the VDOTG and PDOTG run on the LWR. Demonstration videos can be found on the website: `http://research.katzschmann.eu`.

6.2.1. VDOTG Results

Simulation Results

For the first approach of the VDOTG as described in Sec. 5.1.1), several simulations of it using Mathematica have shown its unstable behavior. Only after a few time steps, the algorithm suddenly destabilizes and the values go astray, as you can see in Fig. 6.3. The figure shows in the first row the results of a normal OTG run for 73 time steps and the second row shows what the first approach of the VDOTG does in the same time frame. After 60 steps, axis 1 and axis 4 suddenly go astray from the values the normal OTG calculates (using the acceleration constraints calculated in the first time step). This would not necessarily be a problem if the target velocity would finally be reached, but instead of this, the VDOTG breaks up after 73 steps because the allowable maximum velocity is breached for axis 4 at that moment. The normal OTG trajectory reaches the target velocity after 175 steps.

This behavior even happens when varying the input values randomly, it always destabilizes before reaching the target velocity. During the thesis, several smaller adjustments had been made to the algorithmic procedure, but none of them stabilized the algorithm.

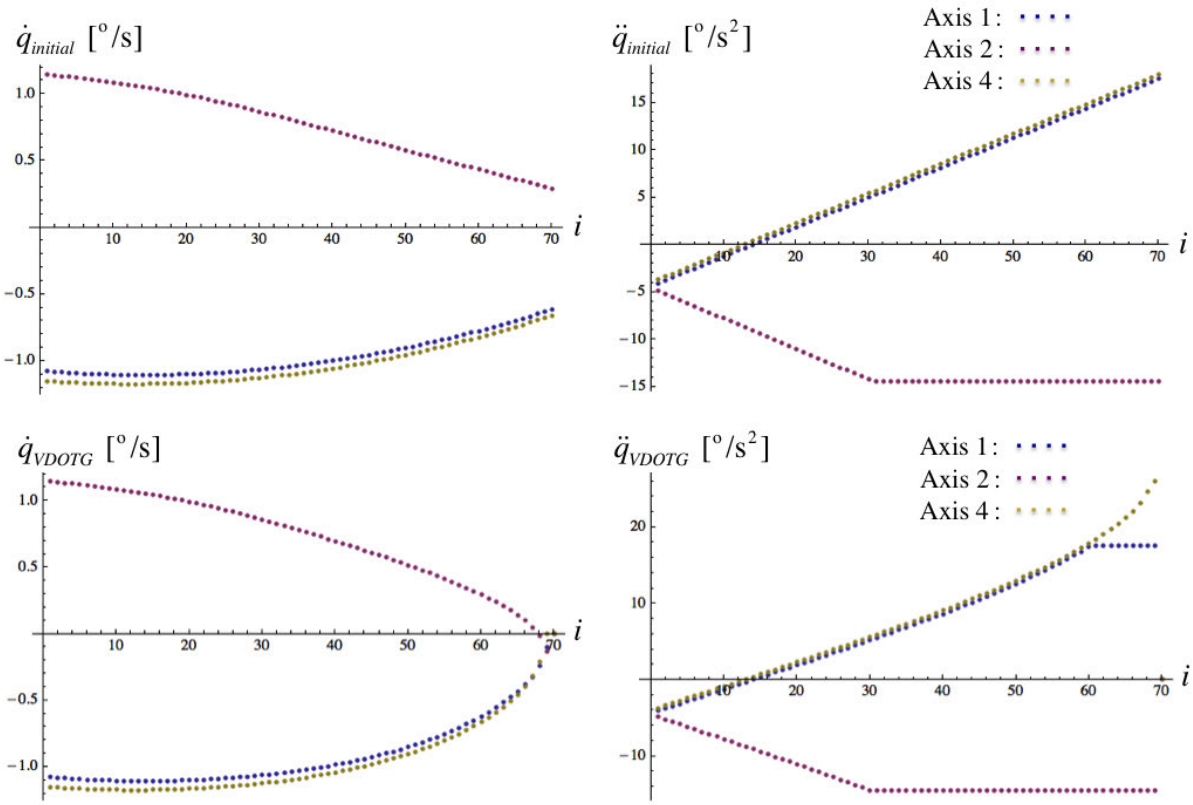


Figure 6.3.: VDOTG first approach issues.

That is why we chose to work on a second approach for the VDOTG, which is outlined in Sec. 5.2. The second approach worked out much better than the first one. As you can see in Fig. 6.4, the initial OTG trajectory shown in the first row uses the acceleration constraints calculated at start, while the VDOTG adjusts the trajectory during the whole motion. The OTG would breach the acceleration constraints after about 50 steps while the VDOTG stays within the upper and lower acceleration constraints. These constraints are explicitly shown in Fig. 6.5.

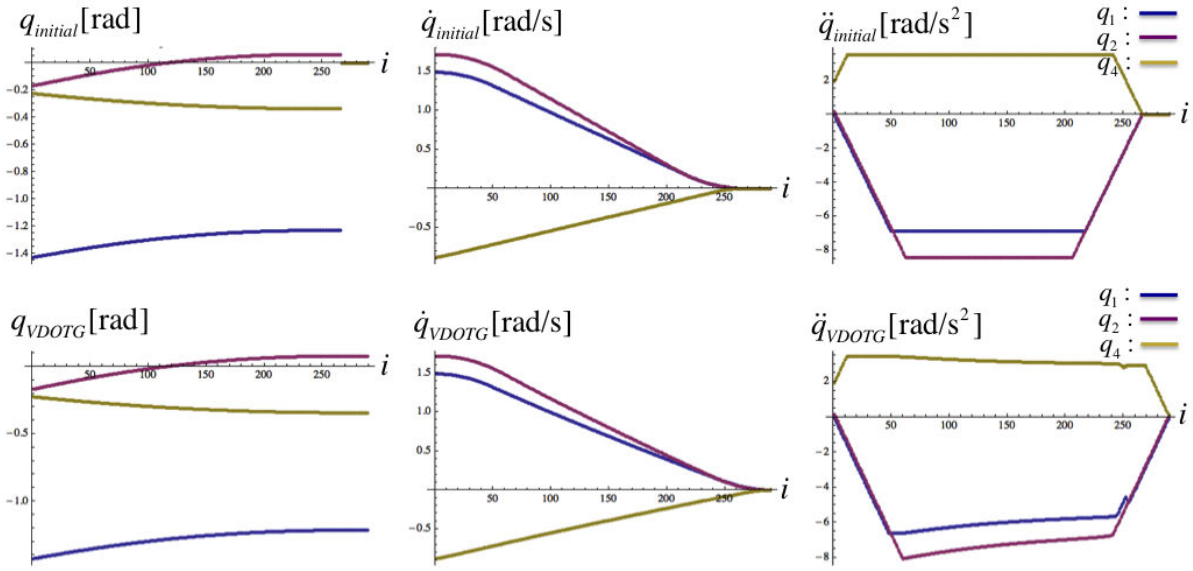


Figure 6.4.: VDOTG second approach: Comparison of the initial OTG trajectory and the VDOTG trajectory.

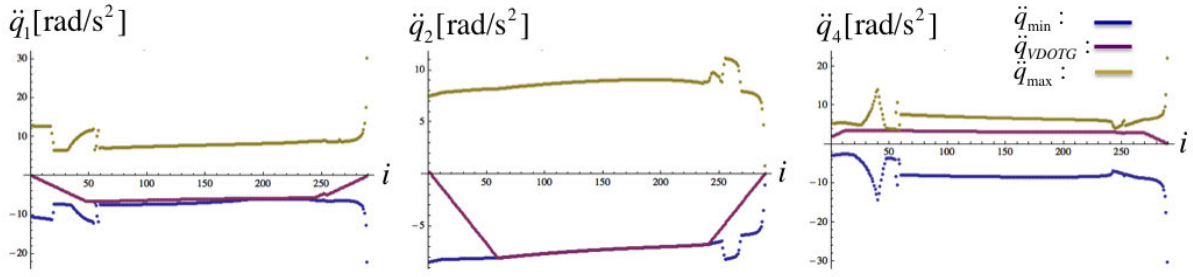


Figure 6.5.: VDOTG second approach: VDOTG values within minimal and maximal acceleration constraints.

Results with LWR

We were able to show in simulation that the second approach of the VDOTG stays within the limits and runs stable. In the simulation we intentionally lowered the calculation values for the minimal and maximal available torque capabilities of the LWR while maintaining reasonable jerk constraints. This way we were able to generate the figures shown above to demonstrate the functionality of the VDOTG. After reimplementing the VDOTG to run directly on the real LWR, we chose more realistic torque capability values that actually resemble the limits of the system. For this scenario we were only able to generate trajectories as shown in Fig. 6.6 which did not reach the limits of the robot's acceleration capabilities. When we lowered the torque capabilities of the robot artificially we received the same results as shown in the simulations before(cf. Fig. 6.4 and Fig 6.5). We concluded that it takes more experiments with specific start and target motion states to eventually allow the VDOTG to bring the LWR to its real limitations without exceeding them. Since we showed that the concept worked and our focus was more on getting a PDOTG running, we did not dig deeper into this.

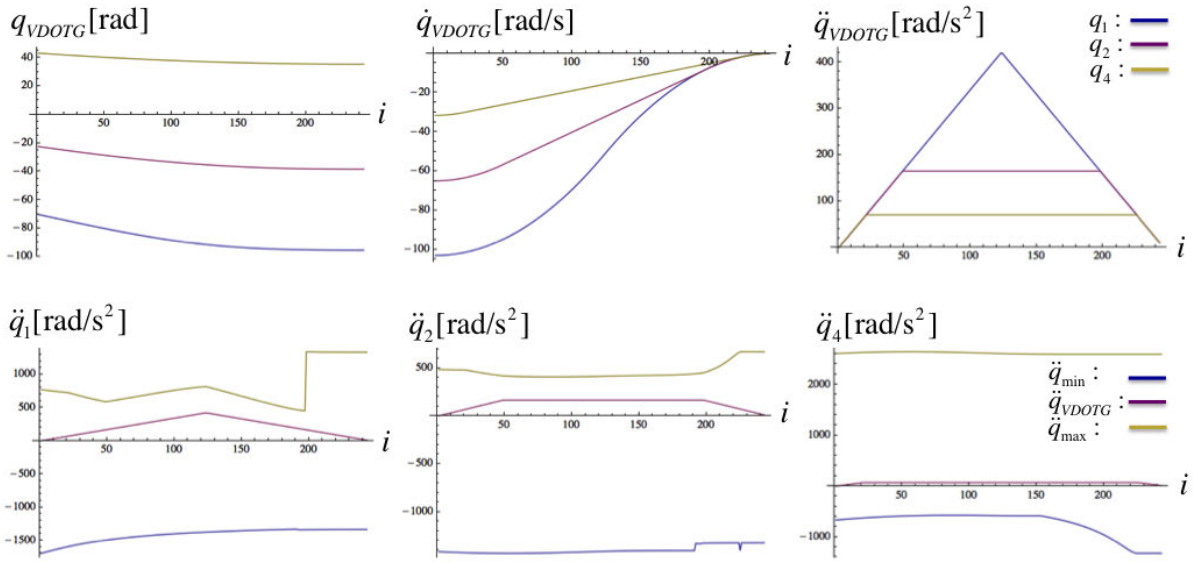


Figure 6.6.: VDOTG second approach: Experimental results running the VDOTG on the controller of the LWR.

6.2.2. PDOTG Results

The PDOTG applied to the LWR leads to acceleration profiles as shown in Fig. 6.7. The constraints are conservative, because they are calculated using the Future Motion State principle (cf. Fig. 5.3) and they are symmetric due to the limitation of the target position-based OTG to only take symmetric constraints as input (cf. Sec. 6.1.3). The lower constraints in blue and the upper constraints in yellow visualize this. Both features disallow the PDOTG to fully use the available acceleration capabilities — the acceleration trajectory generated by the PDOTG is limited more than actually needed.

In comparison to this, Fig. 6.8 shows how the acceleration limits would look like if the PDOTG would allow asymmetric constraints. The lost potential is significant.

The future motion state principle (cf. Fig. 5.3) used for the PDOTG is necessary to plan ahead in time, but requires a lot of potentially available acceleration capabilities to not be used. In Fig. 6.9 is shown how the real acceleration constraints look like in comparison to the conservative trajectory of the PDOTG.

A second set of results is given in Fig. 6.10, 6.11 and 6.12 to show the functionality of the PDOTG and its limitation for another motion path.

The experiments were not only done for 3 DOFs, but actually for all 7 DOFs in motion. The result was the same, the PDOTG generated trajectories for all DOFs while taking into account the acceleration constraints.

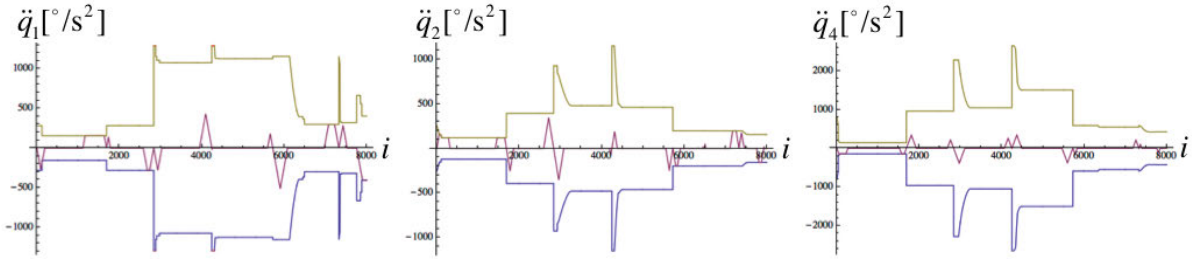


Figure 6.7.: Position Based OTG Example 1: Conservative and Symmetric Acceleration Constraints.

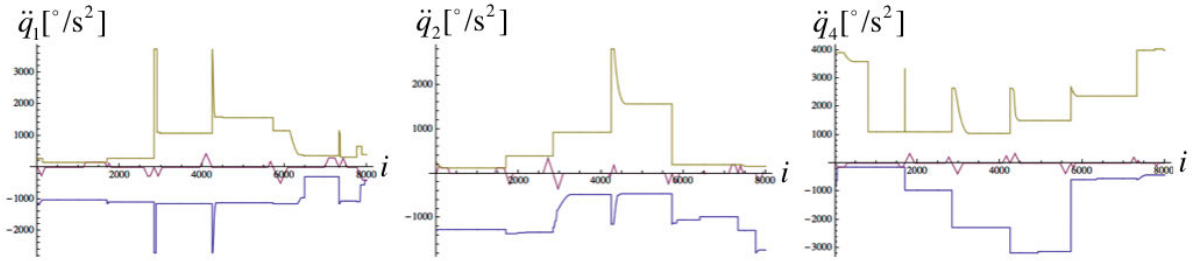


Figure 6.8.: Position Based OTG Example 1: Conservative Acceleration Constraints.

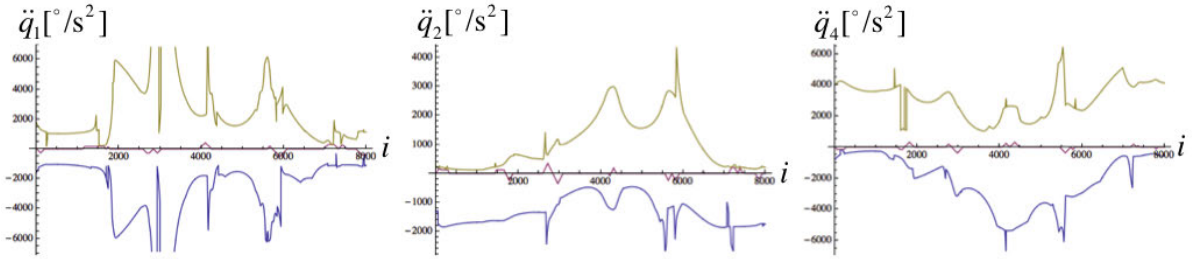


Figure 6.9.: Position Based OTG Example 1: Real Acceleration Constraints.

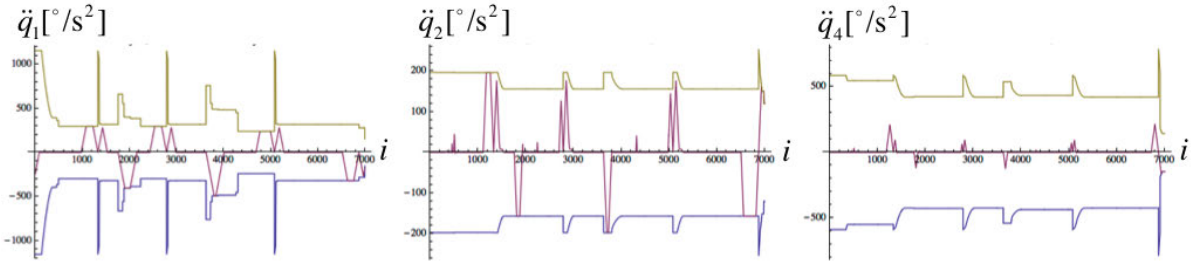


Figure 6.10.: Position Based OTG Example 2: Conservative and Symmetric Acceleration Constraints.

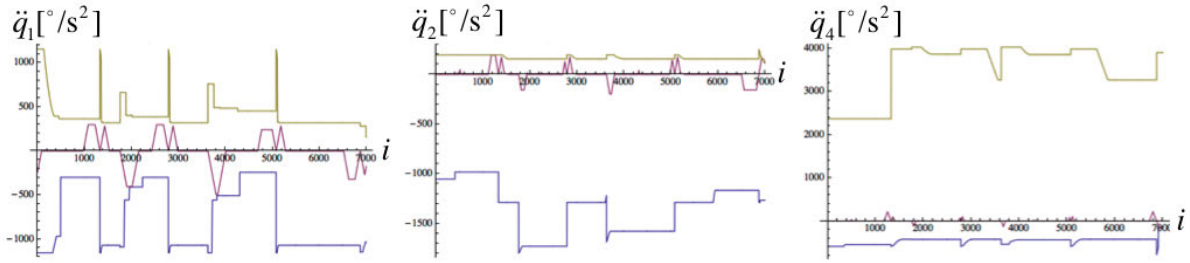


Figure 6.11.: Position Based OTG Example 2: Conservative Acceleration Constraints.

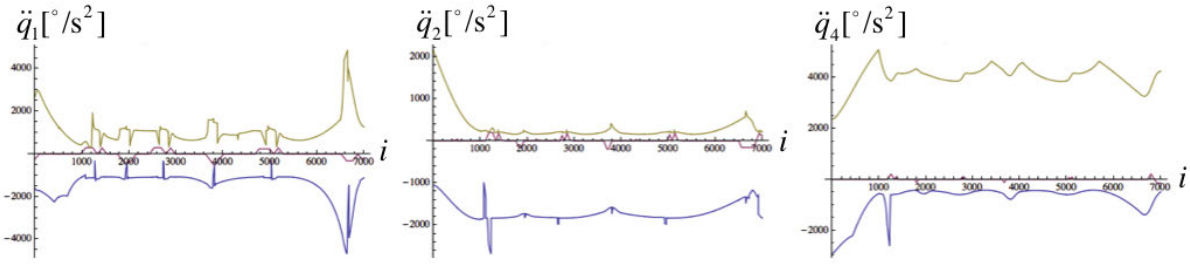


Figure 6.12.: Position Based OTG Example 2: Real Acceleration Constraints.

7. Discussions

It follows a discussion on the problems and limitations of the algorithms and implementations of the previous chapters. Along that line, further ideas and alternative approaches are provided: a description on what else could be done with the new findings on acceleration capabilities and a different approach on generating constrained trajectories are given in the second part of this chapter.

7.1. Problems and Limitations

The problems and limitations of the proposed algorithm can be divided into three areas: Modeling of the robot dynamics, trajectory generation and the connection/interrelation of both.

7.1.1. Dynamics Modelling

- Accurate dynamic model parameters would certainly improve the modeling results of the acceleration capabilities. In Sec. 6.1.4 gives a pragmatic approach on how to quickly obtain useful parameters for the LWR by identifying the masses of the links and estimating the inertias by pure calculation. More accurate results for any robot can be obtained using more advanced methods, for example the periodic excitation approach by Swevers [38].
- The dynamic model could be extended to account for effects like friction in the robot drivetrains. The acceleration capability modeling would benefit from a more detailed robot model. For this thesis work, only the major effects were considered when describing the dynamics of the system (cf. Sec. 2.2).
- The real motor torque capabilities are usually not known as in the case of the LWR and therefore would need to be identified through experiments. It might be possible to find motor specifications for certain industrial robots, but this is not always the case. The manufacturers do not provide these values, much less a model for its dependency to the joint velocities, see also the next point.
- The extremal torque capabilities are in general a function of the joint velocities. This relation was neglected for the implementation in this thesis, but it could easily be added into the algorithms — it would not change the concept of the algorithm. Unfortunately, such extreme torque models are not provided by the manufacturer but have to be determined through own identification experiments.

7.1.2. Online Trajectory Generation

- The OTG allows only for constant motion constraints as input when planing the trajectory from the current to the target motion state. The OTG assumes that the constraint is valid throughout the motion. Instead of using the OTG, other trajectory generators might be more suitable to take into consideration variable motion constraints. As part of this thesis work we looked into alternatives, but among all the related works found, none was capable of all the features need. Every approach that is able to consider the change of constraints was neither real-time capable nor able to consider velocity and jerk constraints. This is why we tried to develop another trajectory generator based on the concept of dynamic scaling. This approach is briefly described as a further idea in Sec. 7.2.2.
- The target position-based OTG only takes symmetric motion constraints as input. This limitation is shown when we compare the limits in Fig. 6.7 with the limits in Fig. 6.8. The OTG Type IV concept would allow for asymmetric constraints, but it was only implemented for symmetric ones.

- The OTG Type IV algorithms do not allow to define target accelerations unequal to zero, which means that the motion has to always end with zero acceleration on all joints. Only the OTG Type V allows to also define an acceleration unequal to zero. The implementation of this Type of OTG would probably take several months of full-time work, but it could be worth the effort: An initial motion trajectory could be split in several parts and for each part the OTG Type V could be run separately talking into account the acceleration capabilities valid for that part.

7.1.3. Interrelation of Dynamics Modelling and Trajectory Generation

- The so-called "TAO dynamics engine" [37] was used for the implementation on the LWR. An even faster implementation of a dynamics engine combined with a more powerful computer would allow to predict the acceleration capabilities of more future motion states within one real-time cycle. This could enhance the DOTG algorithm with more possibilities when choosing which path to take towards the target motion state.
- In the context of this thesis work it appeared most suitable to describe the acceleration capabilities along a prescribed path (cf. Sec. 4.1). But other approaches could be taken to do so, for example the approach described briefly in Sec. 3.2. As long as we consider the OTG as the trajectory generator to be used, it essentially comes down to finding a more efficient way to describe the acceleration capabilities so that they can be directly used without any conversion or merging steps as acceleration constraints for the OTG.
- Instead of a placing a strong emphasis on constrained trajectory generation for position, velocity and acceleration, we could consider a pure path planning approach as described in Sec. 3.2.4: A graph search along many configurations in the full workspace of the robot. The description of the parallelotopes at every configuration can be simplified using the jerk hypercubes approach of Sec. 3.2.2 and better described with features identification idea of Sec. 3.2.3. This graph search approach would allow to find a global optimal path in space to go from start to target, but would take into consideration the velocity and acceleration progression.
- Optimizing the trajectory generation of existing robots for a better use of its acceleration capabilities is one way to deal with the problem. If we take one step back and consider to create new designs for robots, we could optimize it for its acceleration capabilities. How such a design optimization tool could look like is described in Sec. 7.2.1.
- It is obvious that if at every time step a new set of trajectories would have to be planned, this would eventually result in no functional planning, a problem which has already been described in the introduction of this thesis (cf. Fig. 1.8). This is why we have chosen for the PDOTG a conservative approach when calculating the acceleration constraints: the Future Motion State Principle applied at Step. 12 of the PDOTG loop. This approach allowed to plan ahead in time, so that before we actually arrive at the target motion state, we already know how much acceleration capabilities will then be available. When I compare the limits used for the PDOTG shown in Fig. 6.8 with the real limits shown in Fig. 6.8, we have to admit, that there is still a lot of unused potential left. There is a lot of room for improvements to the PDOTG approach, but for right now the proposed approach is better than what was there before.

7.2. Further Ideas

A few further ideas were looked into during this thesis work. The first idea is a user interface to describe and manipulate the acceleration capabilities of the workspace of a robot by interactively switching the parameters. The second idea is a new approach for trajectory generation that allows for variable motion constraints.

7.2.1. Design Optimization Tool for Acceleration Capabilities

The parallelotope description of Sec. 3.2.1 is not only useful when defining the acceleration constraints of trajectories, but also when designing new robots. New robot designs could be optimized for better acceleration

capabilities in specific areas of their workspace.

The visualization of the parallelotopes can be fully parametrized and then interactively adjusted using the "Manipulate" function of the Mathematica Software application. The environment can be freely configured and therefore allows to set up individual views for the acceleration capabilities, both in joint space and operational space.The visualization displays the parallelotopes for the end-effector positions evenly distributed over the workspace. A sidebar control to manipulate the parameters allows for manual design optimizations. Such an example is given in Fig. 7.1, the shown applet allows to change the length, mass and other parameters of the robot while updating at a high rate the visualization on the right.

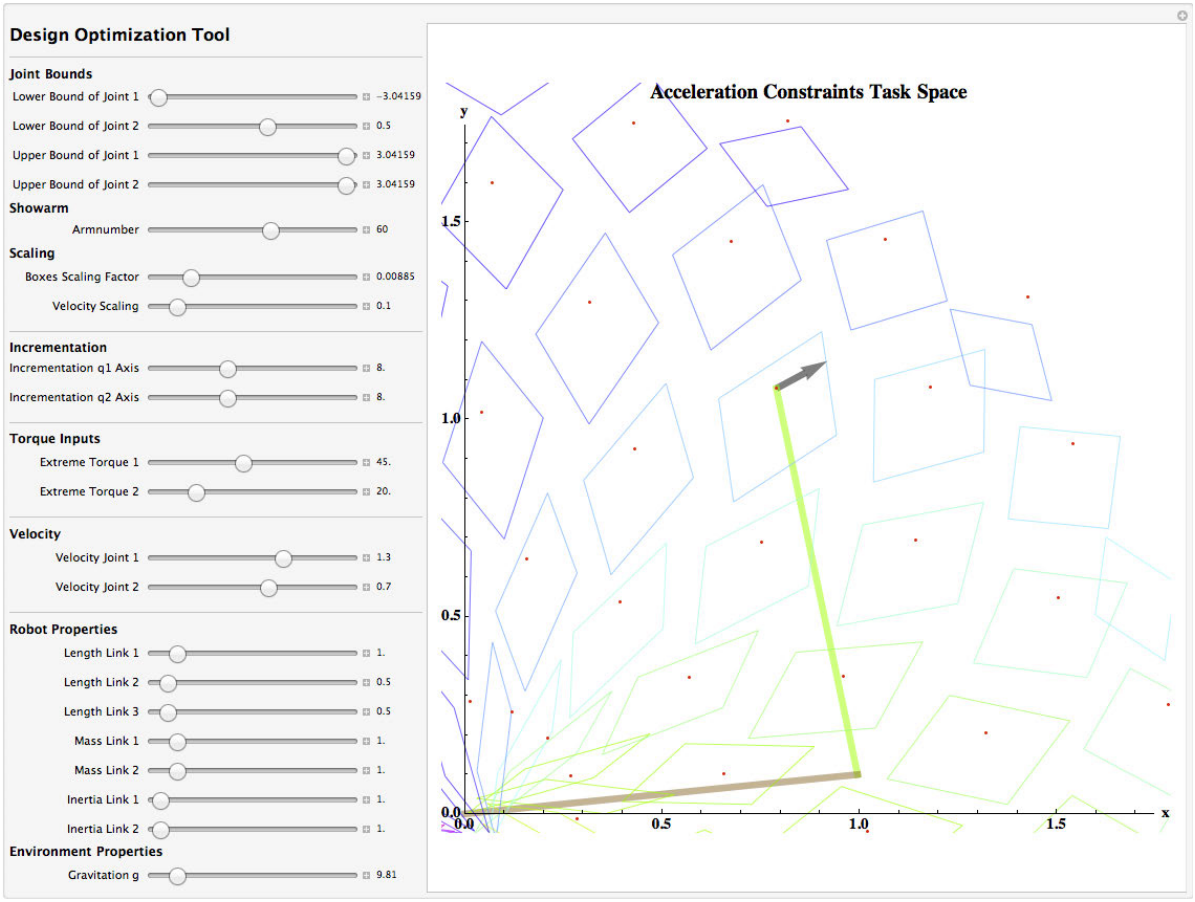


Figure 7.1.: Design tool to manually optimize the acceleration capabilities of an end-effector decribed in 2D cartesian space.

A set up in operational space would be split into two views. The first view would show 3D parallelotopes displayed for the end-effector positions of the robot while assuming a fixed orientation of the end-effector. The second represents the parallelotopes for the variation of orientations at a fixed position.

The focus of this thesis work is on the combination of acceleration capabilities with online trajectory generation, this is why no further elaboration was done on the design optimization tool. This can be seen as future work.

7.2.2. Adjustable Splines

The limitations of the OTG as described in Sec. 7.1.2 raised the question whether we can develop a new real-time capable approach that has less limitations in regards to allowing variable acceleration constraints as input. After the review of various approaches as shown in Sec. 2.1.1 and 2.1.2), the combination of the Dynamic Scaling

concept (cf. Sec. 5.2.2 of [2]) and the Cubic Splines concept (cf. Sec. 4.4.6 of [2]) appeared to be the best foundation to start with.

Overview

The new concept is called "Adjustable Splines". It is a real-time capable method that takes several motions states and time instants from an initial OTG trajectory, fits it into a cubic spline, calculates the variable acceleration constraints and scales every section of the spline individually so that it touches the motion constraints. It repeats this for n steps till the trajectory execution time converges to a value. The acceleration trajectories of step 1 and after n steps are shown in Fig. 7.2. The algorithm was only working for 1DOF.

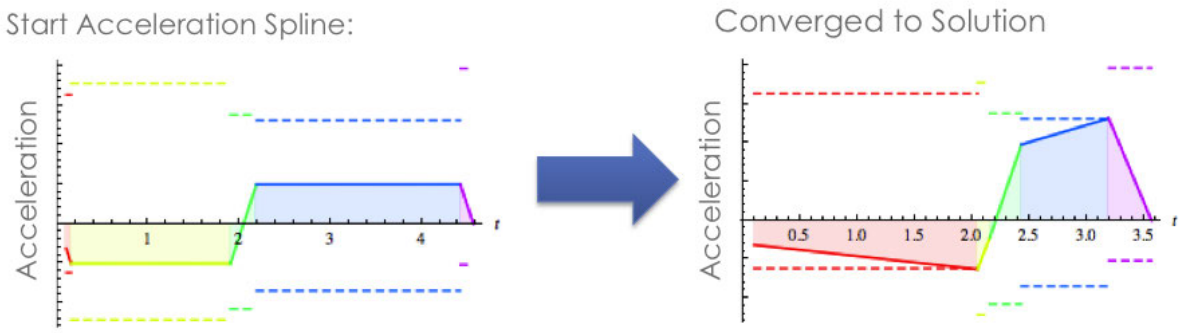


Figure 7.2.: The beginning and the converged solution of an adjustable spline.

The difference between this new approach and older approaches using time scaling as described in [17] and [2] is that we use the OTG for an initial solution, we do not assume to have constant acceleration constraints along every spline section and we are able to calculate in real-time for every section the acceleration constraints and the minimum and maximum values of the trajectory values for the velocities, accelerations and jerks.

Algorithm in Detail

1. The start and target motion state and the velocity and jerk constraints are given. We also have the torque capabilities and a dynamic model set up.

2. For the start motion state, we update the dynamic model and calculate the acceleration constraints according to Sec. 4.3.2.

3. We use the given and calculated values of Step 1 and 2, whereas the acceleration constraints are redefined by dividing it with a safety factor with for example the value 2. Using these values, we do an initial OTG run to receive a motion trajectory from start to target.

4. We define a start time t_{start} (e.g. $t_{start} = 100ms$), where our adjustable spline method will begin. For the time steps before, the OTG continues to run by himself using only the redefined acceleration constraints till t_{start} is reached.

5. We choose $k+1$ motion states along the motion trajectory by only taking the $k-1$ jerk switching points of the initial motion trajectory by the OTG and the start state state at t_{start} and the target motion state. We then define a Cubic splines of k-th order. The chosen motion states and the according time instants are now the building nodes for the cubic spline. The first node is the motion state at t_{start} and the last node is the target motion state. The cubic spline formula can be found in Sec. 4.4.4 of [2]. A cubic spline would only allow to define a start and end position while the start and end velocity and acceleration are implicitly determined through defined intermediate motion states. We would therefore not be able to fully set our first motion state and target motion state as a node. In order to define start and final velocities and accelerations for a cubic spline, two free nodes/points on the spline have to be defined: the second first and the second

last node will not be defined by the according motion states, but only implicitly defined by having new conditions: explicitly setting the velocity and acceleration for start and end node. The motion states picked for the second first and second last node are completely disregarded.

6. We calculate for every section the minimum and maximum acceleration constraints by using the motion state at the middle of the section and at the start. The other motion constraints (velocity and jerk) are constant and were already defined at the start of the algorithm. After this, we have for every spline section a velocity, acceleration and jerk trajectory and lower and upper constraint.

7. We consider every section individually. Every section has a trajectory and constraints for velocity, acceleration and jerk. For the velocity, acceleration and jerk individually, we calculate the quotient of the lowest constraint value of that section and the maximal absolute trajectory value of that section. We receive three quotients per section, one for the velocity, one for the acceleration and one for the jerk.

8. The inverted quotient for the velocity is the time scaling factor. The time scaling factor is the value needed to adjust the velocity trajectory of the viewed spline section so that the trajectory would touch at least at one point one of the velocity constraints (lower or upper constraint). For the acceleration constraint, we first have to pull the square root of its inverse to also get the scaling factor for the acceleration trajectory. The jerk constraint requires us to pull the cubic root of its inverse to get the scaling factor. The mathematical explanation to this is given on p.191 of [2].

9. The three time scaling factors found for the current section are now compared with each other: we take the smallest of the three values and save this value as the scaling factor for the spline section which we currently consider. We repeat this whole procedure for the remaining spline sections till we have k time scaling factors.

10. We apply the time scaling factors to every individual section. The position trajectory will stays continuous at the section transitions, but the velocity and acceleration trajectory will not be continuous anymore at the transition from one section to another. The reason for this is that every section was scaled with generally a different factor.

11. After that spline time scaling, we get new times for every spline node and the total execution time of the spline did therefore also change. Based on these new times, but keeping the motion states for every node, we calculate again a cubic spline analog to what was done in Step 5.

12. Based on the new execution time which we received after the time scaling and which remained constant after recalculating the spline, we check if it is within a Δt of the execution time in the step before. If yes, then we go to the next step, if no, we go back to Step 6.

13. The Adjustable Spline algorithm converged to a result. The new cubic spline calculated in Step 11 is used as the trajectory after t_{start} is reached.

Results and Discussions

The result of the first spline interpolation can be seen in Fig. 7.3. Every color describes a new spline section. The upper and lower limit lines are the constraints.

After k steps, a new spline interpolation looks like in Fig. 7.4.

The variation in execution time after k steps is within a Δt, we can break up at that point and take this trajectory as a suitable solution.

The algorithm run most of the times stable for one DOF and provided after 20-30 iterations a converged result, as shown for example in Fig. 7.4. This trajectory generator approach seemed promising but was not always stable for 1 DOF. The algorithm did not always converge to a constant result but sometimes diverged after a few steps and became unstable. For multiple DOFS, the approach has also been implemented in Mathematica, but it turned out to never lead to a stable convergence after several steps.

After we redefined the start and target velocity and acceleration to be zero, we tried to see if the algorithm becomes always stable for 1 DOF, but this did not make a difference. A possible issue could be certain sequences

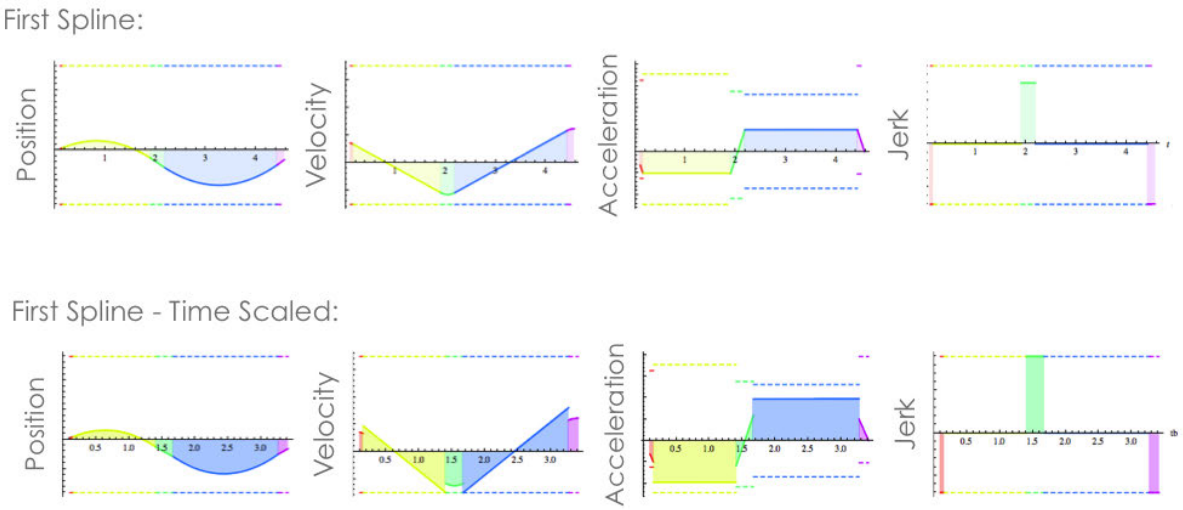


Figure 7.3.: The first iteration of an adjustable spline.

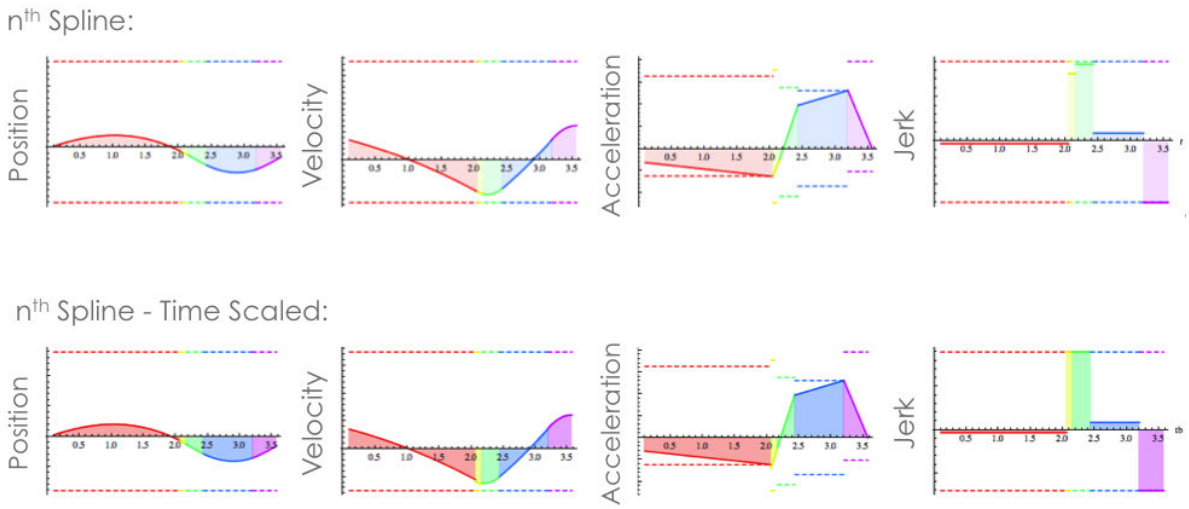


Figure 7.4.: The n-th adjustable spline.

of variable acceleration constraints. One of the new aspects of this algorithm is, that we have variable acceleration constraints for every section. The adjustment of the splines sometimes has issues if there are sudden jumps in the acceleration constraints at certain parts of the acceleration profile.

One possible remedy to this instability based on the variable acceleration constraints could be the following (not tested yet): Instead of using constant acceleration constraint values per section, we define linearly changing values for every section. This would lead to a continuous acceleration constraint profile which would go up and down linearly. The time scaling of the acceleration profiles, which are also first order polynomials, would then better fit into the constraints without having to deal with sudden jumps of the constraints. The implementation of this remedy is not straight forward and since the focus of this thesis was on finding a working solution, we proceeded with the combination of the OTG and the Path Dynamics.

The features of the Adjustable Spline Method are:

Functionality

- Variable motion constraints
- Most of the time stable for 1 DOF
- Real-Time suitable execution time

Limitations

- Not as fast as the OTG
- Sometimes unstable for 1 DOF
- Unstable for multiple DOF

8. Conclusion and Outlook

8.1. Conclusion

The purpose of this thesis was to combine online trajectory generation with the calculation of robot acceleration capabilities. The work started with a thorough investigation into existing trajectory generation algorithms, how do they differ and which limitations are given. Parallel to this investigation, we also looked deeper into how to describe and visualize acceleration capabilities of robots. After the basics and related works were clear, a general roadmap leading to a Dynamic Online Trajectory Generator (DOTG) was laid out. An Overview of this roadmap is shown in Fig. 8.1. The first part of this thesis was to fully understand and solve the dependencies between the acceleration capabilities and online trajectory generation algorithms. The second part was to find a proper connection of both fields to obtain a set of useful DOTG algorithms, using the target velocity-based and target position-based versions of the OTG. The third an final part was the implementation of the developed algorithms on a robot manipulator.

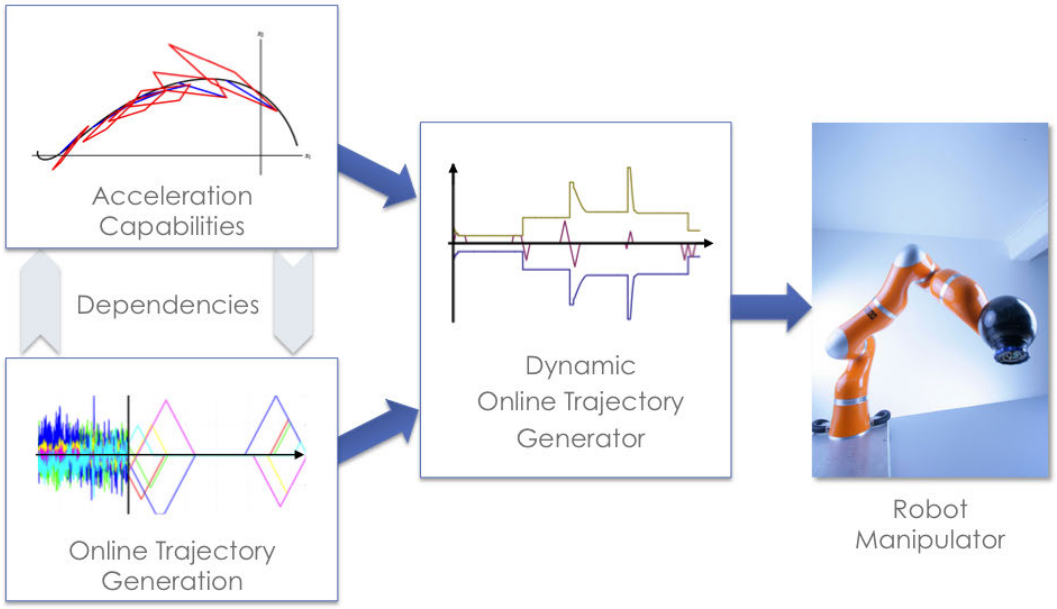


Figure 8.1.: Dynamic OTG development overview.

The acceleration capabilities can essentially be described as parallelotopes and then further analyzed using two different approaches. The first approach consists of size reduction and feature identification of parallelotopes applied to large amounts of configurations in a robot's workspace. Based on this vast data set, a graph search leads to optimal path planning results, but is not suitable for online trajectory generation. That is why this approach was explained but not further elaborated in this thesis. The second approach for analyzing the parallelotopes is the idea of acceleration capabilities along a path. We split the acceleration into a normal and tangential part and then identify the acceleration capabilities at the intersections of the tangential acceleration lines with the parallelotope shape.

The online trajectory generation consists of two parts in context of this thesis. Either use the existing class of Online Trajectory Generators by [24, 21, 23] or enhance another, more suitable trajectory generator for the

combination task. The Adjustable Splines method has been newly developed, but proved itself to be insufficient as a multiple DOF trajectory generator. The use of the OTG was finally the method of choice to proceed with.

The next step was then the combination of both by identifying the dependencies and interfaces between the two fields. In order to obtain suitable acceleration constraints for use as input to the OTG, the real acceleration capabilities were converted and merged with the axes acceleration capabilities for useful values. This procedure is the new Path Dynamics Merging Algorithm. The combination of this algorithm with the existing target velocity-based OTG emerged in two versions of a VDOTG, whereas only the second approach was successfully implemented. The addition of the Future Motion State Principle then allowed to develop based on the VDOTG the PDOTG, which proved itself in experiments as fully capable to consider robot acceleration capabilities in online trajectory generation.

The VDOTG shows on the one hand full capability in using the acceleration capabilities of a robot, but its use cases are rather limited. Only emergency braking scenarios seem meaningful for this type of algorithm. The PDOTG on the other hand is not able to make full use all of the available acceleration capabilities. The reason for this are the limitations of the OTG in its target position-based version. It can not take asymmetric motion constraints as input and does not allow to define a target acceleration state unequal to zero. If both limitations would not exist, an almost perfect PDOTG could be created

8.2. Outlook

As already mentioned earlier, the Adjustable Splines method has not shown itself as a stable trajectory generator. Further tests and implementations of variations to the basic concept might lead to a useful algorithm that runs stable not only for 1 DOF, but also for several DOF. The method should not yet be given up.

The further development of the Design Optimization Tool could become useful for future robot projects where a strong emphasis on the acceleration capabilities of system is needed. The DOTG can perform even better if the used robot is optimized for better acceleration capabilities in the respective workspace.

Further tests and experiments will be run with the second version of the VDOTG approach in order to show its stability and application. The reasons for the instability of the first version of the VDOTG are not clear yet, only assumptions were made on why this approach did not work. Further evaluations on the times the algorithm suddenly destabilizes are needed.

Also further development of the Future Motion State Principle used in the PDOTG algorithm should be considered. Alternative approaches my be useful to deal with the problem of having to plan ahead in time. A multi-threaded applications which can process the acceleration capabilities for several future states and merge the results might be another approach to take. In general, a new approach should be able to make even more use of acceleration potentials not used by the for right now still relatively conservative PDOTG algorithm.

The implementation of the the OTG Type V would be the ultimate step to take in order to create a totally new class of DOTGs. The possibility to set a target acceleration opens new possibilities on how to design the DOTG loops and deal with the acceleration capability changes. We could split an initial trajectory solution in several parts and optimize every section individually for its acceleration constraints. This approach would only work with an OTG Type V.

A. Abbreviations and Symbols

The following lists encompasses most of the abbreviations and symbols used in the thesis except for those, who only occur within one section.

a_j	describing polynomial parameters with $j \in \{1,2,3\}$
$\boldsymbol{\alpha}$	short for $\boldsymbol{M}(\boldsymbol{q})\ \boldsymbol{r}_s$
$\tilde{\boldsymbol{\alpha}}$	short for $\boldsymbol{\Lambda}(\boldsymbol{x})$
$\hat{\boldsymbol{\alpha}}_l$	short for $\boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{e}_l$ for $\forall l \in (1,\ldots,K)$
$\boldsymbol{B}_i$	kinematic-motion constraints at t_i
$\boldsymbol{\beta}$	short for $\boldsymbol{\tau}_g(\boldsymbol{q}) + c(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{M}(\boldsymbol{q})\ \boldsymbol{r}_{ss}\ \dot{s}^2$
$\tilde{\boldsymbol{\beta}}$	short for $\boldsymbol{p}(\boldsymbol{x}) + \boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}}) + \boldsymbol{\Lambda}(\boldsymbol{x})\,\boldsymbol{r}_{ss}\ \dot{s}^2$
$\hat{\boldsymbol{\beta}}$	short for $\boldsymbol{\tau}_g(\boldsymbol{q}) + c(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}$
$\boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}})$	Coriolis and centrifugal torques, depending on the position and velocity
Δt	time shift for polynomial description
DOF	Degrees of freedom
DOTG	Dynamic Online Trajectory Generator
$\boldsymbol{e}_l$	unit vector along each axis with $l \in (1,\ldots,K)$
η_i	time at step i for first version of VDOTG
$\boldsymbol{f}$	force exerted on the end-effector
$\gamma_{i,min}$	maximum element of $\boldsymbol{\rho}_{i,min}$
$\gamma_{i,max}$	minimum element of $\boldsymbol{\rho}_{i,max}$
${}_kh$	upper limit of joint k for keeping manipulator on joint path
${}_k\tilde{h}$	upper limit of joint k for keeping manipulator on operational space path
${}_k\hat{h}_l$	upper limit of joint k for acceleration values ν_l along axis $\boldsymbol{e}_l$
$\boldsymbol{J}$	Jacobian Matrix
$\overline{J}^T$	Dynamically consistent generalized inverse of the Jacobian Matrix
k_{exp}	is an expansion factor
K	total number of degrees of freedom of a system
l	index describing which polynomial of the piecewise collection is meant
L	required total number of polynomials
$L(\boldsymbol{0},\ddot{\boldsymbol{q}}_{given})$	Ray passing through the frame origin and the acceleration $\ddot{\boldsymbol{q}}_{given}$
${}_kl$	lower limit of joint k for keeping manipulator on joint path
${}_k\tilde{l}$	lower limit of joint k for keeping manipulator on operational space path
${}_k\hat{l}_l$	lower limit of joint k for acceleration values ν_l along axis $\boldsymbol{e}_l$
κ	index of the DOF determining the execution time of the trajectory
$\boldsymbol{\Lambda}(\boldsymbol{x})$	operational space mass matrix depending on $\boldsymbol{x}$
$\lambda(q)$	inverse function for lookup on original trajectory
$\boldsymbol{M}(\boldsymbol{q})$	symmetric, positive-definite mass matrix, depending on the position
OTG	Online trajectory generator/generation
$\boldsymbol{\mu}(\boldsymbol{x},\dot{\boldsymbol{x}})$	vector of the velocity-product depending on $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$
$\boldsymbol{\nu}_l$	axis vector with variable scalar value
ν_l	scalar value of vector $\boldsymbol{\nu}_l$ along axis $\boldsymbol{e}_l$
$\nu_{l,min}$	admissible lower value for ν_l
$\nu_{l,max}$	admissible upper value for ν_l
$\boldsymbol{p}(\boldsymbol{x})$	gravity forces depending on $\boldsymbol{x}$
PDOTG	Target Position-Based Dynamic Online Trajectory Generator

$\boldsymbol{\Phi}_i(t)$	complete motion trajectory composed of sets of motion polynomials and time intervals
PT	Parallelotope
$\boldsymbol{q}$	joint positions
$\dot{\boldsymbol{q}}$	joint velocities
$\ddot{\boldsymbol{q}}$	joint accelerations
$\ddot{\boldsymbol{q}}_{min,axes}$	minimal axes acceleration limits
$\ddot{\boldsymbol{q}}_{max,axes}$	minimal axes acceleration limits
$\dot{\boldsymbol{q}}_i$	joint positions at t_i
$\ddot{\boldsymbol{q}}_i$	joint velocities at t_i
$\dddot{\boldsymbol{q}}_i$	joint accelerations at t_i
$\underline{q}_i$	joint position for reference DOF κ (κq_i)
$\underline{\dot{q}}_i$	joint velocities for reference DOF κ ($\kappa \dot{q}_i$)
$\underline{\ddot{q}}_i$	joint accelerations for reference DOF κ ($\kappa \ddot{q}_i$)
${}^l\boldsymbol{q}_i(t)$	Polynomial position vector with index l
${}^l\dot{\boldsymbol{q}}_i(t)$	first derivation of ${}^l\boldsymbol{q}_i(t)$
${}^l\ddot{\boldsymbol{q}}_i(t)$	second derivation of ${}^l\boldsymbol{q}_i(t)$
${}^l\dddot{\boldsymbol{q}}_i(t)$	third derivation of ${}^l\boldsymbol{q}_i(t)$
${}_k q_i$	joint position of the k-th DOF at t_i
${}^l_k q_i(t)$	third-order polynomial with index l for DOF k calculated at time t_i and depending on t
$\ddot{\boldsymbol{q}}_{extr}$	extreme joint accelerations
$\ddot{\boldsymbol{q}}_{given}$	given acceleration state
$\dot{\boldsymbol{q}}_{i,max}$	maximal joint velocity constraints
$\dot{\boldsymbol{q}}_{i,min}$	minimal joint velocity constraints
$\ddot{\boldsymbol{q}}_{i,max}$	maximal joint acceleration constraints
$\ddot{\boldsymbol{q}}_{i,min}$	minimal joint acceleration constraints
$\dddot{\boldsymbol{q}}_{i,max}$	maximal joint jerk constraints
$\dddot{\boldsymbol{q}}_{i,min}$	minimal joint jerk constraints
$\ddot{\boldsymbol{q}}_{max,path}$	maximal path acceleration capabilities
$\ddot{\boldsymbol{q}}_{min,path}$	minimal path acceleration capabilities
$\ddot{\boldsymbol{q}}_{max,abs}$	absolute maximal joint space acceleration capabilities
$\ddot{\boldsymbol{q}}_{min,abs}$	absolute minimal joint space acceleration capabilities
$\ddot{\boldsymbol{q}}_{min}$	used minimal joint space optimized acceleration constraints
$\ddot{\boldsymbol{q}}_{max}$	used maximal joint space optimized acceleration constraints
$\boldsymbol{r}(s(t))$	Alternative position trajectory representation
$\boldsymbol{r}_{ss}(s)$	second partial derivation of $\boldsymbol{r}(s)$ by s
$\boldsymbol{r}_s(s)$	first partial derivation of $\boldsymbol{r}(s)$ by s
$\boldsymbol{\rho}_{i,min}$	ratio between $\ddot{\boldsymbol{q}}_{0,min}$ and $\ddot{\boldsymbol{q}}_{i,min}$
$\boldsymbol{\rho}_{i,max}$	ratio between $\ddot{\boldsymbol{q}}_{0,max}$ and $\ddot{\boldsymbol{q}}_{i,max}$
RML	Reflexxes Motion Libraries
$\ddot{s}_{min}$	Admissible lower value for tangential acceleration
$\ddot{s}_{max}$	Admissible upper value for tangential acceleration
$s(t)$	path describing the arc length of a continuous position progression
$\boldsymbol{S}_i$	OTG boolean selection vector determining controlled DOF
t_{cycle}	cycle time of the system
$t_{i,fut}$	future time instant
t_i	time instant i
$t_{i,sync}$	synchronization time when $\boldsymbol{\Xi}_{i,trgt}$ is reached
${}^l_k t_i$	start time of polynomial l for DOF k
$\boldsymbol{\tau}$	actuator torques
$\boldsymbol{\tau}_{extr}$	extreme actuator torques
$\boldsymbol{\tau}_g(\boldsymbol{q})$	gravity torques, depending on the position
${}_k\tau_{max}$	maximal actuator torque for DOF k

${}_k\tau_{min}$	minimal actuator torque for DOF k
${}^l_k\vartheta_i$	time-interval
${}^l\curlyvee_i$	set of time-intervals
VDOTG	Target Velocity-Based Dynamic Online Trajectory Generator
$\boldsymbol{W}_i$	OTG input parameters matrix
$\boldsymbol{x}$	operational coordinates, describing end-effector position and orientation
$\boldsymbol{x}(\boldsymbol{q}(t))$	multidimensional position trajectory depending on the joint positions $\boldsymbol{q}(t)$
$\dot{\boldsymbol{x}}$	first derivation of the operational coordinates, operational-space velocity
$\ddot{\boldsymbol{x}}$	second derivation of the operational coordinates, operational-space acceleration
$\ddot{\boldsymbol{x}}_{extr}$	extremal operational space accelerations
$\ddot{\boldsymbol{x}}_{max,path}$	maximal path acceleration capabilities
$\ddot{\boldsymbol{x}}_{min,path}$	minimal path acceleration capabilities
${}_k\ddot{x}_{max,abs}$	absolute maximal operational space acceleration capabilities
${}_k\ddot{x}_{min,abs}$	absolute minimal operational space acceleration capabilities
$\boldsymbol{\Xi}_i$	motion state at t_i
${}_k\boldsymbol{\xi}_i$	k-th row of the motion state matrix $\boldsymbol{\Xi}_i$
$\boldsymbol{\Xi}_{i,trgt}$	desired target motion state at t_i
${}^l\boldsymbol{\Xi}_i(t)$	matrix of polynomials
${}^l_k\xi_i(t)$	k-th row of ${}^l\boldsymbol{\Xi}_i(t)$

B. Appendix

B.1. Calculation of the path representation numerically

In order to find out how to calculate $\dot{s}$, $\ddot{s}$, $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}$, we need to look into the definitions of $s(t)$ and $\boldsymbol{r}(s)$. The values for $\dot{s}$, $\ddot{s}$, $\boldsymbol{r}_s$ and $\boldsymbol{r}_{ss}$ can be found using the central difference quotient and a little variation of it, as shown in the next two sections.

Path $s(t)$ by arc length and its derivations

The definition of the path defined by its arc length is:

$$s(t) = \int_{t_0}^{t} ||\dot{\boldsymbol{q}}(t)||\, dt \tag{B.1}$$

The velocity progression of the trajectory computed by the OTG algorithm is defined as a piecewise function of polynomials. These polynomials can be up to second degree order. The piecewise time intervals are not the same for every DOF. It is therefore not possible to solve the integral in Eqn. (B.1) in an analytical way.

Discretizing Eqn. (B.1) leads to the following numerical approximation:

$$\begin{aligned} s(t_0) &= s_0 &&= 0 \\ s(t_i) &= s_i &&\approx \sum_{j=1}^{i} \frac{||\dot{\boldsymbol{q}}_j|| + ||\dot{\boldsymbol{q}}_{j-1}||}{2} \cdot t_{cycle} \end{aligned} \tag{B.2}$$

In this first discretization description of the path length we used t_{cycle} as the discretization time. This is a rough approximation for the path length. Since we can calculate the velocity also at times between the time instants t_i, we can find a better approximation for the path length using the approximation time t_{app} with the properties

$$t_{app} << t_{cycle} \tag{B.3}$$

$$t_{cycle} \bmod t_{app} = 0 \tag{B.4}$$

Using t_{app}, we can do a better approximation of Eqn. (B.1) than with Eqn. (B.2):

$$\begin{aligned} s(t_0) &= s_0 &&= 0 \\ s(t_i) &= s_i &&\approx \sum_{j=1}^{i \cdot \frac{T_{cycle}}{t_{app}}} \frac{||\dot{\boldsymbol{q}}(j \cdot t_{app})|| + ||\dot{\boldsymbol{q}}((j-1) \cdot t_{app})||}{2} \cdot t_{app} \end{aligned} \tag{B.5}$$

The central difference quotient is used to find the derivation of s_i:

$$\dot{s}_i = \frac{s(t_i + t_{app}) - s(t_i - t_{app})}{2 \cdot t_{app}} \tag{B.6}$$

The dividend of Eqn. (B.6), which is the path value difference for time step t_i, can be abbreviated by:

$$\Delta s_i := s(t_i + t_{app}) - s(t_i - t_{app}) \tag{B.7}$$

Derived from Eqn. (B.5), we can calculate

$$\begin{aligned} \Delta s_i &\approx \frac{||\dot{\boldsymbol{q}}(t_i)|| + ||\dot{\boldsymbol{q}}(t_i - t_{app})||}{2} \cdot t_{app} + \frac{||\dot{\boldsymbol{q}}(t_i + t_{app})|| + ||\dot{\boldsymbol{q}}(t_i)||}{2} \cdot t_{app} \\ &= \frac{||\dot{\boldsymbol{q}}(t_i + t_{app})|| + 2\,||\dot{\boldsymbol{q}}(t_i)|| + ||\dot{\boldsymbol{q}}(t_i - t_{app})||}{2} \cdot t_{app} \end{aligned} \tag{B.8}$$

Using Eqn. (B.7) and(B.8) simplify Eqn. (B.6) to

$$\dot{s}_i = \frac{||\dot{\boldsymbol{q}}(t_i + t_{app})|| + 2\,||\dot{\boldsymbol{q}}(t_i)|| + ||\dot{\boldsymbol{q}}(t_i - t_{app})||}{4} \tag{B.9}$$

We find the current path acceleration $\ddot{s}_i$ analog to Eqn. (B.6) $\sim$ (B.9). The current path acceleration $\ddot{s}_i$ is defined by the central difference quotient as:

$$\begin{aligned} \ddot{s}_i &\approx \frac{\dot{s}(t_i + \frac{t_{app}}{2}) - \dot{s}(t_i - \frac{t_{app}}{2})}{t_{app}} \\ &= \frac{\frac{s(t_i + t_{app}) - s(t_i)}{t_{app}} - \frac{s(t_i) - s(t_i - t_{app})}{t_{app}}}{t_{app}} \\ &= \frac{(s(t_i + t_{app}) - s(t_i)) - (s(t_i) - s(t_i - t_{app}))}{{t_{app}}^2} \end{aligned} \tag{B.10}$$

The two terms in the dividend of Eqn. (B.10), can be abbreviated by:

$$\Delta s_{i,forw} := s(t_i + t_{app}) - s(t_i) \tag{B.11}$$

$$\Delta s_{i,back} := s(t_i) - s(t_i - t_{app}) \tag{B.12}$$

Derived from Eqn. (B.5), we can calculate

$$\Delta s_{i,forw} := \frac{||\dot{\boldsymbol{q}}(t_i + t_{app})|| + ||\dot{\boldsymbol{q}}(t_i)||}{2} \cdot t_{app} \tag{B.13}$$

$$\Delta s_{i,back} := \frac{||\dot{\boldsymbol{q}}(t_i)|| + ||\dot{\boldsymbol{q}}(t_i - t_{app})||}{2} \cdot t_{app} \tag{B.14}$$

Applying Eqn. (B.13) and (B.14) to Eqn. (B.10) yields the path acceleration $\ddot{s}_i$:

$$\ddot{s}_i \approx \frac{||\dot{\boldsymbol{q}}(t_i + t_{app})|| - ||\dot{\boldsymbol{q}}(t_i - t_{app})||}{2\,t_{app}}. \tag{B.15}$$

Note that when applying the central difference quotient for time step t_0, we need to know the velocity value just before the DOTG was started, which is $\dot{\boldsymbol{q}}(t_0 - t_{app})$.

We derived now how to calculate $\dot{s}_i$. Now we also need to derive how to calculate $\boldsymbol{r}_s(s_i)$ and $\boldsymbol{r}_{ss}(s_i)$.

The path vector $\boldsymbol{r}(s(t))$ and its derivations

At first we will describe, how you would normally find the numerically derivation of $\boldsymbol{r}(s(t))$. Unfortunately, this approach does not work, but we propose a slightly different approach.

The first derivation of $\boldsymbol{r}_i$ would normally be found using the standard definition of the central difference quotient:

$$\boldsymbol{r}_{si} \approx \frac{\boldsymbol{r}\left(s_i + \Delta s_{i,back}\right) - \boldsymbol{r}(s_i - \Delta s_{i,back})}{2 \cdot \Delta s_{i,back}} \tag{B.16}$$

$\Delta s_{i,back}$ describes a small change of the path variable $s(t)$. We randomly choose $\Delta s_{i,back}$ defined in Eqn.(B.14),

but we could have also taken $\Delta s_{i,forw}$, defined in eq (B.11), as the delta value.

According to Eqn. (4.25), the path vector can be found by

$$\boldsymbol{r}(s(t)) \quad = \quad \boldsymbol{q}(t) \tag{B.17}$$

In order to calculate $\boldsymbol{r}(s_i - \Delta s_{i,back})$, we first look at its argument $s_i - \Delta s_{i,back}$ and simplify it:

$$s_i - \Delta s_{i,back} = s(t_i) - (s(t_i) - s(t_i - t_{app})) = s(t_i - t_{app}) \tag{B.18}$$

$s(t_i - t_{app})$ describes a path value of a known time instant. Therefore

$$\boldsymbol{r}(s_i - \Delta s_{i,back}) = \boldsymbol{r}(s(t_i - t_{app})) = \boldsymbol{q}(t_i - t_{app}) \tag{B.19}$$

We can calculate $\boldsymbol{r}(s_i - \Delta s_{i,back})$ and $\Delta s_{i,back}$, what is left is the term $\boldsymbol{r}\left(s_i + \Delta s_{i,back}\right)$ in Eqn. (B.16). Restating its argument yields to

$$s_i + \Delta s_{i,back} = s(t_i) + (s(t_i) - s(t_i - t_{app})) = 2s(t_i) - s(t_i - t_{app}) \tag{B.20}$$

$s_i + \Delta s_{i,back}$ does not simplify so that we can solve it, because we do not know the absolute path values, we can only define the local deltas.

We can also write:

$$\Delta s_{i,back} \quad \approx \quad \frac{||\dot{\boldsymbol{q}}(t_i + t_{unknown})|| + ||\dot{\boldsymbol{q}}(t_i)||}{2} \cdot t_{unknown} \tag{B.21}$$

This equation has the unknown $t_{unknown}$, appearing as itself and as part of the argument of $||\dot{\boldsymbol{q}}||$. Since it is numerically expensive to solve this equation, we propose another approach:

We find the first derivation of $\boldsymbol{r}_i$ by slightly modifying the central difference quotient of Eqn. (B.16):

$$\boldsymbol{r}_{si} \quad \approx \quad \frac{\boldsymbol{r}(s(t_i + t_{app})) - \boldsymbol{r}(s(t_i - t_{app}))}{s(t_i + t_{app}) - s(t_i - t_{app})} \tag{B.22}$$

Instead of varying s in both directions with a constant Δs, we vary it in both directions with a constant Δt. Since t and s are both monotonously increasing and one-to-one, this modification is valid.

Using the definition in Eqn. (B.7) simplifies Eqn. (B.22) to:

$$\boldsymbol{r}_{si} \quad \approx \quad \frac{\boldsymbol{r}(s(t_i + t_{app})) - \boldsymbol{r}(s(t_i - t_{app}))}{\Delta s_i} \tag{B.23}$$

Using the relation in Eqn. (B.17) simplifies Eqn. (B.23) to

$$\boldsymbol{r}_{si} \quad \approx \quad \frac{\boldsymbol{q}(t_i + t_{app}) - \boldsymbol{q}(t_i - t_{app})}{\Delta s_i} \tag{B.24}$$

In the same way we can find the second derivation of $\boldsymbol{r}_i$ at the time t_i:

$$\boldsymbol{r}_{ssi} \quad \approx \quad \frac{\boldsymbol{r}_s(s(t_i + \frac{t_{app}}{2})) - \boldsymbol{r}_s(s(t_i - \frac{t_{app}}{2}))}{s(t_i + \frac{t_{app}}{2}) - s(t_i - \frac{t_{app}}{2})} \tag{B.25}$$

Analog to Eqn. (B.5), we can calculate the divisor of Eqn. (B.25):

$$s(t_i + \frac{t_{app}}{2}) - s(t_i - \frac{t_{app}}{2}) \quad \approx \quad \frac{||\dot{\boldsymbol{q}}(t_i + \frac{t_{app}}{2})|| + ||\dot{\boldsymbol{q}}(t_i - \frac{t_{app}}{2})||}{2} \cdot t_{app} \tag{B.26}$$

The terms in the dividend of Eqn. (B.25) are calculated analog to Eqn. (B.22)

$$\boldsymbol{r}_s(s(t_i+\frac{t_{app}}{2})) \approx \frac{\boldsymbol{r}(s(t_i+t_{app}))-\boldsymbol{r}(s(t_i))}{s(t_i+t_{app})-s(t_i)} \tag{B.27}$$

$$\boldsymbol{r}_s(s(t_i-\frac{t_{app}}{2})) \approx \frac{\boldsymbol{r}(s(t_i))-\boldsymbol{r}(s(t_i-t_{app}))}{s(t_i)-s(t_i-t_{app})} \tag{B.28}$$

Using the relation in Eqn. (B.17), we can calculate the dividends of Eqn. (B.27) and (B.28):

$$\boldsymbol{r}(s(t_i+t_{app}))-\boldsymbol{r}(s(t_i)) = \boldsymbol{q}(t_i+t_{app})-\boldsymbol{q}(t_i) \tag{B.29}$$

$$\boldsymbol{r}(s(t_i))-\boldsymbol{r}(s(t_i-t_{app})) = \boldsymbol{q}(t_i)-\boldsymbol{q}(t_i-t_{app}) \tag{B.30}$$

Using the Eqn. (B.29) and (B.30) together with the definitions in Eqn. (B.13) and (B.14), we can simplify Eqn. (B.27) and (B.28):

$$\boldsymbol{r}_s(s(t_i+\frac{t_{app}}{2})) \approx \frac{\boldsymbol{q}(t_i+t_{app})-\boldsymbol{q}(t_i)}{\Delta s_{i,forw}} \tag{B.31}$$

$$\boldsymbol{r}_s(s(t_i-\frac{t_{app}}{2})) \approx \frac{\boldsymbol{q}(t_i)-\boldsymbol{q}(t_i-t_{app})}{\Delta s_{i,back}} \tag{B.32}$$

Finally, we apply Eqn. (B.26) $\sim$ (B.32) to Eqn. (B.25) so we can calculate $\boldsymbol{r}_{ssi}$:

$$\boldsymbol{r}_{ssi} \approx \frac{\frac{\boldsymbol{q}(t_i+t_{app})-\boldsymbol{q}(t_i)}{\Delta s_{i,forw}} - \frac{\boldsymbol{q}(t_i)-\boldsymbol{q}(t_i-t_{app})}{\Delta s_{i,back}}}{\frac{||\dot{\boldsymbol{q}}(t_i+\frac{t_{app}}{2})||+||\dot{\boldsymbol{q}}(t_i-\frac{t_{app}}{2})||}{2}\cdot t_{app}} \tag{B.33}$$

B.2. Kuka LWR - Robot and Payload Parameters

The full set of parameters for the LWR can be found in Tab. B.1. The parameters of the payload, which was used for the experiments with the LWR, are given in Tab. B.2.

B.3. Videos and Source Code Files

Demonstration videos and source code files can be found on the website: `http://research.katzschmann.eu`.

	Link 1	Link 2	Link 3	Link 4	Link 5	Link 6	End-Effector	End-Eff.+Payload
Parent link	Base	Link 1	Link 2	Link 3	Link 4	Link 5	Link 6	Link 6
Joint name	A1	A2	E1	A3	A4	A5	A6	A6
Joint type	Rotation z	Rotation z	Rotation z	Rotation z	Rotation z	Rotation z	Rotation z	Rotation z
Neg. joint position limits [rad]	-2.9671	-2.0944	-2.9671	-2.0944	-2.9671	-2.0944	-2.9671	-2.9671
Pos. joint position limits [rad]	2.9671	2.0944	2.9671	2.0944	2.9671	2.0944	2.9671	2.9671
Neg. joint velocity limits [rad/s]	-1.7453	-1.9199	-1.7453	-2.2689	-2.2689	-3.1416	-3.1416	-3.1416
Pos. joint velocity limits [rad/s]	1.7453	1.9199	1.7453	2.2689	2.2689	3.1416	3.1416	3.1416
Position in parent x [m]	0	0	0	0	0	0	0	0
Position in parent y [m]	0	0	0.2	0	-0.2	0	0.052	0.052
Position in parent z [m]	0.11	0.2	0	0.2	0	0.19	0	0
Orientation in parent x [i]	0	0.707106	-0.707106	-0.707106	0.707106	0.707106	-0.707106	-0.707106
Orientation in parent y [j]	0	0	0	0	0	0	0	0
Orientation in parent z [k]	0	0	0	0	0	0	0	0
Orientation in parent w [1]	1	0.707106	0.707106	0.707106	0.707106	0.707106	0.707106	0.707106
Mass [kg]	2.1554	2.1554	2.1554	2.1554	2.00222	1.2	0.5	5.071
Inertia Ixx [kgm2]	0.0104717	0.0104717	0.0104717	0.0104717	0.00907671	0.002352	0.000169792	0.01285
Inertia Iyy [kgm2]	0.0104717	0.00657401	0.0104717	0.00657401	0.00907671	0.002352	0.000169792	0.01285
Inertia Izz [kgm2]	0.00657401	0.0104717	0.00657401	0.0104717	0.00610676	0.002352	0.00030625	0.01474
Inertia Ixy [kgm2]	0	0	0	0	0	0	0	0
Inertia Ixz [kgm2]	0	0	0	0	0	0	0	0
Inertia Iyz [kgm2]	0	0	0	0	0	0	0	0
Center of mass x [m]	0	0	0	0	0	0	0	0
Center of mass y [m]	0.03	0.08	-0.03	-0.08	0.02	0	0	0
Center of mass z [m]	0.12	0.03	0.12	0.03	0.1	0	0.01	0.0331774

Table B.1.: LWR Parameters

Feature	LWR Flange	Payload Flange	Payload
Height	10.0 mm	12.70 mm	25.00 mm
Inner diameter	23.45 mm	23.45 mm	0.00 mm
Outer diameter	60.00 mm	76.20 mm	23.45 mm
Mass	0.220 kg	0.138 kg	4.213 kg
CoM_z	5.0 mm	6.35 mm	12.25 mm
MoI_{xx}			
MoI_{yy}			
MoI_{zz}			

Table B.2.: Used Payload

C. Bibliography

[1] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*, chapter 3, Composition of Elementary Trajectories, pages 59–150. Springer, Berlin, Heidelberg, Germany, first edition, 2008.

[2] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer, Berlin, Heidelberg, Germany, first edition, 2008.

[3] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*, chapter 5, Operations on Trajectories, pages 223–244. Springer, Berlin, Heidelberg, Germany, first edition, 2008.

[4] R. Bischoff. From research to products: The development of the KUKA light-weight robot. In *In Proc. of the 40th International Symposium on Robotics*, Barcelona, Spain, March 2009.

[5] R. Bischoff, J. Kurth, G. Schreiber, R. Köppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger. The KUKA-DLR lightweight robot arm — A new reference platform for robotics research and manufacturing. In *Proc. of the Joint Conference of ISR 2010 (41st Internationel Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, Munich, Germany, June 2010. VDE Verlag.

[6] J. E. Bobrow. Optimal robot path planning using the minimum-time criterion. 4(4):443–450, August 1988.

[7] J.E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

[8] X. Broquère, D. Sidobre, and I. Herrera-Aguilar. Soft motion trajectory planner for service manipulator robot. pages 2808–2813, Nice, France, September 2008.

[9] R. H. Castain and R. P. Paul. An on-line dynamic trajectory generator. 3(1):68–72, March 1984.

[10] F. Chaumette and S. A. Hutchinson. Visual servoing and visual tracking. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, chapter 24, pages 563–583. Springer, Berlin, Heidelberg, Germany, first edition, 2008.

[11] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. 2(1):13–30, 1983.

[12] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer, New York, NY, USA, first edition, 2007.

[13] Roy Featherstone and David E. Orin. Springer handbook of robotics. In Siciliano and Khatib [36], chapter 2, pages 35–65.

[14] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org` (accessed: Jan. 1, 2013), 2010.

[15] S. Haddadin, H. Urbanek, S. Parusel, D. Burschka, J. Roßmann, A. Albu-Schäffer, and G. Hirzinger. Real-time reactive motion generation based on variable attractor dynamics and shaped velocities. pages 3109–3116, Taipei, Taiwan, October 2010.

[16] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. pages 3248–3253, Nice, France, September 2008.

[17] J. M. Hollerbach. Dynamic scaling of manipulator trajectories. *ASME Journal on Dynamic Systems, Measurement, and Control*, 106(1):102–106, 1984.

[18] Lydia E. Kavraki and Steven M. LaValle. Springer handbook of robotics. In Siciliano and Khatib [36], chapter 5, pages 109–131.

[19] O. Khatib, E. Demircan, V. De Sapio, L. Sentis, T. Besier, and S. Delp. Robotics-based synthesis of human motion. *Journal of Physiology-Paris*, 103(3-5):211 – 219, 2009. <ce:title>Neurorobotics</ce:title>.

[20] Oussama Khatib. *Advanced Robotic Manipulation*. Stanford University, April 2012.

[21] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, first edition, January 2010.

[22] T. Kröger. Documentation of the fast research interface library, version 1.0. `http://cs.stanford.edu/people/tkr/fri/html` (accessed: Jun. 2, 2012), November 2011.

[23] T. Kröger. On-line trajectory generation: Nonconstant motion constraints. pages 2048–2054, Saint Paul, MN, USA, May 2012.

[24] T. Kröger and F. M. Wahl. On-line trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. 26(1):94–111, February 2010.

[25] KUKA Roboter GmbH, Zugspitzstraße 140, D-86165 Augsburg, Germany. *KUKA Lightweight Robot 4, Operating Instructions*, 2008. Issued: 10.07.2008 Version: BA LBR 4 V1 en.

[26] KUKA Roboter GmbH, Zugspitzstraße 140, D-86165 Augsburg, Germany. *KUKA Preliminary Documentation: Fast Research Interface (FRI)*, 2010. Issued: 25.02.2010.

[27] Robert S. Laramee. Shot put. `http://www.winslam.com/rlaramee/shotput/index.html` (accessed: Jan. 10, 2013), January 2013.

[28] S. Liu. An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators. In *Proc. of the seventh International Workshop on Advanced Motion Control*, pages 365–370, Maribor, Slovenia, July 2002.

[29] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. volume 3, pages 1399–1405, San Francisco, CA, USA, April 1986.

[30] D-24805 Hamdorf Germany Reflexxes GmbH, Sandknöll 7. Reflexxes motion libraries. `http://www.reflexxes.com` (accessed: Jan. 6, 2013), January 2013.

[31] G. Sahar and J. M. Hollerbach. Planning of minimum-time trajectories for robot arms. volume 2, pages 751–758, St. Louis, MO, USA, March 1985.

[32] G. Schreiber, A. Stemmer, and R. Bischoff. The fast research interface for the KUKA lightweight robot. In *Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications — How to Modify and Enhance Commercial Controllers at the IEEE International Conference on Robotics and Automation*, pages 15–21, Anchorage, AK, USA, May 2010.

[33] S.T. Seroyer, S.J. Nho, B.R. Bach, C.A. Bush-Joseph, G.P. Nicholson, and A.A. Romeo. Shoulder pain in the overhead throwing athlete. *Sports Health: A Multidisciplinary Approach*, 1(2):108–120, 2009.

[34] K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. 30(5):531–541, June 1985.

[35] K. G. Shin and N. D. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. 31(6):491–500, June 1986.

[36] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, Berlin, 2008.

[37] Simbios project SimTK. Tao dynamics engine. https://simtk.org/home/tao_de (accessed: Jan. 15, 2013), January 2013.

[38] J. Swevers, W. Verdonck, and J. De Schutter. Dynamic model identification for industrial robots. *Control Systems, IEEE*, 27(5):58 –71, oct. 2007.

[39] L. Villani, , and J. De Schutter. Force control. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, chapter 7, pages 161–185. Springer, Berlin, Heidelberg, Germany, first edition, 2008.

[40] Su-Hau Hsu Wankyun Chung, Li-Chen Fu. Springer handbook of robotics. In Siciliano and Khatib [36], chapter 6, pages 133–159.

[41] Inc. Wolfram Research. Mathematica edition: Version 8.0. Website of Wolfram Research, Inc., Champaign, Illinois http://www.wolfram.com/mathematica/ (accessed: Jan. 6, 2013), January 2010.

[42] Inc. Wolfram Research. Mathlink. http://reference.wolfram.com/mathematica/guide/InstallableMathLinkPrograms.html (accessed: Jan. 6, 2013), January 2010.

[43] Chia-Ju Wu. A numerical approach for time-optimal path-planning of kinematically redundant manipulators. *Robotica*, 12(05):401–410, 1994.